PCB101 — PCB Design with KiCad

Design the boards a chip sits on: schematics, footprints from datasheets, layout, protection, fab outputs in CI, review, and a bring-up board for a team chip.

UofT ASIC Team

Contents

1 PCB101 — PCB Design with KiCad 5

I 1. Reading a board

2 Getting started 11

3 What a board does for a chip 17

4 Tiny Tapeout's boards, part by part 21

5 Lab 01: Open a board you didn't design 27

II 2. Parts and packages

6 Schematics other people can read 39

7 Symbols, footprints and libraries 45

8 From die to package: pads, bond wires, QFN, QFP and BGA 53

9 Lab 02: A footprint from a datasheet 59

III 3. A board of your own

10 Layers and fabricator rules 71

11 Placement and routing 77

12 Pins, Pmods and planning a pinout 83

13 Parts you can solder by hand 87

14 Lab 03: A Pmod for one project's pins 91

IV 4. Protection and bring-up

15 Protecting the chip's I/O 101

16 Supplies, back-powering and ESD 107

17 Measuring current . . . 113
18 Bringing up a chip . . . 117
19 Lab 04: Protect and plan . . . 123

V 5. Outputs and CI

20 Fabrication outputs . . . 133
21 Outputs in CI . . . 141
22 KiCad files in git . . . 147
23 Lab 05: Fab outputs in CI . . . 153

VI 6. Review and ordering

24 Reviewing and ordering a board . . . 167
25 Lab 06: Review a seeded board . . . 171

VII 7. Capstone

26 Boards for chips: carriers, motherboards and characterization boards . . . 179
27 Capstone: A board for a team chip . . . 183

10.1 F.Cu, the DIP dock's top copper . . . 72
10.2 B.Cu, the DIP dock's bottom copper . . . 72

1. PCB101 — PCB Design with KiCad

In this course you design the boards a chip sits on, in KiCad. You start by reading the boards that Tiny Tapeout makes for its chips, including the breakout that the team's SKY130 chips arrive on. Next you draw a footprint from a datasheet's package drawing, for a chip package that KiCad's library doesn't have. Then you draw a board of your own, a small Pmod for one project's pins. In the next two labs you add protection to it, write its bring-up plan, and set up GitHub Actions to build its fabrication files on every push. After that you review a Pmod like yours that we planted faults in, and get a fabricator's quote for your own board without ordering it. In the capstone you take a board for one of the team's chips to the point where it could be ordered.

PCB101 is part of the UofT ASIC Team education materials, published at https://uoftasic.com/pcb101/. Unlike the analog and digital courses, it doesn't run in the workbench that you set up in IC101. You install KiCad 10.0 on your own computer, and the course's checks run in KiCad's own Docker image. Getting started explains why, and what to install on Windows, macOS and Linux.

Who this course is for

The course is for anyone on the team, from the analog track or the digital one, who will bring up, measure or demonstrate one of the team's chips. Today that means the people working on SimProc, CrossWalk, Sentry and HART-1, and later it means whoever brings up the next shuttle's chips. The team has no board designs of its own yet, so the course covers the boards that the team's chips will need. The first of them is the Pmod you design in Labs 03 to 06, which is the simplest kind of board you can make for a Tiny Tapeout chip. The carriers, motherboards and characterization boards that come after it are described in Boards for chips, on the last Read row of the course path.

You don't need to have designed a board before. You do need the circuits you learned in class, because the course explains decoupling with the voltage that a fast current pulse develops across a wire's inductance, and filtering with an RC time constant. It explains protection with the current that a series resistor lets through a clamp diode, and with the voltage divider that the same resistor forms with a pull-down.

Track	Core: for both the analog and the digital track, after IC101 and IC102
Prerequisites	IC101, then IC102 (either version), and circuits from class. AD103 helps with the analog capstone brief and DD102 with the digital one.
Tools	KiCad 10.0 on your own computer; Docker, which you set up in IC101, for the checks; Python 3, for the script that fetches Tiny Tapeout's boards; git and a GitHub account.
Checks	`kicad/kicad:10.0.5`, pinned by digest. Every count and message that a lab page quotes from KiCad or from one of the course's checks comes from a run in that image. The exceptions are KiBot's output in Lab 05, which comes from KiBot's own image (with KiCad 10.0.5), also pinned by digest, and Lab 01's comparison with KiCad 10.0.8.
Total time	About 22 hours, self-paced. The course path rows below add up to 21 h 55 min.

IC102 is a prerequisite because from Lab 03 on you keep your board in a repository of your own on GitHub. You commit to it in every lab after that, and in Lab 05 you add a GitHub Actions workflow that checks it on every push.

The capstone

The capstone has two briefs, and you pick one. Both are 2-layer boards. Our reference board for Brief D is 64 × 32 mm, and the one for Brief A is 50 × 30 mm.

- **Brief D** is a bring-up Pmod for SimProc, Sentry or a digital Tiny Tapeout project of your own. It plugs into the demo board's I/O sockets, and if you built the SimProc Pmod in Labs 03 and 04, you can carry that board through.
- **Brief A** is a source board for CrossWalk or an ASIC Zoo cell. It plugs onto the demo board's analog header and drives one analog pin from a known, filtered source.

For either brief, you put the KiCad project, a design note (`DESIGN.md`) and a bring-up plan (`BRINGUP.md`) in your own GitHub repository, and hand in the repository's URL and the hash of a commit that passes the capstone's check and has a green CI run. The course doesn't require you to order the board or to bring it up. Both depend on the team's budget and on there being a chip to put on the board, so they're a separate step.

Course path

Work down the table in order. A Read row points you to one or more guide pages, which have nothing for you to run. A Do row is a lab, where you run things, make something of your own and check your results. Each lab's "Before you start" section repeats which pages it expects you to have read.

#		What	Time
1	Read	Getting started · What a board does for a chip · Tiny Tapeout's boards, part by part	45 min
2	Do	Lab 01: Open a board you didn't design. Install KiCad 10.0, open Tiny Tapeout's SKY130 breakout, find which parts are fitted, trace the chip's three supplies from the connectors to its pins through 0 Ω links and, for VCCD, the one fitted regulator, and explain each kind of error that ERC and DRC report and what most of the warnings mean.	1 h 30 min
3	Read	Schematics other people can read · Symbols, footprints and libraries · From die to package	1 h
4	Do	Lab 02: A footprint from a datasheet. Work out the pads for a QFN package that KiCad's library doesn't have, from the drawing in TI's datasheet, then draw its footprint and a symbol for the chip	2 h

#		What	Time
		that comes in it, and check both against your own numbers and on a small board.	
5	Read	Layers and fabricator rules · Placement and routing · Pins, Pmods and planning a pinout · Parts you can solder by hand	1 h
6	Do	Lab 03: A Pmod for one project's pins. Start your board repository, then draw and lay out a 2-layer Pmod for a Tiny Tapeout project until ERC, DRC and parity are clean and its connections match the project's pin list.	4 h
7	Read	Protecting the chip's I/O · Supplies, back-powering and ESD · Measuring current · Bringing up a chip	1 h
8	Do	Lab 04: Protect and plan. Put a buffer and a resistor on each UART line of your Lab 03 board, add ESD protection and a current-sense point, and write its `BRINGUP.md` with the checks before power and an expected reading for every step.	2 h
9	Read	Fabrication outputs · Outputs in CI · KiCad files in git	45 min
10	Do	Lab 05: Fab outputs in CI. Add a workflow to your board repository that fails on any ERC or DRC error or any parity item, and builds the fabrication files, review images, and KiBot's interactive BOM and design report, then break the board on purpose and fix it.	1 h 40 min

#		What	Time
11	Read	Reviewing and ordering a board	30 min
12	Do	Lab 06: Review a seeded board. Find the faults we planted in a board that comes with the course, using the course checklist, then record a fabricator's quote and DFM report for your own board.	1 h 15 min
13	Read	Boards for chips	30 min
14	Do	Capstone: A board for a team chip. Take a bring-up Pmod for a digital project, or a small source board for an analog pin, to the point where it could be ordered, with a design note, a bring-up plan and a green CI run in your own repository.	4 h

Getting help

Ask in the team Discord, where no question is too basic. Getting started says what to include so that someone can answer quickly.

I

1. Reading a board

2 **Getting started** 11
2 Requirements 11
2 KiCad and the workbench 11
2 Which version to install 11
2 Installing on Windows 13
2 Installing on macOS 13
2 Installing on Linux 13
2 Your KiCad and the course's checks 14
2 Course layout 14
2 Getting help 15
2 Check yourself 15
2 Answers 15

3 **What a board does for a chip** 17
3 The devkit 17
3 The chip's pin limits 18
3 Hardware for one project 18
3 Four kinds of board 19
3 Check yourself 19
3 Answers 20

4 **Tiny Tapeout's boards, part by part** . 21
4 The SKY130 QFN breakout 21
4 Unfitted footprints and DNP 23
4 Links and jumpers 23
4 The demo board 24
4 The DIP dock 25
4 Two layers and four 25
4 Tiny Tapeout's errata 25
4 Check yourself 26
4 Answers 26

5 **Lab 01: Open a board you didn't design** 27
5 Before you start 27
5 Part 1: Install KiCad and get the boards 27
5 Part 2: Fitted and unfitted parts 28
5 Part 3: The supply rails 30
5 Part 4: ERC and DRC 31
5 Expected results 34
5 Finishing 35
5 Going further (optional) 35
5 Answers 35

2. Getting started

In PCB101 you draw everything in **KiCad**, a free, open-source suite for schematics and printed circuit boards. It has a Schematic Editor, a PCB Editor, a Symbol Editor and a Footprint Editor, and a command-line program, `kicad-cli`, that runs the same checks and exports without opening a window. You'll use all five.

IC101 and IC102 use the workbench, and so do the analog and digital courses. In PCB101 you install KiCad on your own computer instead and do all your drawing there, while the course's checks run `kicad-cli` in KiCad's own Docker image. This page explains why it's done that way, which version of KiCad to install and where to get it on Windows, macOS and Linux, and how the numbers you see in your own KiCad relate to the ones on the lab pages. You won't install anything while you read it. You'll do the installing in Lab 01, Part 1.

The facts about KiCad's downloads on this page come from kicad.org, read on November 16, 2026, and each section links the page it comes from. KiCad updates those pages with every release, so if a version number there has moved on since, believe kicad.org.

Requirements

- **A computer that runs KiCad 10.** KiCad's system requirements ask for a 64-bit processor, 1 GB of RAM (">2G RAM recommended"), up to 10 GB of disk space, and a graphics card with OpenGL 2.1 or greater that supports hardware shaders. A screen of 1280 × 1024 works, but KiCad recommends 1920 × 1080 or more, because at lower resolutions some toolbar buttons may be hidden, and then you have to reach those commands through the menus.
- **Docker**, which you set up in IC101. The first time a check runs, it pulls the course's KiCad image, which is a download of 808 MB and takes 2.6 GB on disk once it's unpacked. Lab 05 also pulls KiBot's image, a download of 1,347 MB that takes 3.9 GB on disk.
- **git, Python 3 and a GitHub account**, from IC101 and IC102. Lab 01 uses Python 3 to run the script that fetches Tiny Tapeout's boards. From Lab 03 on you keep your board in a repository of your own on GitHub, and in Lab 05 you add a GitHub Actions workflow that checks it on every push.

KiCad and the workbench

The workbench image doesn't include KiCad. When we checked `asic-edu/gui-rig:2026.08`, it had no `kicad`, `pcbnew` or `kicad-cli`, and the student account in it couldn't install packages. KiCad's own Docker image doesn't help with drawing either. KiCad's Docker page says its images are "intended for usage of the kicad-cli binary" and that "Using the GUI environment via Docker is not supported."

So you draw in a KiCad installed on your own computer, which is the supported way to run the editors, and the checks run `kicad-cli` in the official image, which is what KiCad made that image for. The checks don't need `kicad-cli` or `make` on your own machine. They need Docker, and bash to run the course's scripts, which on Windows means running them in WSL.

Which version to install

Install KiCad 10.0.5. Every count, message and screenshot in the course was recorded with KiCad 10.0.5, which is also the version the course's checks run, so with 10.0.5 on your own computer the windows and numbers you see match the pages as closely as they can. It isn't the newest release. On November 16, 2026, every platform's download page offered 10.0.8 (KiCad 10.0.0 came out on March 20, 2026), and kicad.org keeps the older releases on a separate page, which the section for your platform below points to. If you install KiCad through a Linux distribution's packages, or you already have a later 10.0 release, you can do the course with that; the end of this section says what to expect.

The major version matters because a KiCad file doesn't open in an older major version than the one that saved it. KiCad's Linux packages page puts it this way: every year a new major release "comes out, bringing new features and bug fixes, but also usually bringing file format changes that prevent downgrading. In

other words, once you update to 9.x and start modifying a project, you will no longer be able to open that project in any 8.x version." The same is true of 10.x and 9.x. When we gave a board saved by KiCad 10 to `kicad-cli` 9.0.9, it printed `Failed to load board` and exited with status 3. Everything you make in this course is saved in the 10.0 format, so KiCad 9 can't open your work.

KiCad 10 does open boards saved by KiCad 9, and Tiny Tapeout's SKY130 breakout, the first board you open in Lab 01, is one of those. When you open it, KiCad says so in a bar across the top of the canvas:

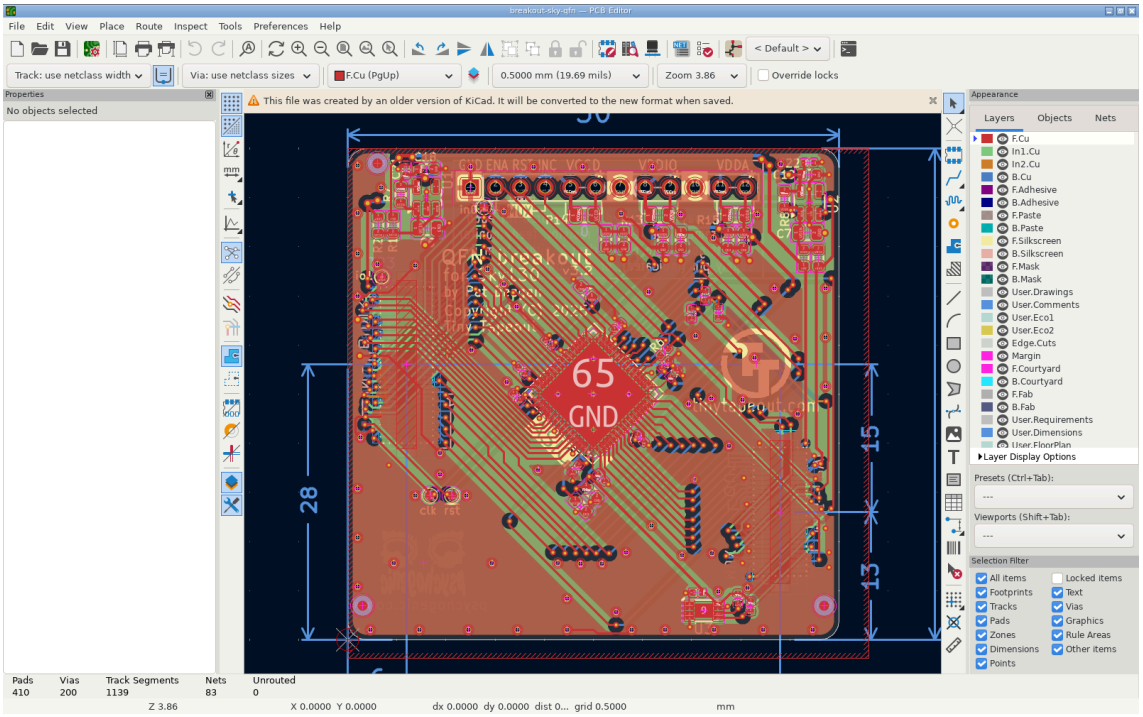


Figure 2.1: The PCB Editor in KiCad 10.0.5 with Tiny Tapeout's SKY130 QFN breakout open

Tiny Tapeout's SKY130 QFN breakout, rev 3.2 (`TinyTapeout/breakout-pcb` at 91a2bf1, Apache-2.0), open in the PCB Editor of KiCad 10.0.5. The bar under the toolbars says "This file was created by an older version of KiCad. It will be converted to the new format when saved." The board is 50 × 50 mm. It has four copper layers, so the Appearance panel on the right lists the two inner layers, In1.Cu and In2.Cu, as well as F.Cu and B.Cu.

The conversion happens only if you save. Once you have saved the board, it's a 10.0 file and KiCad 9 can't open it any more, which is worth knowing before you save someone else's project. Don't save the boards you fetch for the labs at all, because the course's fetch script then refuses to update them until you put the originals back with `--force`.

Within the 10.0 series, the patch releases (10.0.1, 10.0.2 and so on) share a file format. The three we checked in KiCad's source, 10.0.0, 10.0.5 and 10.0.8, all write boards as version 20260206 and schematics as version 20260306, so any 10.0 release opens files saved by another. KiCad encourages everyone to update to the latest bugfix release regularly. Patch releases can still differ in what ERC and DRC report, though. On November 16, 2026 we ran the course's checks in both 10.0.5 and 10.0.8, in KiCad's Docker images, and compared the counts:

- On the course's own reference boards for Labs 03 and 04 and the capstone, as they were on November 16, every count was the same. Lab 04's board and the capstone's Brief D board have gained a buffer since, and we haven't run 10.0.8 on them again.
- On Tiny Tapeout's SKY130 breakout, the KiCad 9 board you open in Lab 01, both releases found the same errors, 10 from ERC and none from DRC. The warning counts differed: 10.0.5 reported 97 ERC and 120 DRC warnings, and 10.0.8 reported 103 and 123. 10.0.5 also reported 37 schematic parity warnings that 10.0.8 didn't, every one of them about a part's Datasheet field, which is ~ on the board and empty in the schematic.
- On a copy of one of KiCad's demo boards with a GND track laid across a VCC track, both releases found the tracks crossing and a solder mask bridge. 10.0.5 also reported the GND track as shorting to a pad on a third net, so it counted three DRC errors, and 10.0.8 counted two.

So if your own KiCad is a later 10.0 release, expect the same results on boards like the ones you'll draw, different warning counts on older boards such as Tiny Tapeout's, and now and then a different number of errors on a board with something wrong with it. The course's checks always run 10.0.5 in the pinned image, whatever you have installed, and their results are the ones the labs go by.

Installing on Windows

KiCad's Windows page says "KiCad supports Windows 10 and 11", and it offers two installers for each release:

- `kicad-10.0.8-x86_64.exe`, for the Intel and AMD processors that most Windows laptops have;
- `kicad-10.0.8-arm64.exe`, for a Windows computer with an ARM processor.

Those are for the current release, 10.0.8. For older releases such as 10.0.5, the same page says "Previous releases should be available for download on: https://downloads.kicad.org/kicad/windows/explore/stable", and the list there had `kicad-10.0.5-x86_64.exe` and `kicad-10.0.5-arm64.exe`.

Each installer is code-signed, and Windows checks the signature when you run it. The page says a valid signature names KiCad Services Corporation as the signer, and links a guide to checking that yourself.

If you're still on Windows 10, read KiCad's system requirements as well. Its table gives Windows 10 a support date of 14 October 2025, and KiCad's policy is that once an operating system's maker stops releasing updates for it, KiCad isn't tested on it any more, and a bug has to be reproduced on a supported system before the KiCad team will look at it. KiCad may well keep working on Windows 10, but Windows 11 is the supported choice.

Installing on macOS

KiCad's macOS page has one download for all Macs, a disk image named `kicad-unified-universal-10.0.8.dmg`, and says "The current stable release of KiCad is supported on macOS 12 and newer." For older releases it says "Previous releases are available for download on: https://downloads.kicad.org/kicad/macos/explore/stable", and the 10.0.5 disk image there is `kicad-unified-universal-10.0.5.dmg`.

The system requirements page is stricter than the macOS page. It says KiCad supports the macOS versions that Apple currently supports, which it takes to be the three most recent major releases, and its table gives approximate end-of-support dates of 1 November 2025 for macOS 13, 1 November 2026 for macOS 14 and 1 November 2027 for macOS 15. By that page's own dates, only macOS 15 was still supported on November 16, 2026, and its table lists no newer macOS at all, so it looks out of date. If your Mac runs a recent macOS, both pages cover you. On an older one, KiCad still offers you the download, but a bug you hit there may have to be reproduced on a supported version before the KiCad team looks at it.

Installing on Linux

KiCad publishes its own Linux build on its Linux page: "The current stable release of KiCad is available as an AppImage for Linux x86-64 systems." It comes in two versions, each packed in a `.tar` file:

- Full, `kicad-10.0.8-x86_64.AppImage.tar`, which includes the 3D packages (the 3D models of parts that the 3D viewer draws);
- Lite, `kicad-10.0.8-x86_64-lite.AppImage.tar`, without them.

For older releases, the page says "Previous releases are available for download on: https://downloads.kicad.org/kicad/linux/explore/stable", where the 10.0.5 versions of the same two are `kicad-10.0.5-x86_64.AppImage.tar` and `kicad-10.0.5-x86_64-lite.AppImage.tar`.

You can also install KiCad through your distribution. The Linux packages page lists what KiCad supports:

- Ubuntu: KiCad runs its own PPA for each release series, and for 10.0 it's `ppa:kicad/kicad-10.0-releases`. KiCad recommends it over Ubuntu's own package, because "the latter is usually outdated". The system requirements give Ubuntu 24.04 LTS support until April 2029, and list Ubuntu 22.04's support as ending in May 2025.
- Fedora: the `kicad` package in your Fedora release, or the newest stable build from KiCad's Copr repository, `@kicad/kicad-stable`.

- Flatpak: KiCad on Flathub, which the page calls "our recommended way to install the latest version of KiCad on any distribution other than Ubuntu and Fedora".
- Debian: a package in Trixie and in Trixie's backports. The page doesn't say which KiCad version each one is, so check that it's 10.0 before you install it.

Arch Linux, FreeBSD, Gentoo, openSUSE, Linux Mint and Guix are listed too, as platforms the KiCad team doesn't officially support.

The page heads the PPA's commands "10.0.8 Stable Release", and it doesn't say how to get an older 10.0 release from any of these. The nearest it comes is for Ubuntu, where it says "Older stable releases of KiCad are available in other similarly-named PPAs" and gives `sudo apt install kicad=<version>` for going back from the testing PPA to a stable release. So if you install from your distribution, you'll most likely have a later 10.0 release than the course's, and the end of Which version to install says what that changes.

Whichever way you install it, run KiCad in an X11 session if you can. The same page says "The KiCad team does not support running KiCad on Wayland due to the much higher incidence of related bugs and problems", and that bug reports about freezes, crashes, window size and position, focus and similar problems "will not be investigated unless they can be reproduced on an X11 system".

Your KiCad and the course's checks

Every lab's check runs through a script in `labs/common/` that starts `kicad/kicad:10.0.5`, named by its digest (`sha256:182c8005cb77…`, in full in `labs/common/kicad-image.txt`) so that it's the same build on every machine, and runs the check inside it. In every lab except Lab 06, the check includes the course's board check, which uses `kicad-cli` to run ERC on the schematic and DRC on the board, compares the board with the schematic, and prints the counts, followed by PASS or FAIL. Lab 01 runs it on Tiny Tapeout's breakout, Lab 02 on a small board with your footprint on it, and Labs 03 to 05 and the capstone on your own board. Lab 05's check also runs KiBot, in KiBot's own image. Lab 06's check reads your review of a seeded board and your fabricator's quote, and doesn't run ERC or DRC. Each lab shows you the command.

Your own KiCad can run the same ERC and DRC from its menus, and you'll do both in Lab 01. The errors it finds should be the same as the checker's, but the warning counts can differ, because many of KiCad's warnings compare the parts in your design with the libraries KiCad knows about, and which libraries those are depends on how KiCad was set up on that machine. Here is the same board, Tiny Tapeout's SKY130 breakout, checked in two setups:

	ERC errors	ERC warnings	DRC errors	DRC warnings
The course's checker	10	97	0	120
The same image with no library tables at all, and without Tiny Tapeout's own library beside the board	10	236	0	130

Only the warning counts changed, so the labs give warning counts only for the checker's setup, and when you look at a board in your own KiCad they ask you about kinds of warning rather than numbers. If your KiCad shows a different number of errors from the lab page, though, something is different, and it's worth finding out what.

Course layout

The course path on the home page lists every page in the order you should work through them. Each row is one of two kinds:

- A Read row points to guide pages like this one, which explain the ideas with figures and worked examples. You don't need KiCad open to read them.
- A Do row points to a lab, and the labs are where you run things.

Each lab is split into numbered parts with numbered steps. Most parts open with the question they answer and end with a check, which says what you should see if what you did worked. In at least one part of every lab you make something yourself, such as a footprint, a schematic, a routed board or a bring-up plan, and the lab's final check runs on what you made, never on files the course ships.

Some steps ask you to write down what you expect before you run something, so that you have a prediction of your own to compare the result with. When a step asks a question, its answer is in a collapsed block right under it; try to answer it yourself before you open the block. (In the PDF version of a page, the answers are collected at the end of the page instead.)

After its last part, every lab has a Finishing section that tells you how you'll know you're done, asks any questions you can answer from your own work, and says what to submit. Then comes a "Going further (optional)" section with extra reading and experiments. None of the required parts depends on it, and the lab's time estimate leaves it out.

The files each lab uses are in the course's GitHub repo under `labs/`, apart from Tiny Tapeout's boards, which the repo doesn't store. A script fetches those from Tiny Tapeout's own repositories at fixed commits, so you always get the same version of each board, and Lab 01 has you run it.

Getting help

The course is self-paced, but you don't have to work through it alone.

- **Ask in the team Discord.** No question is too basic.
- **Quote the exact message, and say where you saw it**: a dialog in KiCad, the checker's output, or a report file. Paste the text rather than paraphrasing it, along with the command you ran, if there was one.
- **Say which KiCad you used.** For your own install, that's the version in KiCad's About dialog. For the checker, it's the `kicad` and `image` lines at the top of its output.
- **Post a screenshot.** A wiring or placement problem that's obvious in a picture can be very hard to describe in words.
- **If a page doesn't match what you see, tell us.** We ran the commands in the labs before writing them down, apart from the pushes to GitHub, which we couldn't make from the machine we wrote the course on, and we copied every output and count from a real run. So a mismatch means something has changed, and we want to know.

Check yourself

Answer these before you start Lab 01.

1. A friend on the team has KiCad 9.0 installed and wants to look at your Lab 03 board. What happens when they open it?

   Answer: A1, in Answers at the end.

2. You run DRC in your own KiCad on the board from a lab, and it reports a different number of warnings from the lab page, but the same number of errors. Is something wrong?

   Answer: A2, in Answers at the end.

3. Why do the checks run in Docker when you draw in a KiCad installed on your own computer?

   Answer: A3, in Answers at the end.

4. You open Tiny Tapeout's SKY130 breakout and KiCad says the file was created by an older version. What changes if you save it?

   Answer: A4, in Answers at the end.

Next: What a board does for a chip.

Answers

A1. Check yourself

It doesn't open. Your board is saved in KiCad 10.0's file format, and KiCad 9 can't read a newer format. When we gave a 10.0 board to `kicad-cli` 9.0.9, it printed `Failed to load board`. Your friend needs KiCad 10.0, or you can send them the PDF and image exports you'll make in Lab 05.

A2. Check yourself

Probably not. Many warnings compare the board's footprints with the libraries your KiCad has set up, so the count depends on your installation. The lab page's counts come from the checker's setup. The error counts stayed the same in every library setup we tried, so a different error count would be worth looking into.

A3. Check yourself

KiCad's Docker image is made for `kicad-cli`, and KiCad doesn't support running its editors in Docker, so you draw in KiCad on your own computer. The checks use the image because it's pinned to one build by its digest, so they give the same counts on your laptop, on everyone else's and in CI.

A4. Check yourself

KiCad converts it to the 10.0 format when it saves, and from then on KiCad 9 can't open that copy. The course's fetch script also refuses to update a board you've changed until you put the original back with `--force`.

3. What a board does for a chip

This page answers: *the team's chips arrive already sitting on a board. What is left for us to design?*

A chip on its own can't do much. It needs its supplies at the right voltages, a clock, some way for you to set its inputs and read its outputs, and some protection from whatever you connect to its pins. Boards provide all of that, and for a Tiny Tapeout chip the first boards are Tiny Tapeout's own. The next page goes through the same boards part by part, and in Lab 01 you open one in KiCad.

On November 16, 2026, we read the Tiny Tapeout web pages that this page quotes. The board files and READMEs are the ones at the commits the course fetches, which are listed in `labs/common/upstream.json`.

The devkit

Tiny Tapeout doesn't sell bare chips. Its FAQ says "We don't ship bare dies either", and describes what you get instead: "The devkit consists of two PCBs (printed circuit boards) - a 'demo board' and a 'breakout board'. The breakout board contains the shuttle's ASIC, and connects to the demo board via a set of pins." The same FAQ says "You only get 1 chip", and that if you want more you have to order more devkits, so be careful with the one on your bench.

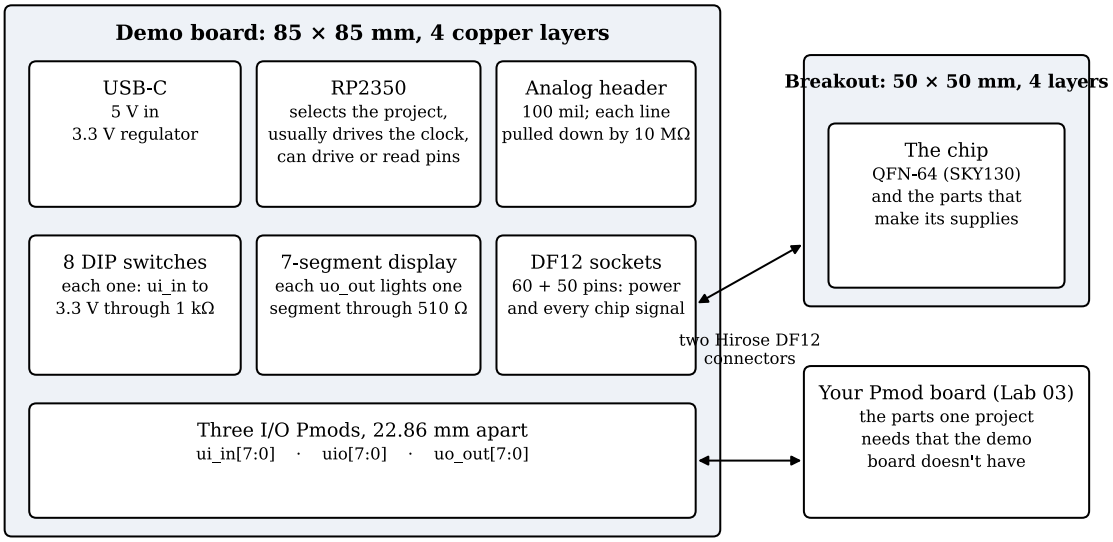


Figure 3.1: The parts of a Tiny Tapeout devkit, and where a board of your own plugs in

A Tiny Tapeout devkit for a SKY130 shuttle. The labels come from the demo board's README and board file (`TinyTapeout/tt-demo-pcb` at bbf3dd1), the breakout's board file (`TinyTapeout/breakout-pcb` at 91a2bf1) and Tiny Tapeout's guide to the demo board. Drawn by `scripts/figures/devkit_blocks.py`.

The **breakout** is a small board that carries the chip and the few parts the chip needs right beside it. On a SKY130 shuttle it's 50 × 50 mm with four copper layers, and the chip is in a QFN-64 package, which From die to package takes apart. A regulator on the breakout makes the chip's 1.8 V core supply from the demo board's 3.3 V, and the chip's I/O and analog supplies take the 3.3 V through 0 Ω links with no regulator in between, which you trace for yourself in Lab 01. It plugs into the demo board through two Hirose DF12 connectors on its underside, one with 60 pins and one with 50, which the FAQ describes as "2 connectors where you get all of the signals of the chip."

The **demo board** is everything else. Its README lists three ways to reach a project: "directly, using the input DIP switches and 7-segment display", "via breadboard or extension boards through PMODs" (a

Pmod is a small add-on board on a 2 × 6 pin header; Digilent defined the format, and the README links its specification), or "by interacting through the on-board RP2". The RP2 is an RP2350 microcontroller, and the README says it "is responsible for selecting projects, by controlling the project MUX that's on the ASIC and, under most circumstances, providing the clock for the projects." That multiplexer is there because a shuttle's chip carries many projects at once (TT08′s carries 135, according to Tiny Tapeout's news post about it), and the RP2350 connects one of them to the pins. The demo board takes its power from a USB-C connector, and Tiny Tapeout's demo board guide says that, running the factory test project, the kit draws "around 180-200 milliAmps, on the 5V supply". Before you power a board of your own for the first time, Bringing up a chip has you power the kit without it and compare the kit's current with that figure.

The chip's pin limits

Every board you design for these chips starts from the limits of the chip's pins, so it's worth having them in one place. Tiny Tapeout's GPIO page gives each project 26 digital pins:

Pins	Count	Direction
`clk`	1	input, the clock
`rst_n`	1	input, active-low reset
`ui_in[7:0]`	8	input
`uo_out[7:0]`	8	output
`uio[7:0]`	8	bidirectional

For SKY130 chips the same page lists the limits of the I/O pads from the documentation of the pad cell the chip uses, and it calls the two frequencies SkyWater's official ratings:

Parameter	Limit
Maximum output frequency	33 MHz
Maximum input frequency	66 MHz
Drive strength (source or sink)	4 mA
I/O supply voltage	1.71 V to 5.5 V

Under the table, the page adds "The demo board provides 3.3V IO supply voltage. The input pins are not 5V tolerant." Signals also pass through the multiplexer on their way to and from a project, and the page reports a worst-case round trip of 20 ns, measured on Tiny Tapeout 3.5 silicon.

Almost every board in this course has to allow for two of those limits: the 4 mA drive strength, and inputs that aren't 5 V tolerant. A 4 mA output is a logic-level output, enough to drive another chip's input or a small LED through a resistor (the demo board's display segments are driven through 510 Ω), and anything that needs more current needs a transistor or a buffer on your board. And a 5 V logic output can't drive the inputs directly, nor can anything else that can rise above the chip's 3.3 V I/O supply. Protecting the chip's I/O covers what happens then and what to put in the way.

Analog pins have limits of their own. The analog spec says that for SKY130 the path between a pad and a project's analog pin has a resistance under 500 Ω and a capacitance under 5 pF, with a maximum current of 4 mA, and that an analog switch (`tt_asw_3v3`) sits in that path. The demo board guide says SKY130 chips "have 12 pins dedicated to analog, while projects may use up to 6 of these", that the demo board brings them to a 100 mil header, and that "Each of these lines is pulled down with a large 10 MegaOhm resistance". So a source on a board of yours that drives an analog pin also drives the 10 MΩ pull-down, and it charges up to 5 pF through up to 500 Ω of path, the switch included.

Hardware for one project

The demo board is general on purpose. It gives every project the same switches, the same display, the same three Pmods and the same microcontroller, and that covers a lot of projects, but it can't give one particular project hardware that only that project needs.

SimProc, the team's processor on the TTSKY25b shuttle, is a good example. Its `info.yaml` puts its UART receive line on `ui[0]` and its transmit line on `uo[0]`, its source sets all eight `uio` pins as inputs and uses them to set the UART's baud rate, and its documentation says "Send commands using the UART RX pin (using a USB to UART conveter), receive data through the UART TX pin" and "Set UIO7-UIO0 to set the BAUD rate of the internal UART module." Now look at the demo board's README, which has a table of the RP2350 pin behind each project pin. `uo_out[0]` goes to GPIO33, which the table lists as a hardware UART receiver (`UART0.rx`), so the RP2350 could listen to SimProc. `ui_in[0]` goes to GPIO17, and the table lists no UART function for it. Getting a UART onto that pin, and setting the eight baud-rate inputs, would be a job for the demo board's firmware. A small board with a header for a USB-to-UART adapter on `ui[0]` and `uo[0]`, and eight switches on `uio[7:0]`, does the same job with no firmware at all, and that's the board you draw in Lab 03.

The same README warns that the three I/O Pmods give "access to all the pins, but extension boards will often need to span at least two, and sometimes three, distinct headers." The SimProc board spans all three, because it uses an input, an output and the eight bidirectional pins.

Four kinds of board

The boards a chip team builds around Tiny Tapeout chips fall into four kinds. The order below is the order we expect the team to need them in, which is a plan for this course rather than a rule.

1. **A Pmod for one project's pins.** It plugs into the demo board's I/O Pmods and adds what one project needs: connectors for an adapter, switches, test points and protection. It's a small 2-layer board, and it's what you draw in Labs 03 and 04 and, for a digital project, in the capstone.
2. **A board for an analog pin.** It drives one of the chip's analog pins from a known, filtered source, or measures what comes out of one, through the demo board's analog header. The capstone's analog brief, Brief A, is a small board of this kind. A full characterization board for an analog block needs a precision reference and a way to capture codes, which makes it a bigger design than anything in this course.
3. **A motherboard for the breakout.** This replaces the demo board, and it's the route Tiny Tapeout suggests to anyone designing their own board: "we recommend that you design your PCB to interface with our breakout board", says the FAQ. A motherboard has to supply everything the demo board did. The demo board guide says that "All the I/O expects 3.3V logic levels", that the 5 V from USB is also passed to the breakout, and that no breakout so far actually needs it. Mating with two fine-pitch DF12 connectors is harder than plugging into a Pmod, and Tiny Tapeout's own breakouts and demo board use four copper layers.
4. **A board for a chip from another kind of shuttle.** Not every chip comes in a QFN. The FAQ says that some shuttles, "such as our GF180-based ones", don't come with a packaged chip: "they instead have the die bonded to a PCB directly (called chip-on-board, or COB)". On those shuttles the PCB is a small board shaped like a DIP chip, and it plugs into Tiny Tapeout's DIP dock, a 2-layer board that the next page shows. From die to package covers what bare dies, wire bonds and packages mean for a board. Boards for chips, the last reading row on the course path, comes back to the analog board, the motherboard and this kind.

Check yourself

1. A project wants to light an LED at 10 mA straight from `uo_out[3]`. Can the pin do it?

   Answer: A1, in Answers at the end.

2. A sensor module has 5 V logic outputs. Can you wire one straight to `ui_in[2]` through the demo board's input Pmod?

   Answer: A2, in Answers at the end.

3. The team wants a spare chip for bring-up, in case the first one is damaged. What does it have to buy?

   Answer: A3, in Answers at the end.

4. A source with 10 kΩ of output resistance drives a SKY130 analog pin through the demo board's analog header. Using the limits on this page, and counting only the chip's own capacitance, what is the longest the time constant can be, and how far does the 10 MΩ pull-down pull the pin's voltage below the source's?

   Answer: A4, in Answers at the end.

Next: Tiny Tapeout's boards, part by part.

Answers

A1. Check yourself

No. The GPIO page rates the SKY130 pads at 4 mA, source or sink. Your board needs a transistor or a buffer between the pin and the LED, or an LED that's bright enough at the current the pin can give.

A2. Check yourself

No. The demo board runs the chip's I/O at 3.3 V, and the GPIO page says "The input pins are not 5V tolerant." The signal needs translating down to 3.3 V on your board first.

A3. Check yourself

A second devkit. Tiny Tapeout ships each chip on a breakout, one chip per devkit, and doesn't sell bare dies.

A4. Check yourself

The resistance is at most 10 kΩ + 500 Ω = 10.5 kΩ and the chip's capacitance at most 5 pF, so the time constant is at most 10.5 kΩ × 5 pF ≈ 53 ns. The header and any wiring add capacitance of their own, so the real figure is larger. For the steady-state error, the pull-down and the source's 10 kΩ form a divider (the chip's 500 Ω carries no current once the pin has settled, so it drops nothing), and the header sits at 10 MΩ / (10 MΩ + 10 kΩ) of the source's voltage. That's about 0.1 % low, or about 1 mV for a 1 V signal.

4. Tiny Tapeout's boards, part by part

This page answers: *what is actually on the boards a Tiny Tapeout chip arrives on, and how do you read a board that someone else designed?*

Tiny Tapeout publishes its boards as KiCad projects under the Apache 2.0 licence, so you can open the same files the boards were made from. They're the best boards to read first, because they were built and shipped, and their files record the decisions a board designer makes for a chip, in KiCad's own format. The course fetches them from Tiny Tapeout's repositories at fixed commits, with a script you run in Lab 01, so the boards on this page are the versions you'll open.

Board	Repository and folder	Title block	Size	Copper layers	Saved by
SKY130 QFN breakout	`breakout-pcb/breakout-sky-qfn`	rev 3.2, 2025-10-04	50 × 50 mm	4	KiCad 9
IHP QFN breakout	`breakout-pcb/breakout-ihp-qfn`	rev 1.2, 2026-08-18	50 × 50 mm	4	KiCad 10
Demo board	`tt-demo-pcb`	"Tiny Tapeout Demoboard v3", rev 3.3, 2026-06-23	85 × 85 mm	4	KiCad 9
DIP dock	`breakout-pcb/dip-dock`	rev 1.0, 2026-10-19	85 × 85 mm	2	KiCad 10

The sizes are the extent of each board's outline, the copper layers are the ones each board file defines, and "saved by" is the `generator_version` in the board file. `breakout-pcb` is at commit 91a2bf1 and `tt-demo-pcb` at bbf3dd1.

The SKY130 QFN breakout

This is the board that carries a SKY130 chip, so most of the team's chips will arrive on it, and it's the board you open and read in Lab 01.

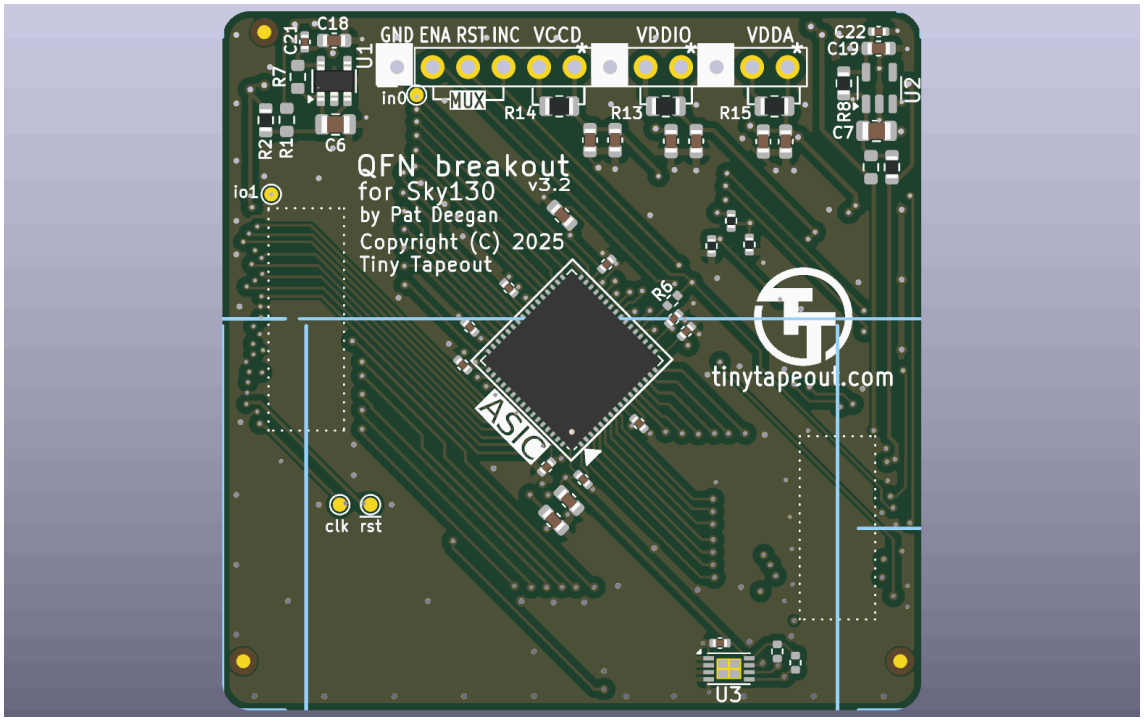


Figure 4.1: A 3D render of the top of Tiny Tapeout's SKY130 QFN breakout

The top of the SKY130 QFN breakout, rendered by `kicad-cli 10.0.5` from the board file, with KiCad's 3D models of the parts. Around the parts, the render shows the board's own pads, holes and silkscreen, and the copper under the solder mask. `kicad-cli pcb render` leaves out the models of parts marked DNP, so U2 and U3 show only their pads. Made by `scripts/figures/kicad_figures.sh`, as are the other renders on this page.

The chip, U4, sits in the middle on KiCad's own footprint `QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm`, which is for a 9 × 9 mm QFN with 64 pins at 0.5 mm pitch and a 7.65 mm square exposed pad underneath. Its schematic symbol, `Openframe_QFN_TT06_REV2`, comes from Tiny Tapeout's own library. The chip is placed at 45° to the board's edges, which the board file records as a rotation of 135°.

Around it are the footprints of the parts that power the chip, and a few others. U1 and U2 are SOT-23-5 footprints for an AP2112K-1.8 regulator. U3 is a footprint for a 24LC02 EEPROM. J2 is the footprint for a 1 × 12 header along the top edge, and the silkscreen labels its pins GND, ENA, RST, INC, VCCD, VDDIO and VDDA. There are also four test points, a set of resistors (several of them 0 Ω), and capacitors in three sizes, 0402, 0603 and 0805. Those are imperial size codes, and KiCad's footprint names give the metric size as well, so `C_0402_1005Metric` is for a part 1.0 × 0.5 mm. The schematic explains what the parts are for in notes on the sheet. One reads "For Sky: VCC* 1v8 / VDD* 3v3", which gives the voltages of the chip's core supply (`VCCD`, 1.8 V) and of its I/O and analog supplies (`VDD_IO` and `VDDA`, 3.3 V). Another says the "Breakout is fed 3v3 (guaranteed) and 5V (assumes USB supply) and regulates core voltages as needed. Each LDO has jumpers to select source or bypass completely." A third says that "each supply pin is jumpered and may be monitored or fed through these headers by popping off the relevant resistor (0805s for easier manip)".

So the board has footprints for more than one way of powering the chip, and a finished breakout is built for one of them. In Lab 01 you list which parts are fitted, and then follow each supply from the connectors to the chip's pins to find out which way this revision is built.

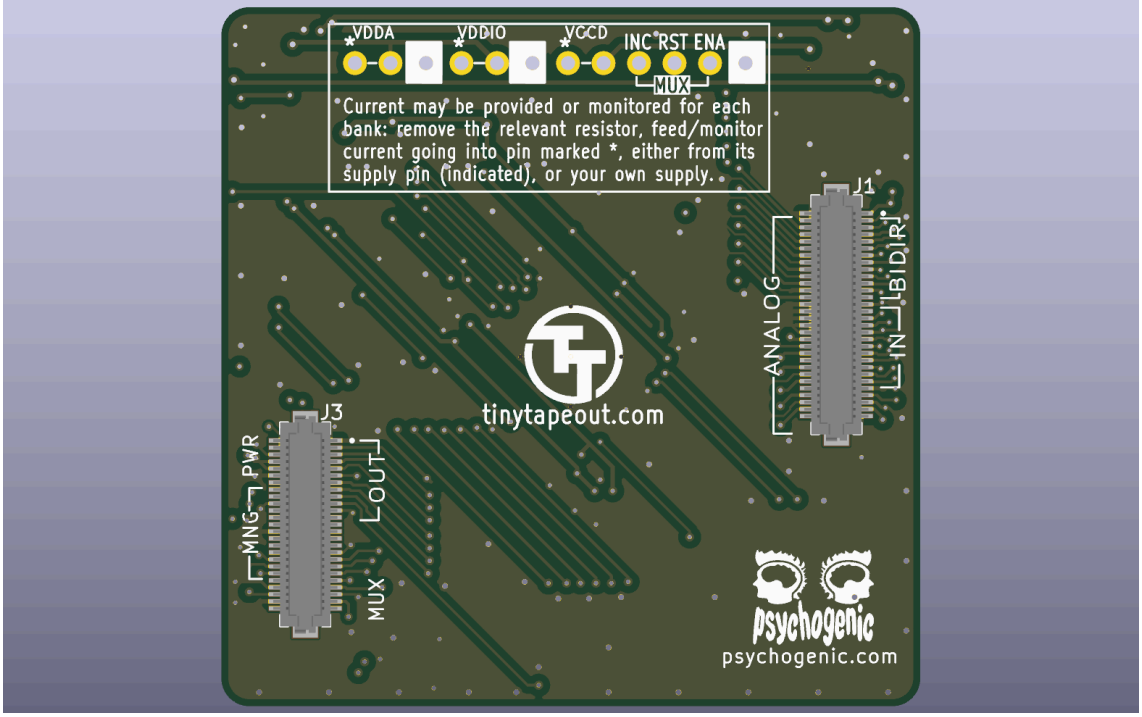


Figure 4.2: A 3D render of the bottom of the same breakout

The bottom of the breakout, with its two Hirose DF12 connectors, J1 with 60 pins and J3 with 50. They plug into the demo board and carry every signal and supply between the two boards.

Unfitted footprints and DNP

A footprint on a board means the copper will be there, but not that a part will be fitted to it. When a designer leaves room for a part that a particular build doesn't use, they mark it **DNP** in KiCad, for "do not populate". In the board file a DNP footprint carries `dnp` in its attributes, and in the schematic its symbol carries `(dnp yes)` and is drawn crossed out. Several of the breakout's footprints are marked that way.

DNP affects assembly and not fabrication. The fabricator still makes the pads of a DNP footprint (you can see U2's and U3's pads in the render above), and the assembler leaves them empty. `kicad-cli`, which the course's checks run, adds a DNP column to a bill of materials by default, and its `--exclude-dnp` option leaves DNP parts out of a bill of materials or a position file. You make both files in Lab 05.

The board file says what the designer intended. For what was built, Tiny Tapeout keeps a separate repository, `pcb-files`, which preserves "the PCB/PCBA production files as they were used when creating the relevant run", including the bill of materials sent to the assembler. In Lab 01 you compare the breakout's DNP marks with that bill of materials for the same revision.

Links and jumpers

The breakout's supplies run through **0 Ω resistors**, which are parts in an ordinary resistor package whose resistance is (very nearly) zero. A 0 Ω resistor behaves as a wire, but the assembly machine places it like any other resistor, and you can take it off with a soldering iron to open the connection, so it works as a jumper you can change after the board is built. Three of the breakout's links are larger 0805 parts, "for easier manip" in the note's words, because those are the ones you'd take off by hand to measure a supply or to feed it from outside.

The demo board puts a second kind of jumper on each of its eight `uo_out` lines, so every project's outputs go through one. The net names in its board file give this path from `uo_out0`:

- `uo_out0` goes to one side of JP2, a two-pad solder jumper whose footprint is bridged with copper (KiCad calls the part `SolderJumper_2_Bridged`).
- The other side of JP2 goes to R19, a 510 Ω resistor in an 0402 package.

- The other side of R19 goes to segment A of the 7-segment display, U4.

`uo_out1` takes the same path through JP3 and R20 to segment B, and so on up to `uo_out7`, which reaches the decimal point through JP9 and R32. A bridged solder jumper is closed as made, and you open it by cutting the thin copper between its pads. So unless someone cuts its jumper, each `uo_out` pin drives a display segment through 510 Ω, and any load your own board puts on that pin comes on top of the segment.

On the input side, the input DIP switch SW1 has eight switches, and each one that's on connects its `ui_in` line to the 3.3 V supply through its own 1 kΩ resistor (R1, and R4 to R10).

The demo board

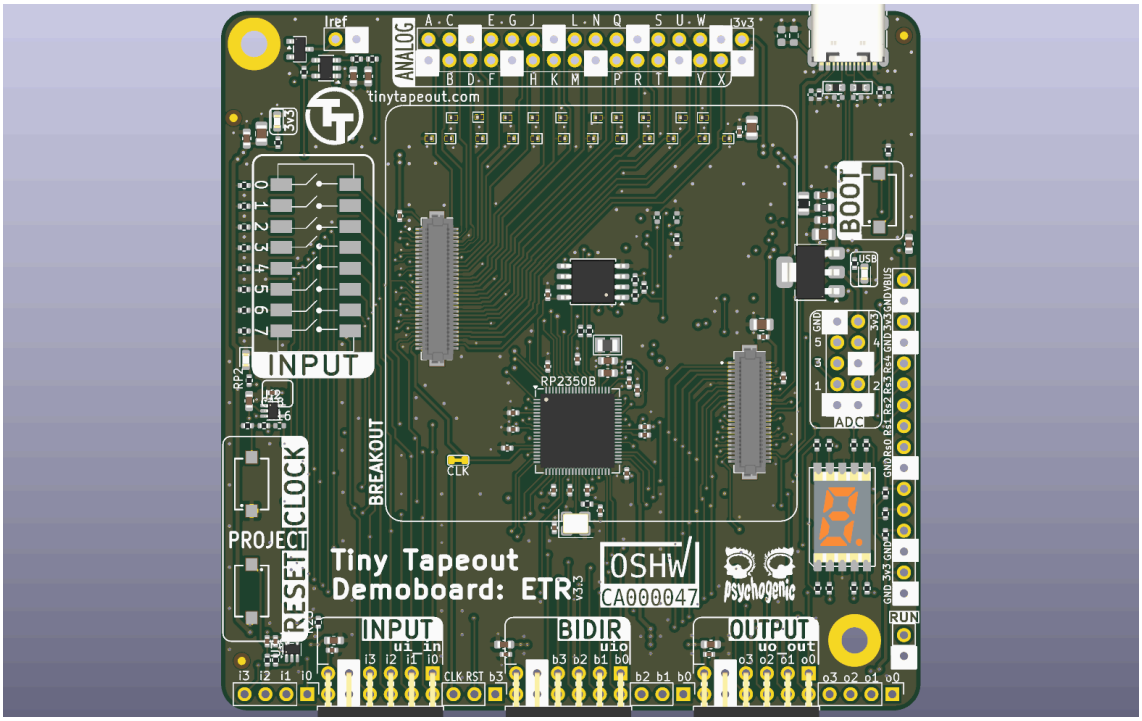


Figure 4.3: A 3D render of the top of Tiny Tapeout's demo board

The demo board, rev 3.3, from `TinyTapeout/tt-demo-pcb` at bbf3dd1. The rectangle marked BREAKOUT is where the breakout plugs in.

The demo board's microcontroller is an RP2350, U6, in a QFN-80 with 0.4 mm pitch, and its flash memory, U7, is an 8-pin SOIC. A TLV1117LV33 regulator, U1, makes 3.3 V from the USB supply, which comes in on the USB-C connector, J1. Besides the input switches and the display, the board has the unfitted analog header along the top, J13, whose lines are labelled A to X, and the three I/O Pmod sockets along the bottom edge, J14, J15 and J16. In the board file the three Pmod sockets have their pin 1 at x = 92.70, 115.56 and 138.42 mm, 22.86 mm (0.9 in) apart, and a board that plugs into all three has to match that spacing exactly.

The demo board also has 24 TVS diodes, which clamp fast voltage spikes such as an electrostatic discharge. Two of them, D1 and D2 (value "TVS"), are on the USB data lines, and the other 22 (value "5VTVS", in SOD-882 packages) are on the analog header's lines `/A` to `/X`. None of them is on a digital project pin, so the project pins that a Pmod board reaches have only the chip's own protection. Supplies, back-powering and ESD covers the ESD protection you add to your own board for them.

The DIP dock

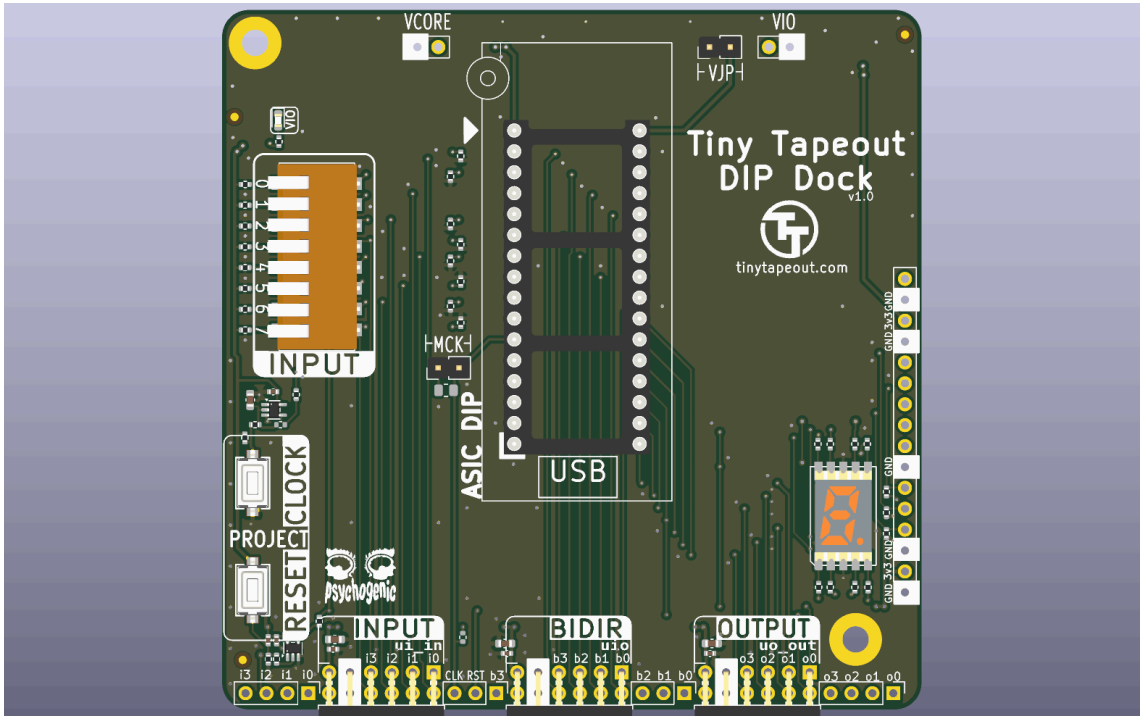


Figure 4.4: A 3D render of the top of Tiny Tapeout's DIP dock

The DIP dock, rev 1.0, from `TinyTapeout/breakout-pcb` at 91a2bf1.

Not every Tiny Tapeout chip comes in a QFN on a breakout. For chips delivered on a DIP carrier, Tiny Tapeout made the DIP dock. Its README calls it a "Docking station for Tiny Tapeout DIP breakouts" and says it "provides the same footprint, and same interfaces" as the demo board, which it lists as the input DIP switch, the 7-segment display, Pmods for I/O, and manual clock and reset. The README also says "The board is very simple, and designed using only two layers."

The board file does have two copper layers, and much of it follows the demo board's file. Its outline is the same 85 × 85 mm, its three Pmod sockets (J5, J6 and J7) sit at the same x = 92.70, 115.56 and 138.42 mm, and its three fiducials (the bare copper dots an assembly machine aligns to) are at the same coordinates. Its custom design rules file, `dip-dock.kicad_dru`, is an exact copy of the demo board's, including rules for a QFN-80 and a USB-C connector that the dock doesn't have. Copying a rules file from an older board is common, and here it does no harm, because a rule that matches no parts checks nothing. Layers and fabricator rules shows a rule on the SKY130 breakout whose condition matches no part on that board either, and explains why you should read the rules you inherit before you rely on them.

Of Tiny Tapeout's boards, the DIP dock is the most like the board you draw in Lab 03, which also has two layers, the same Pmod spacing and simple parts. The dock's Pmod sockets are the host side, the side a Pmod plugs into, and your Lab 03 board is the Pmod that plugs into them.

Two layers and four

Three of these four boards have four copper layers, and the DIP dock has two. A 4-layer board's two extra layers are inside it, between the top and bottom layers. On the SKY130 breakout, a ground zone fills all four layers around the tracks, and the first inner layer is almost solid ground, which gives the signals on the top layer a return path close underneath them. A 2-layer board has to route on both of its layers and fit its ground in around the signals. Layers and fabricator rules shows both stackups from the board files.

Tiny Tapeout's errata

The `breakout-pcb` README describes two mistakes in earlier boards. It says the TT07 breakout "has the ASIC project MUX `reset_n` and `increment` wires crossed and is thus mislabelled", and that the production

run of the TT06 demo boards "had very low valued resistances which, if you wish to interact with analog projects, had to be removed or replaced."

KiCad's checks don't find mistakes of either kind. ERC checks the schematic against rules about pins, and DRC checks the board against geometric rules and against the schematic. Neither reports two wires that connect cleanly to the wrong pins, or a resistor of the wrong value. You find mistakes like these by reading the board against what it's meant to do, which is the review you do in Lab 06.

Check yourself

1. U2 on the breakout carries the `dnp` attribute. Does the bare board you'd get from the fabricator have pads for U2?

   Answer: A1, in Answers at the end.

2. Why would a designer put a 0 Ω resistor in a supply line instead of a plain track?

   Answer: A2, in Answers at the end.

3. You design a board to plug into the demo board's three I/O Pmods. Will it also plug into the DIP dock?

   Answer: A3, in Answers at the end.

4. The TT07 breakout had its `reset_n` and `increment` wires crossed. Would ERC or DRC have reported it?

   Answer: A4, in Answers at the end.

Next: Lab 01: Open a board you didn't design.

Answers

A1. Check yourself

Yes. DNP tells the assembler not to fit the part. The footprint's copper is still part of the board, which is why U2's pads show in the render.

A2. Check yourself

So that the connection can be opened after the board is built, by taking the resistor off. With the link removed you can put a meter in its place to measure the supply's current, or feed that pin from somewhere else, as the breakout's notes describe. A track would have to be cut, which is much harder to undo.

A3. Check yourself

Mechanically, yes: the dock's three Pmod sockets are at the same positions, 22.86 mm apart, and its README says it provides the same interfaces as the demo board, Pmods for I/O among them.

A4. Check yourself

No. The wires connected cleanly, just to each other's pins. ERC and DRC report broken rules, such as unconnected pins, conflicting outputs or copper too close together. Two swapped wires break none of those rules, so it takes a review against what the design is meant to do to catch them.

5. Lab 01: Open a board you didn't design

In this lab you install KiCad and open Tiny Tapeout's SKY130 QFN breakout, which is the board that the team's SKY130 chips arrive on. You work out which of its parts are fitted, and follow each of the chip's three supplies from the connectors to the chip's pins. Then you run ERC and DRC on the breakout and explain each kind of error they report. You write all of that into one file of notes, and a check compares your notes with the board.

This lab takes about 1 h 30 min, not counting the time that KiCad and the course's Docker image take to download.

Package: `labs/lab-01-open-a-board-you-didnt-design/`

Before you start

- Read Getting started, What a board does for a chip and Tiny Tapeout's boards, part by part.
- You need Docker, git and Python 3 from IC101. You also need 2.6 GB of disk for the course's KiCad image once it's unpacked, and 84 MB for the boards you fetch.

Part 1: Install KiCad and get the boards

1. Install KiCad 10.0.5 on your own computer, as Getting started describes for Windows, macOS and Linux. If you can only get a later 10.0 release, or you already have one, use that. Any 10.0 release opens the course's files, and Getting started says what differs between releases. The course's checks run 10.0.5 in Docker whichever release you have installed. Start KiCad and open Help → About KiCad, where you should see a version that starts with `10.0`.
2. Clone the course and go into it:

   ```
   git clone https://github.com/uoftasic/pcb101.git
   cd pcb101
   ```

   Run every command in this lab from that folder.
3. Fetch Tiny Tapeout's boards, two of its KiCad libraries and the breakout's production BOM. The course doesn't keep copies of them, so this script checks each one out of Tiny Tapeout's repositories at a fixed commit:

   ```
   python3 labs/common/fetch_boards.py
   ```

   It took about 8 s for us, and you should see:

   ```
   fetched    breakout-pcb                 91a2bf1    21.0 MB  labs/common/upstream/
   breakout-pcb
   fetched    tt-demo-pcb                  bbf3dd1    12.8 MB  labs/common/upstream/
   tt-demo-pcb
   fetched    kicad-tinytapeout-pmod-lib   9ee6345     0.4 MB  labs/common/upstream/
   kicad-tinytapeout-pmod-lib
   fetched    pcb-files                    ca5fd72     0.3 MB  labs/common/upstream/
   pcb-files
   fetched    tinytapeout-kicad-libs       0e99691    28.5 MB  labs/common/upstream/
   breakout-pcb/tinytapeout-kicad-libs
   ```
4. Check that the course's pinned KiCad runs:

```
labs/common/kicad.sh kicad-cli version
```

The first time, it pulls `kicad/kicad:10.0.5`, an 808 MB download. After that you should see:

```
pcb101:                    KiCad                    image                    kicad/
kicad@sha256:182c8005cb775a2c448a4c18681d489f1ff472a761885eba3e08b07e3c0564de
10.0.5
```

The first line is the digest that Docker resolved, which records exactly which build of KiCad ran. `kicad.sh` is a bash script, so it runs on Linux, macOS and WSL. If you don't have bash, `labs/common/README.md` has a plain `docker run` command that runs Part 4's board check without `kicad.sh`; we have run it on Linux but not on Windows.

Check: KiCad's About box says 10.0, `kicad-cli version` prints `10.0.5`, and `labs/common/upstream/breakout-pcb/breakout-sky-qfn/` exists.

Part 2: Fitted and unfitted parts

Which of the parts drawn on this board does the factory actually solder on?

1. In KiCad, open `labs/common/upstream/breakout-pcb/breakout-sky-qfn/breakout-sky-qfn.kicad_pro`, then open its schematic. Tiny Tapeout saved these files in KiCad 9, so a bar across the top says the file was created by an older version of KiCad and will be converted when saved. Don't save it. If you do, the fetch script will name the file you changed and refuse to update the board, and you'll have to rerun it with `--force` to put the original back.
2. Open `breakout-notes.md` in the lab folder in a text editor. It's the file you'll fill in during this lab, and the check reads its headings and table columns, so keep those as they are.
3. Find the block labelled "Core Voltage Selection" at the top left of the sheet and zoom in on it (the mouse wheel zooms around the cursor):

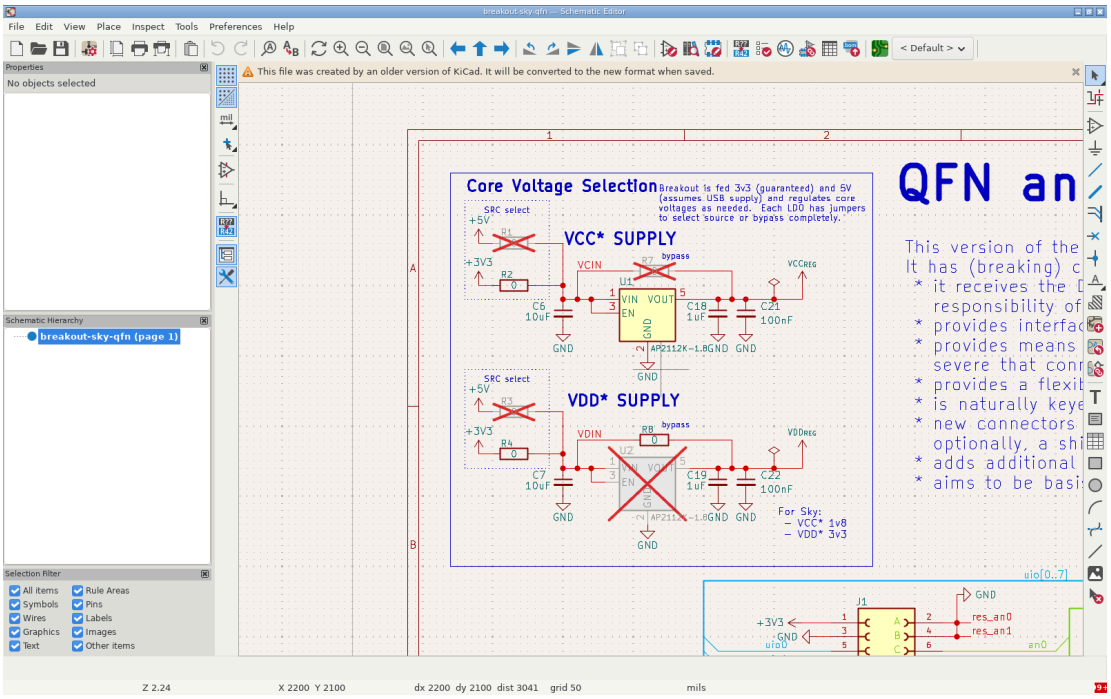


Figure 5.1: The Core Voltage Selection block of the breakout's schematic in KiCad 10.0.5. R1, R7, R3 and the whole of U2 are drawn with a red cross.

KiCad draws a part marked DNP (do not populate) with a red cross. Its footprint is still on the board, so the copper is there, but nothing is soldered to it. In this block, R1, R7, R3 and the second regulator, U2, are marked DNP.

4. Look over the rest of the sheet for crossed-out parts, and list every one under "## Not fitted" in your notes. The "Presence Detection", "EEPROM ID" and "MUX and Supply Headers" blocks have some, and so does the corner beside the title block:

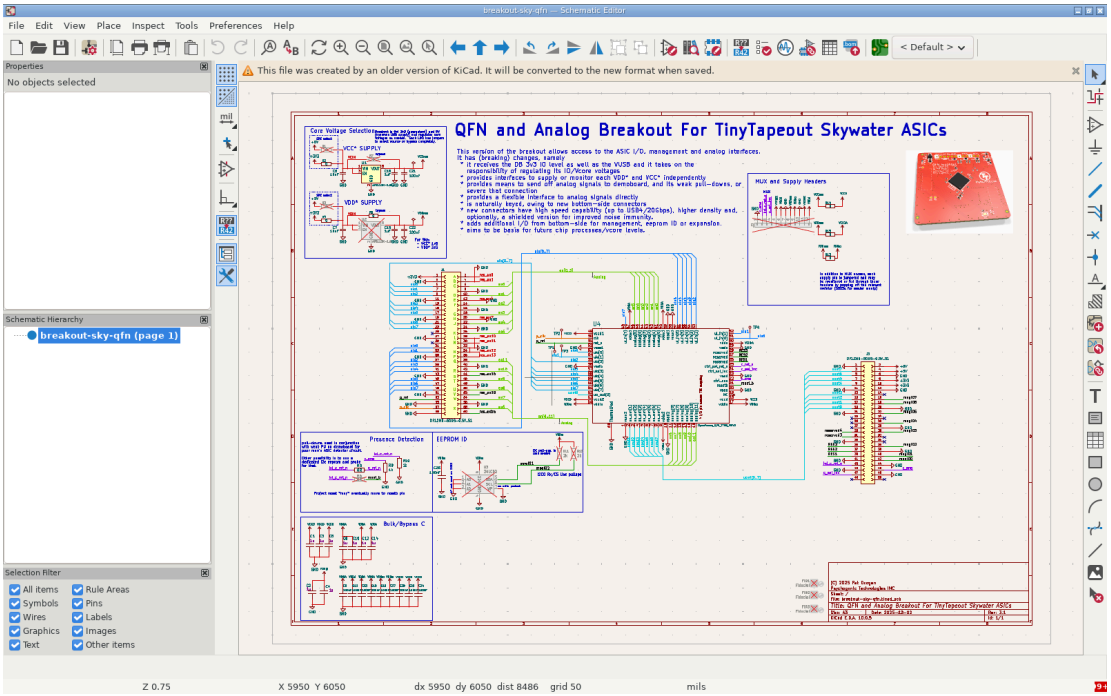


Figure 5.2: The whole of the breakout's schematic in KiCad 10.0.5's Schematic Editor. Core Voltage Selection is at the top left, MUX and Supply Headers at the top right, Presence Detection and EEPROM ID side by side under the left-hand connector, and the title block at the bottom right.

With the whole sheet in view, as in the screenshot, the red crosses are small, so zoom in on each block to read the references of the parts under them. You can confirm any one by opening its properties (double-click it) and looking for the "Do not populate" box.

5. Open the Symbol Fields Table with the grid-shaped button near the right-hand end of the top toolbar:

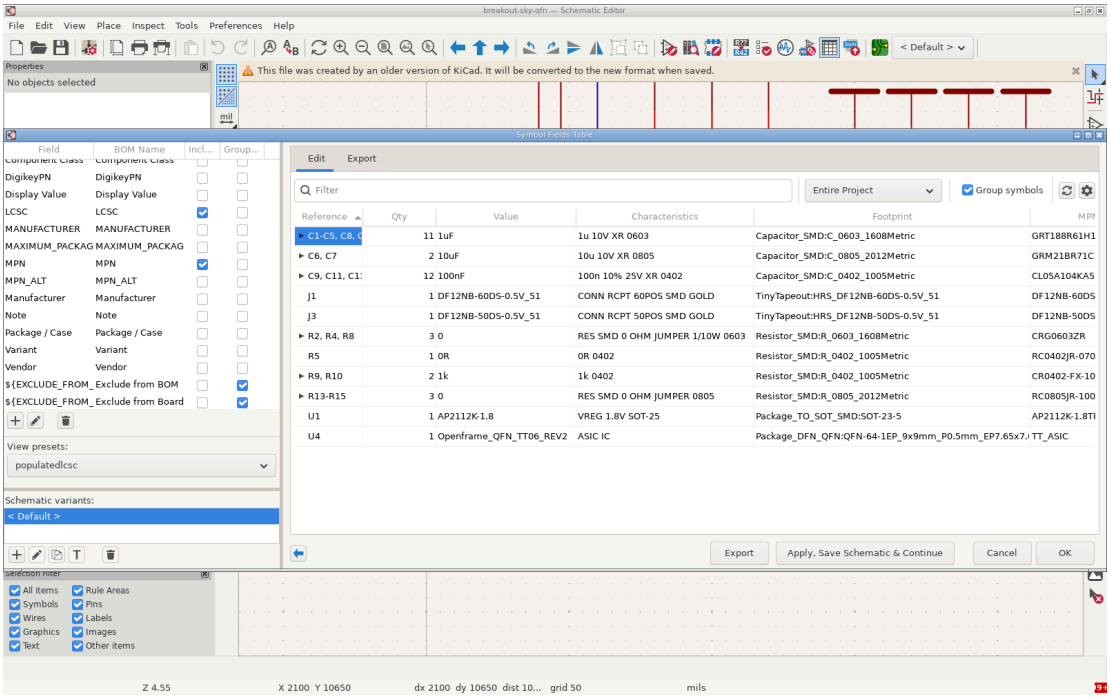


Figure 5.3: The Symbol Fields Table for the breakout, opened on Tiny Tapeout's saved view "populatedlcsc", with one row per group of identical fitted parts.

It opens on a view that Tiny Tapeout saved in the project, called "populatedlcsc". The view leaves out the DNP parts and puts identical parts on one row, the way a factory's parts list does, so C1–C5, C8 and five more 1 µF capacitors share a row, and U1 has a row of its own. None of the parts you listed in step 4 is in it.

6. Compare that table with what the factory was actually given. Tiny Tapeout's production BOM for this revision is `labs/common/upstream/pcb-files/ETR/breakout-skyqfn-v3p2/bom/breakout-skyqfn_bom_jlc.csv`; open it in a text editor or a spreadsheet. You should see 12 lines after the header. Between them they list 25 capacitors, J1 and J3, 9 resistors, and U1 and U4, and there's no U2, U3 or J2.

   The `v3p2` in the BOM's path means rev 3.2. That's the revision in the board file's title block, dated 2025-10-04. The schematic's title block says rev 3.1, dated 2025-12-02. Both files come from the same commit, though. When the board check in Part 4 compares the board with the schematic, the only differences it reports are in the Datasheet field, which is empty in the schematic. Part 4 explains them.

7. FID1, FID2 and FID3 are marked DNP too, and TP1 to TP4 are in neither the BOM nor your list. Why?

   Answer: A1, in Answers at the end.

Check: your "Not fitted" list has 9 parts, plus the fiducials if you included them.

Part 3: The supply rails

How does each of the chip's three supplies get from the connectors to the chip, and at what voltage?

1. Make a prediction first. A note on the schematic says "For Sky: VCC* 1v8, VDD* 3v3", and both regulators, U1 and U2, are AP2112K-1.8s, which put out 1.8 V. The chip, U4, has three supply nets: VCCD, VDD_IO and VDDA. Write down which you expect at 1.8 V and which at 3.3 V, and which regulator you expect to feed each.

2. Find U4's pins on each rail. U4 is the large symbol in the middle of the sheet; zoom in on it and read the numbers beside its supply pins, which are named `vccd`, `vccd1`, `vccd2`, `vddio`, `vdda`, `vdda1` and `vdda2`. (Edit → Find with "Search pin names and numbers" ticked finds them too, but in our run its first match for `vdda` was the value of a VDDA power symbol, so it's slower than reading the symbol.) Write the pin numbers into the "Chip pins" column.

3. Follow VCCD back towards the connectors. Between U1 and the chip, VCCD's supply passes through the "MUX and Supply Headers" block at the top right of the sheet:

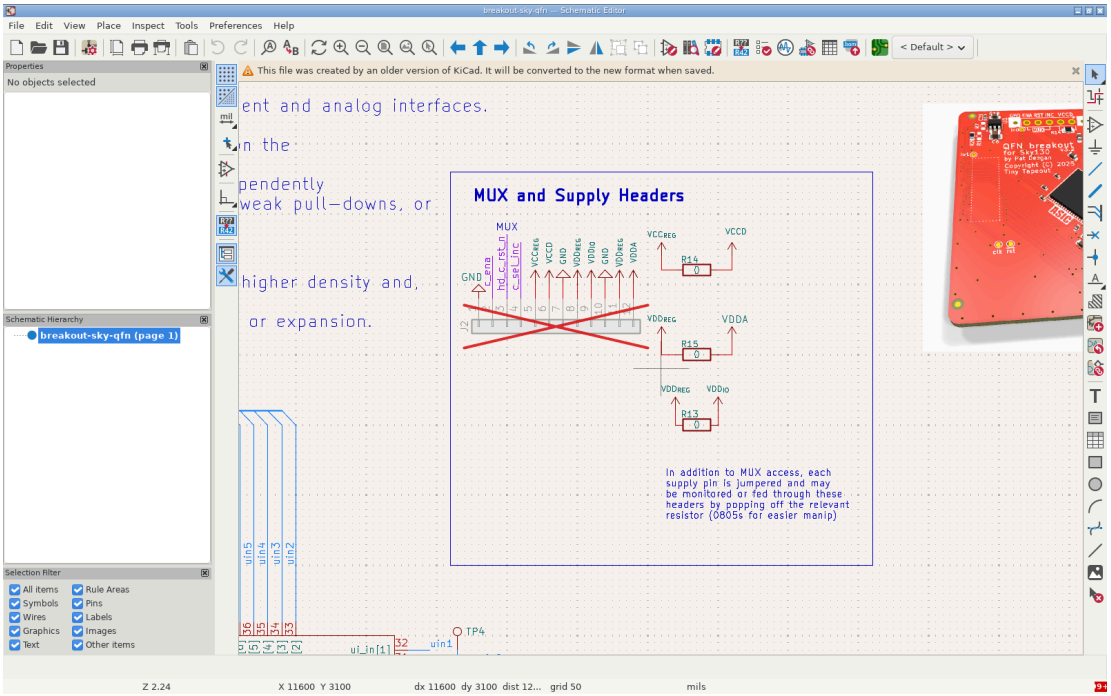


Figure 5.4: The MUX and Supply Headers block: the unfitted 12-pin header J2, and the three 0 Ω links R14 (VCC_REG to VCCD), R15 (VDD_REG to VDDA) and R13 (VDD_REG to VDD_IO).

R14 is a 0 Ω resistor from VCC_REG to VCCD. Then go back to the Core Voltage Selection block. VCC_REG is U1's output, U1's input is the net VCIN, and VCIN comes from +3V3 through R2. Both of the other links on VCIN are DNP: R1, which would bring in +5V, and the "bypass" link R7, which would connect VCIN straight to VCC_REG. So in VCCD's row, "Comes from" is +3V3, and "Through" is R2, U1, R14.

4. Do the same for VDD_IO and VDDA. VDD_IO comes from VDD_REG through R13, and VDDA comes from VDD_REG through R15. VDD_REG is U2's output net, but U2 isn't fitted. What feeds VDD_REG, and what voltage does it carry?

   Answer: A2, in Answers at the end.

5. Find each rail's capacitors. Most of them are in the "Bulk/Bypass C" block at the bottom left of the sheet. Count only the fitted capacitors between the rail itself and GND. C18 and C21 sit on VCC_REG, and C19 and C22 on VDD_REG. Those are the regulators' output nets rather than the chip's rails, so leave them out. Leave out C6 and C7 too, because they're on the regulators' inputs.

6. Fill in the "Supply rails" table: "Chip pins", "Comes from" (the net that the supply arrives on at the connectors), "Through" (the fitted parts, in order from the connector), "Volts" and "Capacitors". Compare the volts with your prediction from step 1.

7. J3 brings +5V onto the board on pins 2 and 4. Where does it go?

   Answer: A3, in Answers at the end.

Check: you have a row for each of VCCD, VDD_IO and VDDA, with every column filled in. Part 4 runs the check on them.

Part 4: ERC and DRC

What do KiCad's checks find on a board that Tiny Tapeout has built and sold, and which of the findings matter?

1. Predict, before you run anything: how many ERC errors and how many DRC errors do you expect on this board? Write the two numbers down.
2. Run ERC in your own KiCad: Inspect → Electrical Rules Checker, or its toolbar button, then Run ERC. Tick "Errors" to list them:

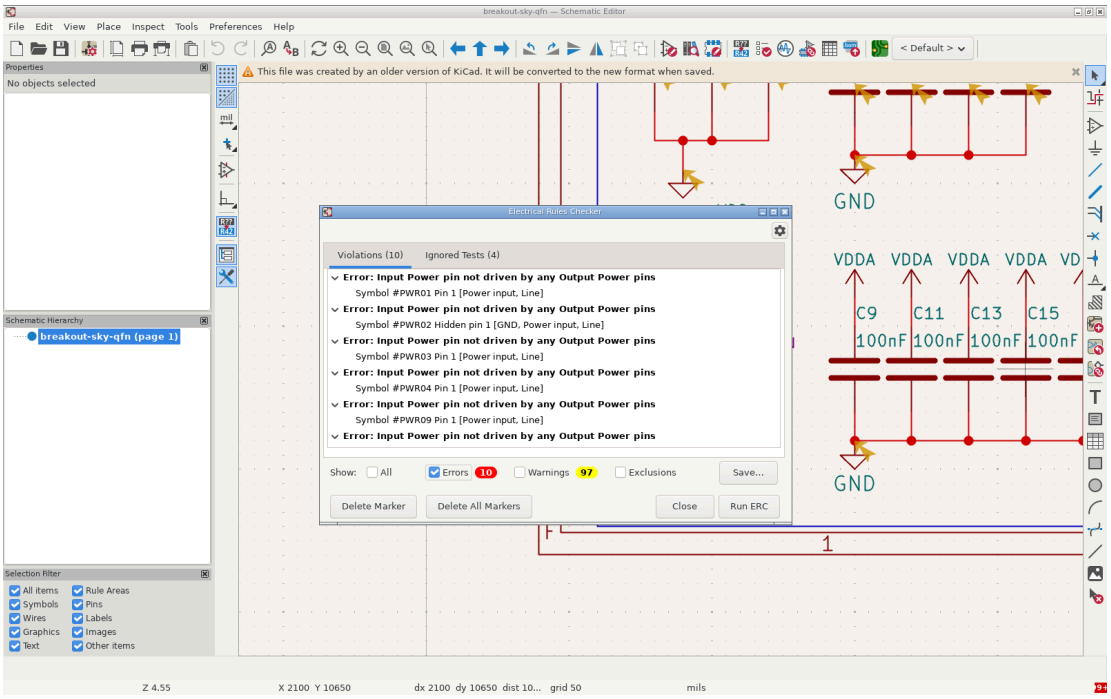


Figure 5.5: KiCad 10.0.5's Electrical Rules Checker on the breakout after Run ERC, with "Errors" ticked: Violations (10), Errors 10, Warnings 97, and the first errors, "Input Power pin not driven by any Output Power pins".

We took this screenshot in the course's KiCad image, with its default library tables, so it shows the same setup that the course's check uses. Your KiCad should also report 10 errors, but it may count a different number of warnings from 97. Many warnings come from comparing the design with whichever libraries your KiCad has set up (Getting started explains this), and later 10.0 releases count some of them differently.

3. Run the course's board check on the breakout. In the pinned image, it runs ERC on the schematic and DRC on the board, compares the board with the schematic (KiCad calls this schematic parity), and counts the results:

```
labs/common/kicad.sh   python3   labs/common/check_board.py   labs/common/upstream/
breakout-pcb/breakout-sky-qfn
```

In our runs it took 23 to 44 s with the container limited to one CPU, and 19 s with two. Below the header lines you should see:

```
                errors  warnings  excluded
  ERC               10        97         0
  DRC                0       120         1
  unconnected        0         0         0
  parity             0        37         0

  ERC errors (10):
    power_pin_not_driven      Input Power pin not driven by any Output Power pins
[Symbol #PWR01 Pin 1 [Power input, Line]]
    power_pin_not_driven      Input Power pin not driven by any Output Power pins
```

```
[Symbol #PWR02 Hidden pin 1 [GND, Power input, Line]]
    power_pin_not_driven     Input Power pin not driven by any Output Power pins
[Symbol #PWR03 Pin 1 [Power input, Line]]
    power_pin_not_driven     Input Power pin not driven by any Output Power pins
[Symbol #PWR04 Pin 1 [Power input, Line]]
    power_pin_not_driven     Input Power pin not driven by any Output Power pins
[Symbol #PWR09 Pin 1 [Power input, Line]]
    power_pin_not_driven     Input Power pin not driven by any Output Power pins
[Symbol #PWR013 Pin 1 [Power input, Line]]
    power_pin_not_driven     Input Power pin not driven by any Output Power pins
[Symbol U1 Pin 1 [VIN, Power input, Line]]
    power_pin_not_driven     Input Power pin not driven by any Output Power pins
[Symbol U2 Pin 1 [VIN, Power input, Line]]
     pin_to_pin                  Pins of type Power output and Power output are
connected  [Symbol #FLG01 Pin 1 [pwr, Power output, Line]; Symbol U1 Pin 5 [VOUT,
Power output, Line]]
     pin_to_pin                  Pins of type Power output and Power output are
connected  [Symbol #FLG02 Pin 1 [pwr, Power output, Line]; Symbol U2 Pin 5 [VOUT,
Power output, Line]]

  parity items (37):
    footprint_symbol_field_mismatch Field 'Datasheet' differs (PCB: '~', Schematic:
'')  [Footprint C1]
    footprint_symbol_field_mismatch Field 'Datasheet' differs (PCB: '~', Schematic:
'')  [Footprint C2]
...
    ... and 17 more (--show 37 lists them all)

     ERC  warnings  by  type:  lib_symbol_mismatch  63,  lib_symbol_issues  22,
isolated_pin_label 11, multiple_net_names 1
  DRC warnings by type: lib_footprint_issues 76, lib_footprint_mismatch 44
  DRC excluded by type: courtyards_overlap 1

FAIL  erc.error: 10 (want 0)
FAIL  parity.warning: 37 (want 0)
```

We've left out 18 of the parity lines, which are the same apart from the footprint they name (C3 to C20). Both FAILs are what should happen here. By default the check allows no ERC errors, and this board has ten. It allows no parity items either, and in KiCad 10.0.5 this board has 37. Every one is a `footprint_symbol_field_mismatch` on the Datasheet field, where the board's copy of the footprint has `~` and the schematic has nothing. KiCad 10.0.8 doesn't report these 37 on the same two files (Getting started lists what else differs between the two releases), so they come from the way 10.0.5 compares the files rather than from a part that is missing or wired differently.

4. Eight of the errors are `power_pin_not_driven`. The first six name `#PWR` symbols, which are the power symbols for VCCD, GND, +5V, +3V3, VDD_IO and VDDA, and the last two name the regulators' VIN pins. Why does ERC think nothing drives them?

   Answer: A4, in Answers at the end.

5. The other two errors are `pin_to_pin`, one between `#FLG01` and U1's VOUT pin, and the other between `#FLG02` and U2's VOUT pin. What has the designer done here?

   Answer: A5, in Answers at the end.

6. DRC reports no errors, but it counts one item as excluded. Run DRC in KiCad's PCB Editor (Inspect → Design Rules Checker, then Run DRC) and look at its exclusions, or add `--reports labs/lab-01-open-a-board-you-didnt-design/results` to the command in step 3 and search the `drc.json` it writes there for `"excluded": true`. Then open the board's design rules in a text editor. They're in `breakout-sky-qfn.kicad_dru`, in the board's folder. What is the excluded item, and why doesn't it count?

   Answer: A6, in Answers at the end.

7. Now look at the kinds of warning, which the check listed by type above. Most of them come from comparing the design with the libraries. `lib_footprint_mismatch` and `lib_symbol_mismatch` mean that the copy of a part saved in the design differs from the one in today's library. `lib_footprint_issues` and `lib_symbol_issues` mean that the design names a library this setup doesn't have, or a part that isn't in its library. 74 of the 76 `lib_footprint_issues` name `flyingcarsfootprints`, which supplies the board's 73 stitching vias and one logo. Will installing Tiny Tapeout's libraries make those 74 go away?

   Answer: A7, in Answers at the end.

8. Fill in the "Checks" table in your notes: the three "all" rows, then one row for each type of error and each type of excluded item, with its count and what it means in your own words. Add warning rows too if you like; the check doesn't read them, because their counts depend on your setup. Compare the error counts with your prediction from step 1.

9. Run the lab's check on your notes:

   ```
   labs/common/kicad.sh    python3    labs/lab-01-open-a-board-you-didnt-design/src/
   check_notes.py
   ```

   It works out its own answers from the board file and runs `check_board.py` for the counts, so it takes about as long as the board check, 35 s with one CPU in our run and 12 s with two. On the notes as shipped, where every row is TODO, its output ends:

   ```
     MISMATCH  Checks        ERC error all: the count is TODO
     MISMATCH                DRC error all: the count is TODO
     MISMATCH                DRC excluded all: the count is TODO
     MISMATCH                ERC error: the check finds 2 types, and your table has 0
   of them
     MISMATCH                DRC excluded: the check finds 1 type, and your table has
   0 of them

   FAIL  5 of 5 rows disagree with the board
   ```

   Each MISMATCH line says what disagrees without giving the answer away. If you list U2 on VDD_IO's path, for example, it says `Through: U2 is marked DNP, so no current flows through it`. Fix the rows it names and run it again until it passes.

Check: `check_notes.py` prints an `ok` row for Not fitted, VCCD, VDD_IO, VDDA and Checks.

Expected results

Check	Expected	
`kicad-cli version` in the image	`10.0.5`	exact
Parts marked DNP	9, plus 3 fiducials	exact
Rails at the chip	VCCD 1.8 V; VDD_IO and VDDA 3.3 V	exact
ERC errors	10: 8 `power_pin_not_driven`, 2 `pin_to_pin`	exact
DRC errors, excluded	0, and 1 excluded `courtyards_overlap`	exact
ERC and DRC warnings	97 and 120 in the pinned image; yours may differ	setup-dependent
Parity items	37 warnings, all `footprint_symbol_field_mismatch` on the Datasheet field, in the	setup-dependent

Check	Expected	
	pinned image; none in KiCad 10.0.8	
`check_notes.py` run time	35 s with one CPU, 12 s with two	ballpark

Finishing

You're done when `check_notes.py` prints `PASS labs/lab-01-open-a-board-you-didnt-design/breakout-notes.md` on your own notes.

Questions:

1. Suppose you wanted to measure VDDA's current alone, with a meter in series. Which part would you take off, and where would you connect the meter?

   Answer: A8, in Answers at the end.

2. Your notes say +3V3 feeds all three rails. What does that mean for a board that the breakout plugs into, if that board needs to change VDD_IO without changing VDDA?

   Answer: A9, in Answers at the end.

Submit: nothing for this lab.

Going further (optional)

- Run `check_board.py` on Tiny Tapeout's demo board, `labs/common/upstream/tt-demo-pcb`, and compare its counts with the table in `labs/common/README.md`. Its parity items include a `missing_footprint`, which KiCad reports as a warning. The course's check fails on parity warnings as well as parity errors so that it catches a missing footprint like this one.
- Open the breakout in the PCB Editor and follow VDDA's copper from R15 to the chip's pins. Which of the board's four copper layers does it use?

Next on the Course path: read Schematics other people can read, Symbols, footprints and libraries and From die to package, then do Lab 02: A footprint from a datasheet.

Answers

A1. Part 2: Fitted and unfitted parts

A fiducial is a bare copper dot that the assembly machine's camera uses to line the board up, so there's nothing to place on it. Tiny Tapeout marked all three fiducials DNP and also excluded them from the BOM, so they stay out of the parts list. The test points here use `TestPoint_Pad_D1.0mm`, a 1 mm pad on the board itself, so they're excluded from the BOM too, but they aren't DNP, because there was never a part to leave off. The check accepts your list with or without the fiducials.

A2. Part 3: The supply rails

R8, the link labelled "bypass", is fitted, and it connects U2′s input net, VDIN, straight to its output net, VDD_REG. VDIN comes from +3V3 through R4 (R3, from +5V, is DNP). So VDD_IO and VDDA both get the demo board's 3.3 V through R4 and R8, and then through R13 for VDD_IO or R15 for VDDA, with no regulator in between. A note on the schematic says that for Sky the VDD rails run at 3.3 V, so U2′s 1.8 V output isn't wanted on them. Another note says that each LDO "has jumpers to select source or bypass completely", and R8 is the jumper that bypasses U2.

A3. Part 3: The supply rails

On this build it goes nowhere. The only parts that +5V reaches are R1 and R3, which would take it to the regulators' inputs, and both are DNP. The schematic's note says the breakout "is fed 3v3 (guaranteed) and 5V (assumes USB supply)", and R1 and R3 are the source-select links that would let either regulator run from 5 V instead.

A4. Part 4: ERC and DRC

ERC treats a net with a power-input pin as driven only if a power-output pin is on the same net. +3V3, +5V and GND arrive through the DF12 connectors, whose pins are passive, and VCCD, VDD_IO, VDDA and the two regulator inputs are each fed through a 0 Ω resistor, which is passive too. ERC can't see the demo board on the far side of the connector, and it doesn't follow a supply through a resistor, so as far as

ERC can tell, none of these nets has a driver. The usual fix is a PWR_FLAG symbol on each such net, which tells ERC that power comes from somewhere it can't see. None of this means the board itself is faulty.

A5. Part 4: ERC and DRC

PWR_FLAG is a symbol with one power-output pin. The designer put the two flags on VCC_REG and VDD_REG. Those are the regulators' output nets, which the regulators' VOUT pins already drive, so each of them ends up with two power outputs. On the nets in the previous question, which have no driver, the flags would have done their job. Like those eight errors, these two come from how the schematic tells ERC where power comes from, and not from a fault on the board.

A6. Part 4: ERC and DRC

It's a `courtyards_overlap` error between J2, the unfitted 12-pin header, and the test point TP4. The last rule in `breakout-sky-qfn.kicad_dru`, called "testpoints under connectors", sets the severity of any courtyard clash between a test point (`TP*`) and a connector (`J*`) to `exclusion`, and when we deleted that rule, DRC reported the same overlap as an error. J2 is DNP, so on this build no header is soldered there to collide with TP4. `check_board.py` counts excluded items separately and never fails on them, and `kicad-cli pcb drc` leaves them out of its report unless you ask for them with `--severity-exclusions` or `--severity-all`.

A7. Part 4: ERC and DRC

No. `flyingcarsfootprints` isn't in the board's own library table, and it isn't in `tinytapeout-kicad-libs`, the library that the fetch script put beside the board, so there's no copy of it for you to install. That's harmless, because KiCad saves a copy of every footprint inside the board file, and the board opens and plots without the library. The other two `lib_footprint_issues` name `Logos`, for the two Tiny Tapeout logos.

A8. Finishing

Take off R15, the 0805 link from VDD_REG to VDDA, and connect the meter between pins 11 and 12 of J2, which carry VDD_REG and VDDA. J2 is the header that the schematic's note means when it says each supply pin "may be monitored or fed through these headers by popping off the relevant resistor (0805s for easier manip)". It isn't fitted on this build, so you'd solder a header into its holes first. Taking off R15 and fitting J2 is rework on the board that carries the kit's only chip, so practise it on a spare board first.

A9. Finishing

As the breakout is built, that board can't change one without the other. VDD_IO and VDDA both come from the connectors' +3V3 through R4 and R8, so changing +3V3 changes both, and the chip's own rails don't reach the connectors at all. To separate them you'd take off R13 or R15 and fit J2, as in question 1.

II

2. Parts and packages

6 **Schematics other people can read . . 39**
6 Nets, labels and power symbols 39
6 Pin types and what ERC checks 39
6 The breakout's ten errors 40
6 Warnings that come from the libraries 41
6 Habits that make a schematic readable 42
6 Check yourself 42
6 Answers 42

7 **Symbols, footprints and libraries . . . 45**
7 Symbols 45
7 Footprints and their layers 47
7 KiCad's library conventions 48
7 Footprints for the same package 49
7 Libraries and library tables 49
7 The copies inside a design 50
7 Check yourself 51
7 Answers 51

8 **From die to package: pads, bond wires, QFN, QFP and BGA 53**
8 The pad ring 53
8 Bond wires 53
8 Packages 54
8 Package parasitics 55
8 Land patterns from a package drawing 55
8 Check yourself 57
8 Answers 58

9 **Lab 02: A footprint from a datasheet . 59**
9 Before you start 59
9 Part 1: Read the drawing 59
9 Part 2: Work out the pads 60
9 Part 3: Draw the footprint 61
9 Part 4: Draw the symbol 63
9 Part 5: Put it on a board 64
9 Expected results 65
9 Finishing 65
9 Going further (optional) 66
9 Answers 66

6. Schematics other people can read

This page answers: *KiCad drew every wire where I put it, so what is ERC complaining about, and what makes a schematic that the next person can check?*

The person who reviews your board reads its schematic, so does whoever brings the board up, and so do you, six months later, with a probe in one hand. KiCad's electrical rules check, ERC, goes through the schematic before any of them, and it can only check what the schematic says about itself. If you leave a pin open on purpose, or a supply comes in from another board, ERC can't know that unless the schematic tells it, and a person reading the sheet has to guess. This page explains how KiCad decides what is connected and what ERC checks. Then it goes through the ten ERC errors on Tiny Tapeout's SKY130 breakout that you found in Lab 01, and the habits that make a schematic readable.

Every ERC count and message on this page comes from `kicad-cli` 10.0.5 in the course's pinned image, run on the breakout at `TinyTapeout/breakout-pcb` 91a2bf1, which is the same setup as in Lab 01.

Nets, labels and power symbols

A **net** is a set of pins that are connected to each other, and on the board each net becomes the copper that joins those pins. KiCad joins pins into nets in four ways:

- **Wires and junctions.** A wire connects wherever one of its ends lands on a pin or on another wire. A **junction**, the dot drawn where two wires cross, joins those two wires.
- **Labels.** A local label joins every wire and pin on the same sheet that carries a label with the same name. A global label does the same across every sheet of the design. A hierarchical label connects a net inside a sheet to the matching pin on that sheet's symbol in the parent sheet.
- **Power symbols.** A power symbol from KiCad's `power` library, such as `+3V3` or `GND`, makes a net that is global across the whole design and is named by the symbol's value.
- **Pins on top of each other.** Two pins at the same point are connected, and you don't need a wire between them.

The breakout's schematic is one sheet with 333 wires, 46 junctions and 128 local labels. It has no global or hierarchical labels, because there's no second sheet for them to reach. In the board file, a net made by a label and a net made by a power symbol are named differently. A local label's net has the sheet's path in front of its name, so the label `VCIN` makes the net `/VCIN`. A power symbol's net has no prefix, so the board's nets for `+3V3`, `GND` and `VCCD` are named just that.

Because a power symbol's value is its net's name, one library symbol can make any number of supply nets. The breakout uses KiCad's `power:+3V3` symbol, with its value changed, for `VCCD`, `VDDA`, `VDD_{IO}`, `VCC_{REG}` and `VDD_{REG}` as well as for `+3V3` itself. (`_{IO}` is KiCad's markup for a subscript, so `VDD_{IO}` is drawn as VDD with a small IO after it.) ERC treats these as six separate nets, and each of them has its own row in the table of errors below.

Two names on one net is a mistake, and KiCad warns about it. On the breakout, ERC gives one `multiple_net_names` warning, "Both GND and res_an7 are attached to the same items; GND will be used in the netlist". It also gives eleven `isolated_pin_label` warnings, "Label connected to only one pin", for labels such as `res_an16` and `res_an19`, each of which is on a net with only one pin. A pin that you mean to leave unconnected should carry a **no-connect flag** (the small cross KiCad places on a pin), which tells ERC that the pin is open on purpose. The breakout has eleven no-connect flags.

Pin types and what ERC checks

Every pin of every symbol has an **electrical type**, which says what the pin does to a net. KiCad's symbol format has twelve, and the standard libraries in the course's image use all of them: input, output, bidirectional, tri-state, passive, free, unspecified, power input, power output, open collector, open emitter and not connected. Resistors and capacitors have passive pins, and so does every connector in KiCad's `Connector_Generic` library. A passive pin carries a signal without driving it.

ERC goes through every net and checks the types of the pins on it against a table of which combinations are fine, which deserve a warning and which are errors. Two outputs on one net are an error, for example, because two drivers would fight over the net's voltage.

ERC also requires a net that has a **power input** pin on it, such as a chip's supply pin or a ground symbol, to have a **power output** pin on it as well, such as a regulator's output. If the net has no power output, ERC reports "Input Power pin not driven by any Output Power pins". The rule catches a supply you forgot to connect, but it also fires whenever a supply reaches a net through passive parts, because ERC doesn't look through a passive pin for a source on the other side. So a supply that comes in on a connector, or through a 0 Ω link, looks to ERC like no supply at all.

To tell ERC that a net like that is driven from somewhere it can't see, you put a **power flag** on the net. It's `PWR_FLAG` in KiCad's `power` library, a symbol with a single power-output pin and no footprint, so it satisfies ERC's rule and puts nothing on the board.

The breakout's ten errors

You found ten ERC errors on the breakout in Lab 01. Here are all ten, with the net each one is on:

Where ERC reports it	Net	Type
`#PWR01` (a `VCCD` power symbol)	`VCCD`	`power_pin_not_driven`
`#PWR02` (a `GND` symbol)	`GND`	`power_pin_not_driven`
`#PWR03` (a `+5V` symbol)	`+5V`	`power_pin_not_driven`
`#PWR04` (a `+3V3` symbol)	`+3V3`	`power_pin_not_driven`
`#PWR09` (a `VDD_{IO}` symbol)	`VDD_{IO}`	`power_pin_not_driven`
`#PWR013` (a `VDDA` symbol)	`VDDA`	`power_pin_not_driven`
U1 pin 1, VIN	`/VCIN`	`power_pin_not_driven`
U2 pin 1, VIN	`/VDIN`	`power_pin_not_driven`
`#FLG01` and U1 pin 5, VOUT	`VCC_{REG}`	`pin_to_pin`: "Pins of type Power output and Power output are connected"
`#FLG02` and U2 pin 5, VOUT	`VDD_{REG}`	`pin_to_pin`, the same message

ERC reports an undriven net only once, at one of its power pins, however many power pins the net has. The `VCCD` net has eleven power symbols in different parts of the sheet, and ERC names only `#PWR01`, so each of the first eight rows is a different net with no source.

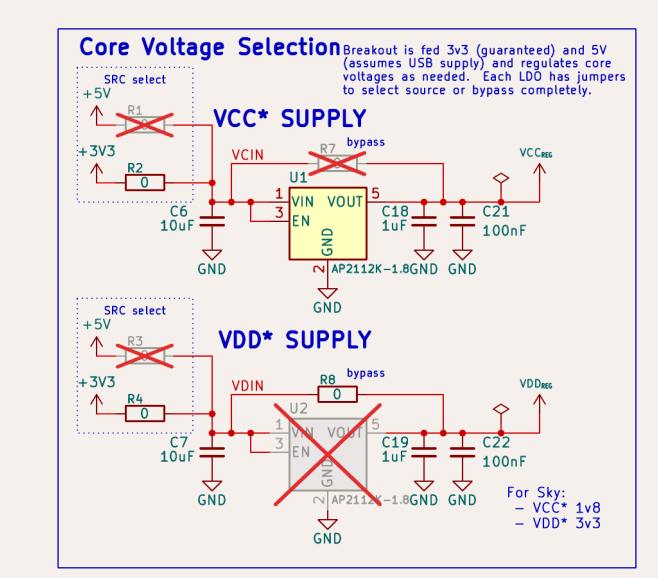


Figure 6.1: The supply section of the SKY130 breakout's schematic

The supply section of the breakout's schematic (rev 3.1, according to its title block), plotted by `kicad-cli` 10.0.5. The crossed-out symbols are the parts marked DNP, and the small diamonds by `VCC_{REG}` and `VDD_{REG}` are the two power flags. Made by `scripts/figures/kicad_figures.sh`.

Six of the ten errors are reported at symbols you can see in the figure. They are the `+5V` and `+3V3` symbols `#PWR03` and `#PWR04` in the upper "SRC select" box, the VIN pins of U1 and U2, and the two power flags. The other four are the `VCCD`, `GND`, `VDD_{IO}` and `VDDA` symbols `#PWR01`, `#PWR02`, `#PWR09` and `#PWR013`, which are in the "Bulk/Bypass C" block at the bottom left of the sheet.

`+3V3`, `+5V` and `GND` reach the breakout on the pins of its two DF12 connectors, and every pin of both connector symbols is passive, so ERC sees no source for any of them. `/VCIN` and `/VDIN`, the regulators' inputs, are fed from `+3V3` through the 0 Ω links R2 and R4, whose pins are passive too, so those nets have power-input pins (the regulators' VIN) and no power output. The chip's three supplies, `VCCD`, `VDD_{IO}` and `VDDA`, have no power output either. Each one is joined to a regulator's output net through another passive 0 Ω link in the "MUX and Supply Headers" block on the right of the sheet. R14 joins `VCC_{REG}` to `VCCD`, and R13 and R15 join `VDD_{REG}` to `VDD_{IO}` and `VDDA`.

The two power flags are on `VCC_{REG}` and `VDD_{REG}`, the regulators' output nets. Each of those nets already has a power output on it, the VOUT pin of its regulator, so the flag adds a second one, and two power outputs on one net is the "Power output and Power output" error.

We checked this by editing a copy of the schematic and running ERC on it again. With the two power flags deleted, ERC reported 8 errors, all `power_pin_not_driven`, on the same eight nets as before, plus 2 new `unconnected_wire_endpoint` warnings for the wires the flags had been on. With a power flag added on each of those eight nets as well, ERC reported no errors. None of the ten errors changes the copper, and the breakout was built and shipped with all ten in its schematic. They're mistakes in what the schematic tells ERC about where power comes from, not in the circuit. On a board of your own, fix them anyway, because if ERC always reports ten errors that you know are harmless, a real eleventh one is easy to miss among them.

So put a power flag on each net whose supply comes in through a connector or a link, and never on a regulator's output net, where the regulator's VOUT pin is already a power output.

Warnings that come from the libraries

Besides the ten errors, ERC reports 97 warnings on the breakout in the course's setup. Apart from the 12 label warnings above, they are 63 `lib_symbol_mismatch` warnings, such as "Symbol 'GND' doesn't match copy in library 'power'", and 22 `lib_symbol_issues` warnings, such as "The current configuration does not include the symbol library 'psypassives'". Both of those kinds compare the copies of symbols saved in the schematic with the libraries KiCad has set up, so their counts change from one KiCad installation to another. Symbols, footprints and libraries explains where those copies come from.

Habits that make a schematic readable

ERC checks connections and pin types, and whether a person can follow your schematic is up to you. These are the habits we ask for in this course, with an example of each from the breakout:

- **Group by function and say what each group is for.** The supply section above sits in a box titled "Core Voltage Selection", with sub-blocks titled "VCC* SUPPLY" and "VDD* SUPPLY" and dotted boxes marked "SRC select", so a reader can find the regulators without searching the sheet for them.
- **Write the intent on the sheet.** The note "Breakout is fed 3v3 (guaranteed) and 5V (assumes USB supply) and regulates core voltages as needed" tells a reviewer what to check the circuit against. Without a note like that, a reviewer can only check the circuit against itself.
- **Name the nets that matter.** A name such as `VCIN` or `VDD_{REG}` tells the reader what a net is for. When nobody names a net, KiCad makes up a name from a part and pad, which is how the demo board ends up with nets like `Net-(R1-Pad2)`. That's fine for a net between a switch and its resistor, which is what that one is, but a net you'd want to probe should have a name.
- **Draw supplies up and ground down.** In the supply section above, every supply symbol points up and every ground symbol points down, so a reader can find the supply rails at a glance. The rest of the breakout's sheet is less consistent. Many of the power symbols beside its connectors and its chip point sideways, and three `GND` symbols on the header J2 point up.
- **Mark parts that aren't fitted.** The crossed-out symbols tell the reader how this revision is built. The board file and the production BOM have to agree with them, and in Lab 01 you checked that the breakout's production BOM leaves them out.
- **Keep the title block current.** The breakout's schematic says rev 3.1, dated 2025-12-02, and its board file says rev 3.2, dated 2025-10-04. Someone comparing a printed schematic with a board in their hand would reasonably wonder whether they had the right files. In a repository of your own, the git history keeps the real record, and the title block should agree with it.

Check yourself

1. Your Pmod board gets 3.3 V from a pin of a Pmod connector, and ERC reports "Input Power pin not driven by any Output Power pins" at your `+3V3` symbol. Why, and what's the fix?

   Answer: A1, in Answers at the end.

2. Why did the two power flags on the breakout cause errors rather than fix them?

   Answer: A2, in Answers at the end.

3. ERC lists one `power_pin_not_driven` error for `VCCD`, at `#PWR01`, but there are many `VCCD` power symbols on the sheet. How many power flags does it take to clear it?

   Answer: A3, in Answers at the end.

4. A wire on your schematic carries two labels, `SDA` and `I2C_SDA`. What does ERC report, and what should you do?

   Answer: A4, in Answers at the end.

Next: Symbols, footprints and libraries.

Answers

A1. Check yourself

Nothing on the `+3V3` net is a power output. The power symbol and your parts' supply pins are power inputs. The connector's pin doesn't count as a source either, because a generic connector's pins are passive, and the power pins of the Tiny Tapeout Pmod symbol that you'll use in Lab 03 are power inputs. The 3.3 V comes from a board on the other side of the connector, which ERC can't see. Put a `PWR_FLAG` on the net, near the connector where the supply comes in.

A2. Check yourself

They're on `VCC_{REG}` and `VDD_{REG}`, the regulators' output nets, which already have a power output pin each (VOUT). A power flag is another power output, and two power outputs on one net is an error in ERC's table, "Pins of type Power output and Power output are connected".

A3. Check yourself

One, anywhere on the `VCCD` net. ERC reports each undriven net once, and a single power output anywhere on a net drives the whole net.

A4. Check yourself

A `multiple_net_names` warning, in the form “Both … and … are attached to the same items; … will be used in the netlist”, which names the label KiCad picked. Decide which name you want and delete the other, so the net has one name everywhere: on the schematic, in the board and in the fabrication files.

7. Symbols, footprints and libraries

This page answers: *the schematic shows the chip as a box with 65 pins, and the board shows it as 64 pads around a square. What ties the two together, and where do they come from?*

Most parts in a KiCad design are drawn twice. A part's **symbol** is what you place in the schematic, and it says what the part does: which pins it has, what they're called and what kind of pin each one is. Its **footprint** is what you place on the board, and it says where the copper, solder and markings go. (A few things are drawn only once. A ground symbol has no footprint, and a stitching via on the board has no symbol.) KiCad joins the two through one field on the symbol, which names the footprint, and through the symbol's pin numbers, each of which has to match a pad number on the footprint. Both drawings come from **libraries**, which KiCad finds through **library tables**. This page covers symbols, then footprints, then libraries, with the chip on Tiny Tapeout's SKY130 breakout as the example throughout, and in Lab 02 you make a symbol and a footprint of your own.

Symbols

A symbol is a drawing with **pins**. Each pin has a number, a name and an electrical type. The electrical types are what ERC checks, as Schematics other people can read showed. A symbol also has **fields**: a reference (U4), a value, a footprint, a datasheet link and a description. A part with several identical sections, such as a quad op-amp, can be split into **units**, each drawn separately.

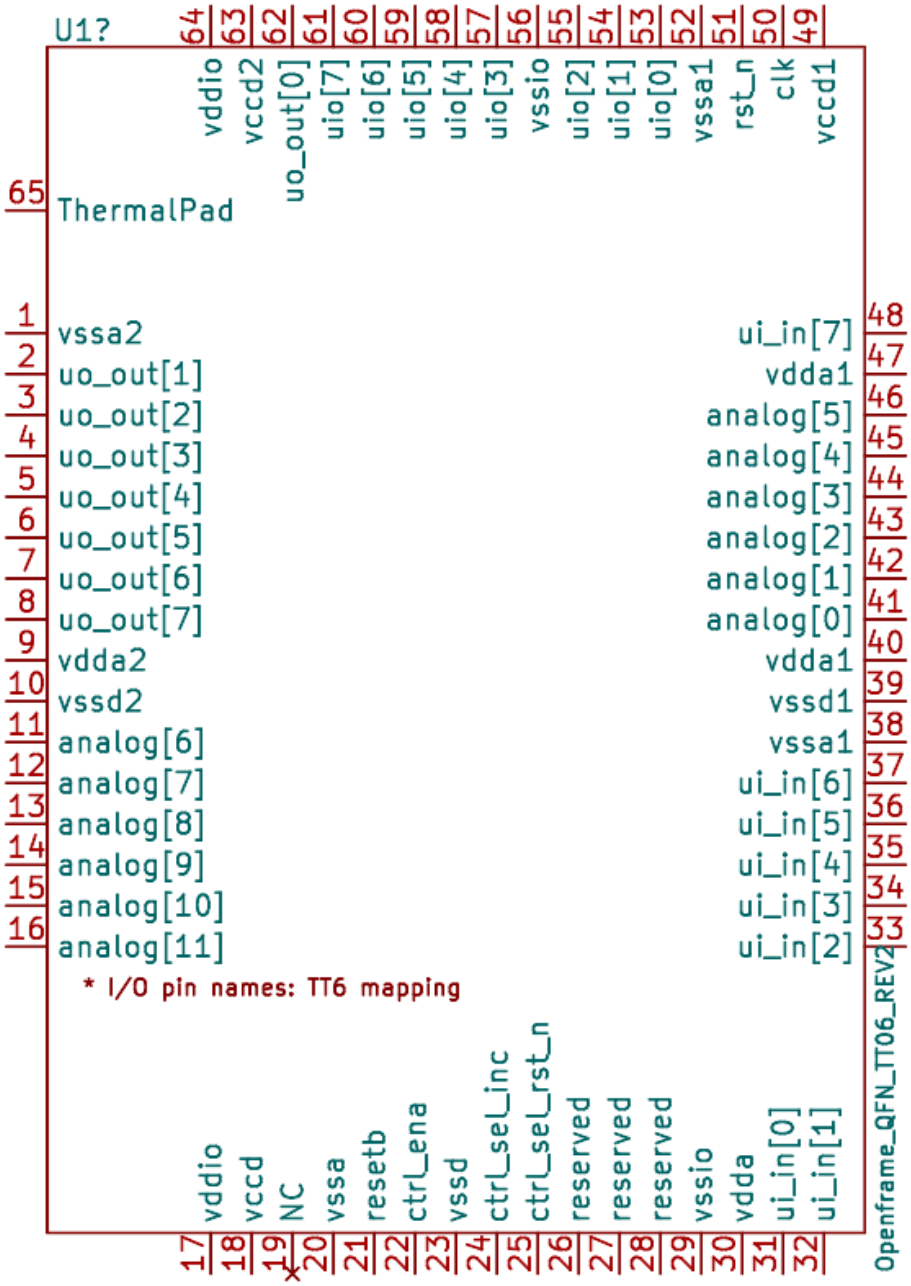


Figure 7.1: Tiny Tapeout's symbol for the SKY130 chip in its QFN-64

The symbol `Openframe_QFN_TT06_REV2` from Tiny Tapeout's library (`tinytapeout-kicad-libs` at 0e99691), plotted by `kicad-cli` 10.0.5. Made by `scripts/figures/kicad_figures.sh`.

The breakout's chip uses the symbol `Openframe_QFN_TT06_REV2`, whose description field reads "Openframe QFN packaging, with TinyTapeout 6+ pin mapping". It's a single unit with 65 pins. Pins 1 to 64 are

the package's pins, and pin 65, named `ThermalPad`, is the exposed pad underneath the package. Of the 65, 19 are power inputs (the supply and ground pins, and the thermal pad), 12 inputs, 11 bidirectional, 10 outputs, 12 passive (the analog pins) and 1 not connected. The symbol's footprint field already names a footprint, `Package_DFN_QFN:QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm`. Every footprint reference has that form: a library's nickname, a colon, and the footprint's name in that library.

ERC works from the pin types in the symbol, so a pin drawn with the wrong type can hide a mistake from it. If a supply pin were drawn as passive, ERC couldn't tell you when the supply was missing, and if two outputs were drawn as bidirectional, it couldn't tell you when they fought. So when you make a symbol from a datasheet, check the pin types as carefully as the pin numbers.

Footprints and their layers

A footprint is a set of numbered **pads** plus drawings on the board's technical layers. The pad numbers join it to the symbol. Symbol pin 17 connects to whatever pad is numbered 17, so the symbol's pin 65, `ThermalPad`, connects to the footprint's pad 65, which is the exposed pad. KiCad checks only that the numbers match, not that a pin and its pad are the same terminal of the package, so a symbol and a footprint made from two different datasheets can disagree without any check reporting it.

These are the layers a footprint draws on, and what each one becomes:

Layer	What it holds	What it becomes
`F.Cu`	the pads' copper	copper on the board
`F.Mask`	openings in the solder mask, usually over each pad	the parts of the board that aren't covered by mask
`F.Paste`	openings in the solder paste stencil	where paste is printed before parts are placed
`F.SilkS`	outline and pin 1 marker (KiCad calls this layer F.Silkscreen in its menus)	printed ink on the board
`F.Fab`	the part's body outline and its reference	assembly drawings, not the board itself
`F.CrtYd`	the courtyard, the area the part and its pads need	nothing on the board; DRC checks that courtyards don't overlap

A part on the back of the board uses the `B.` layers instead. The copper, mask, paste and silkscreen layers go to the fabricator and the assembler, so a mistake on any of them ends up on the real board. The fab and courtyard layers are for people and for DRC.

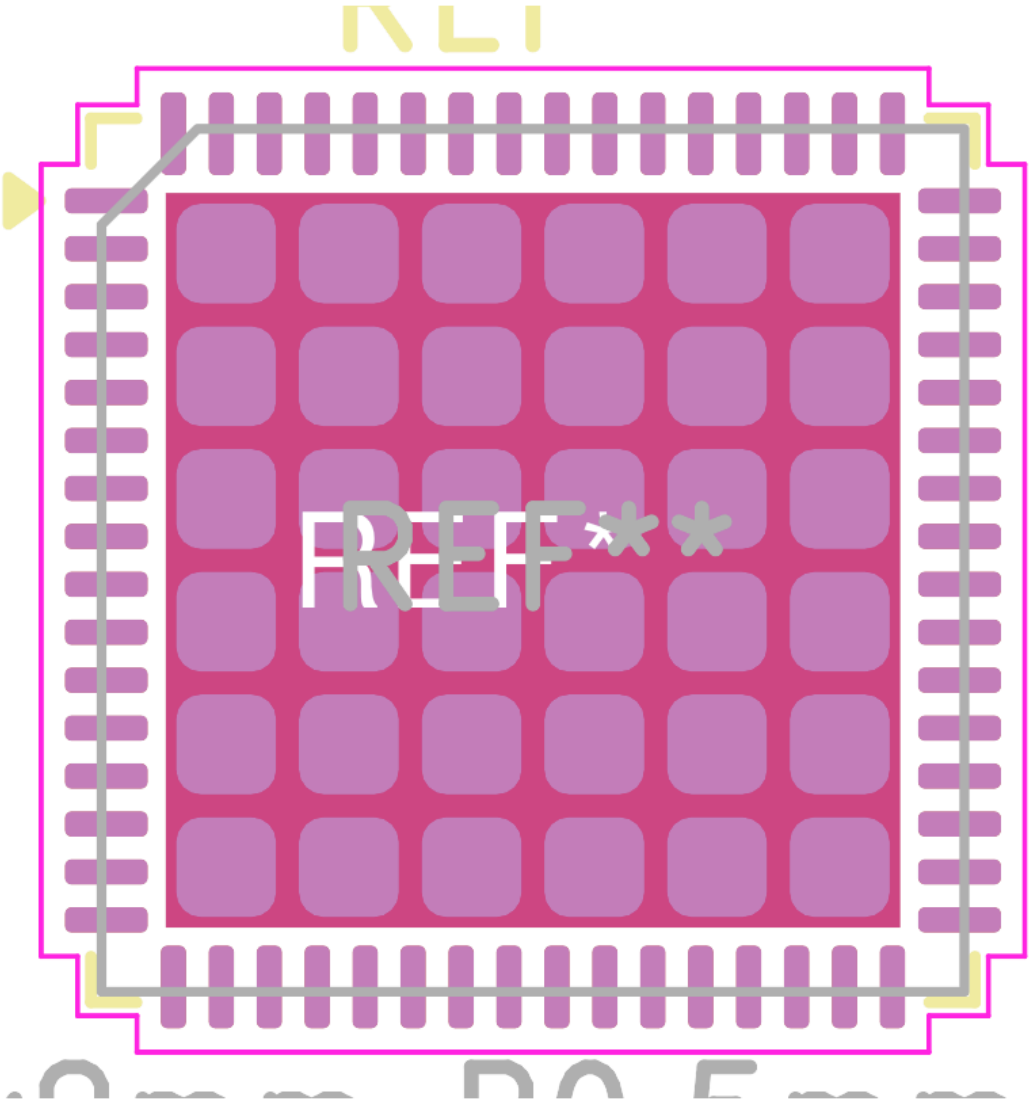


Figure 7.2: KiCad's footprint for a 9 × 9 mm QFN-64 with a 7.65 mm exposed pad

`QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm` from KiCad 10.0.5's `Package_DFN_QFN` library, with its copper, paste, mask, silkscreen, fab and courtyard layers. Made by `scripts/figures/kicad_figures.sh`.

The footprint's 64 signal pads are 0.85 × 0.25 mm rounded rectangles on `F.Cu`, `F.Mask` and `F.Paste`. The exposed pad, pad 65, is a 7.65 mm square on `F.Cu` and `F.Mask` only, and its paste is printed through 36 separate 1.03 mm squares in a 6 × 6 grid, which together cover $36 \times 1.03^2 / 7.65^2 = 65$ % of it. KLC's rule F6.3 says that the exposed pads of large IC packages most often need custom stencil openings like these "to reduce the amount of solderpaste that is applied during stencil application". The silkscreen lines are 0.12 mm wide, the fab outline is drawn in 0.1 mm lines, and the courtyard is drawn in 0.05 mm lines and reaches 5.13 mm from the centre on each side. From die to package shows where the exposed pad comes from.

KiCad's library conventions

KiCad's own libraries follow a set of rules called the **KiCad Library Convention** (KLC), and the numbers above come from it. Its rules make a good default for footprints of your own. These are the ones you'll use most, as they read in the KLC's source at commit 6e8cbad:

Rule	What it says
F4.2	Pin 1 at the top left (“IPC-7351”); for a QFN with pin 1 in the middle of a side, that side at the top.
F5.1	Silkscreen lines between 0.10 and 0.15 mm, nominally 0.12 mm; no silkscreen over pads, with a recommended 0.2 mm clearance from exposed copper.
F5.2	Fab outline in lines of 0.10 to 0.15 mm (0.10 mm recommended), with a bevel at pin 1 on IC packages.
F5.3	Courtyard in 0.05 mm lines, 0.25 mm clear of the part and its pads by default (0.15 mm for parts smaller than 1.5 mm), 0.5 mm for connectors, crystals and canned capacitors, 1.0 mm for BGAs.
F6.3	Rectangular SMD pads as rounded rectangles; at least 0.2 mm between an exposed pad and the other pads; paste over 50 % to 80 % of an exposed pad’s area, 65 % recommended.
F7.5	Annular ring of a through-hole pad at least 0.15 mm (“IPC-2221”).
F7.6	Finished hole at least the lead’s maximum diameter plus 0.2 mm (“derived from IPC-2222 class 2”).

KLC is written for a library that has to work for everyone, not for any one fabricator, and a fabricator’s own minimums can be stricter than KLC’s numbers. Of the three fabricators that Layers and fabricator rules compares, JLCPCB asks for silkscreen lines at least 0.15 mm wide and PCBWay for silkscreen characters at least 0.15 mm wide, and KLC’s footprints draw their silkscreen at 0.12 mm. That page also shows what happens when you check a board against a fabricator’s numbers.

Footprints for the same package

The KiCad image that the course’s checks run in has 155 footprint libraries, and `Package_DFN_QFN` alone holds 776 footprints. Thirty of them have names that start `QFN-64-1EP_9x9mm_P0.5mm`. They share a 9 × 9 mm body and a 0.5 mm pitch, and they differ in the exposed pad, which comes in 16 sizes from 3.4 mm to 7.65 mm square, and 14 of those sizes come in two versions, one with thermal vias and one without. The name spells out what you need to match: the package, the pin count, `1EP` for one exposed pad, the body size, the pitch and the exposed pad’s size.

So it isn’t enough to pick a footprint by its body size and pin count. If you put a QFN-64 package with a 4.7 mm exposed pad on the 7.65 mm footprint, the solder would spread far past the edge of the package’s pad, and a package with a 7.65 mm pad would overhang the pad of a 4.7 mm footprint. Tiny Tapeout’s IHP breakout, for example, uses a footprint from Tiny Tapeout’s own library, `QFN-64-1EP_9x9mm_P0.5mm_EP4.7x4.7mm_IHP_C`, with 4.7 mm in its name where the SKY130 breakout’s footprint has 7.65 mm. The package drawing in the part’s datasheet tells you which footprint is right, and in Lab 02 you work from one.

Libraries and library tables

A footprint library is a folder whose name ends in `.pretty`, with one `.kicad_mod` file per footprint. A symbol library is a single `.kicad_sym` file. KiCad finds them through **library tables**, which map a nickname such as `Package_DFN_QFN` to a path. The global tables (`fp-lib-table` and `sym-lib-table` in your KiCad settings) list the libraries every project can use, and KiCad sets them up with its own libraries the first time it runs. A project can also have tables of its own, files with the same names beside its `.kicad_pro`, which add libraries for that project only.

The breakout has both project tables, and each one adds a library from Tiny Tapeout's library repository under the nickname `TinyTapeout`. This is the line in its `fp-lib-table`:

```
(lib (name "TinyTapeout")(type "KiCad")(uri "${KIPRJMOD}/../tinytapeout-kicad-libs/
footprints/TinyTapeout.pretty")(options "")(descr ""))
```

`${KIPRJMOD}` is the folder the project is in, so this path means "the folder `tinytapeout-kicad-libs` beside this project's folder". In Tiny Tapeout's repository that folder is a git submodule, and the course's fetch script puts the same commit of the library there, so the breakout finds its library when you open it. Because the path starts from `${KIPRJMOD}`, it doesn't depend on where a computer keeps its libraries, and the breakout finds Tiny Tapeout's library on any computer where that folder sits beside it.

The copies inside a design

Once a part is placed, a KiCad design keeps its own copy of the part's drawings and no longer depends on the library. The schematic stores each symbol it uses in a section of its own, and the board stores every footprint in full. That's why the breakout opens even without Tiny Tapeout's library. It's also why the Footprint Editor, opened on U4 from the board, says that it's editing the board's copy.

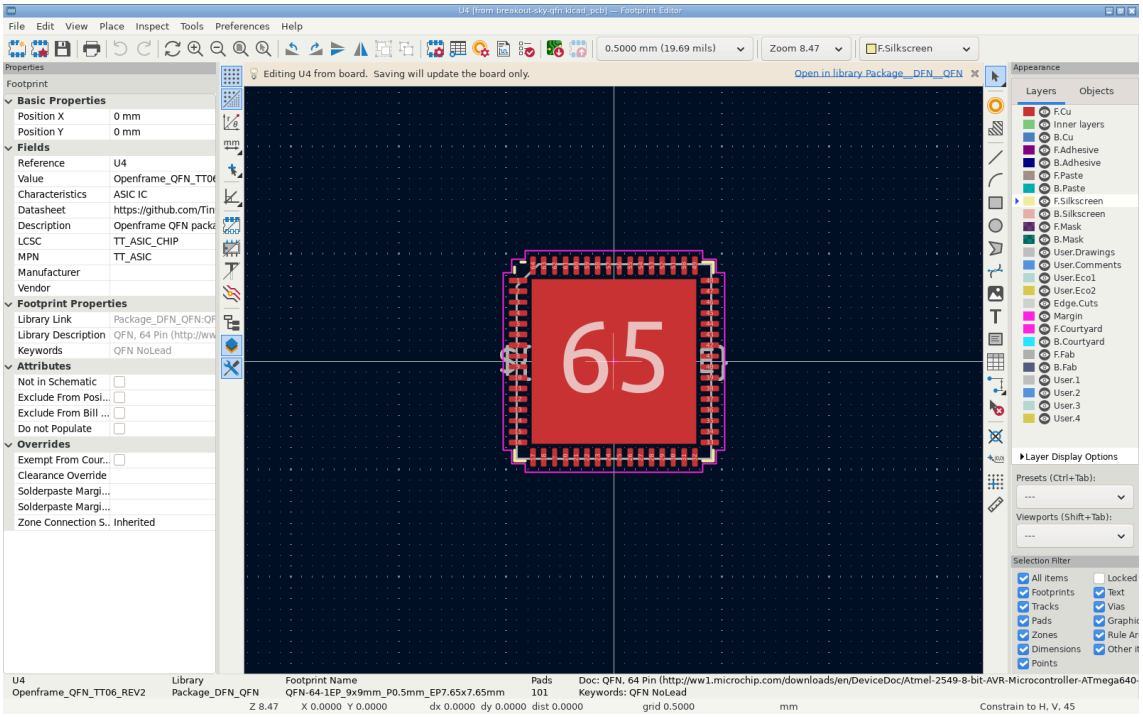


Figure 7.3: The Footprint Editor with the breakout's U4 open from the board

U4 opened from the board in the Footprint Editor of KiCad 10.0.5. The status bar counts 101 pads: 64 signal pads, the exposed pad and its 36 paste openings, which KiCad stores as pads with no copper. Screenshot from `scripts/screenshots/kicad/shot.sh`.

Keeping copies leads to two kinds of warning. When KiCad can find the library a part came from, it compares the copy with the library's version, and when they differ it warns. When it can't find the library at all, it warns about that instead. In the course's setup, DRC on the breakout reports:

- 44 `lib_footprint_mismatch` warnings, such as "Footprint 'C_0402_1005Metric' does not match copy in library 'Capacitor_SMD'". They cover 25 capacitors, 15 resistors, the two regulators, the header J2 and the chip, whose footprints the board took from KiCad's own libraries. The three fiducials, the four test points and the EEPROM U3 also came from KiCad's libraries, but their copies match the library's, so they get no warning. The warning says that the copies differ, not how. We compared the two versions of U4, and their pads are identical. At the pin 1 corner, the library's version draws the silkscreen corner mark all the way into the corner and puts a filled triangle beside it, while the board's copy stops the mark short of the corner and has no triangle. The fields differ as well.

- 76 `lib_footprint_issues` warnings, all "The current configuration does not include the footprint library …". 74 are for `flyingcarsfootprints`, the library that the board's 73 stitching vias and one of its logos came from, and 2 are for `Logos`. `flyingcarsfootprints` isn't in either of Tiny Tapeout's library repositories, so installing Tiny Tapeout's libraries doesn't clear these 74.

ERC reports the same two kinds of warning for symbols, 63 `lib_symbol_mismatch` and 22 `lib_symbol_issues`. All 63 mismatches are power symbols, such as `GND`, from KiCad's `power` library. Of the 22 issues, 20 are for two libraries that the course's setup doesn't have, `psycomponents` and `psypassives`, and 2 are for the connectors J1 and J3, because the symbols that the schematic names for them aren't in Tiny Tapeout's library. These warnings are about whether the design still matches its libraries, and none of them is about the board's copper. When a design is a record of a board that was built, its own copies are what was made, so leave them as they are. When you start a board of your own, though, it's worth resolving them: either update the part from the library, or know why you've kept the old copy.

Check yourself

1. In a symbol you've made, pin 17 is `vddio`. What connects it to a particular pad on the board?

   Answer: A1, in Answers at the end.

2. A friend opens the breakout without Tiny Tapeout's library beside it. Does the chip's footprint still appear on the board, and what changes?

   Answer: A2, in Answers at the end.

3. Your package's drawing gives a 9 × 9 mm body, 64 pins at 0.5 mm pitch and a 4.7 mm exposed pad. Why is `QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm` the wrong footprint?

   Answer: A3, in Answers at the end.

4. The paste openings on that footprint's exposed pad cover 65 % of it. What range does KLC allow?

   Answer: A4, in Answers at the end.

Next: From die to package.

Answers

A1. Check yourself

The symbol's footprint field picks the footprint, and then pin 17 connects to the pad numbered 17 in that footprint, whatever that pad is. KiCad doesn't check that pad 17 is really the package's `vddio` lead, so the symbol's pin numbers and the footprint's pad numbers have to come from the same package drawing.

A2. Check yourself

It appears, because the board stores a full copy of every footprint. The warnings change. KiCad can no longer find the `TinyTapeout` library, so ERC reports it missing for the chip's symbol and for the two connectors, J1 and J3, and DRC reports it missing for the connectors' footprints. The chip's own footprint came from KiCad's `Package_DFN_QFN` library, so its DRC warning stays as it was.

A3. Check yourself

Its exposed pad is 7.65 mm square, far larger than the package's 4.7 mm pad, so its copper and paste would put solder across an area that the package doesn't cover. Every part of the footprint's name has to match the drawing, the exposed pad's size included.

A4. Check yourself

50 % to 80 % of the exposed pad's area, with 65 % recommended (rule F6.3).

8. From die to package: pads, bond wires, QFN, QFP and BGA

This page answers: *a chip is a small square of silicon whose connections are a fraction of a millimetre apart. How does it become something you can solder to a board, and how do you draw the copper it lands on?*

A fab makes a chip as a **die**, a thin piece of silicon with the circuit on its top surface. The die's connections are its **bond pads**, which sit in a ring around its edge. You can't solder a die to a circuit board the way you solder other parts, because its pads are far too small and too close together, and the silicon is fragile. A **package** holds and protects the die, and carries each bond pad out to a lead or a solder ball that is big enough to solder to a board. This page follows one connection from the die out to the board. The copper on the board that a package is soldered to is its **land pattern**, and the second half of the page shows how to work it out from the dimensions on the package's drawing. You'll do that yourself for a different package in Lab 02.

The pad ring

The pads around a die's edge make up its **pad ring** (or padframe). The chip's designers choose which signal or supply each pad carries, and the board has to be wired to match. wafer.space runs shuttles on GF180 and offers chip-on-board packaging as an option. It publishes its padframe in its repository `chip-on-board-wire-bonded-pcbs` (Apache 2.0, commit 5cb4668). The padframe "follow[s] the Tiny Tapeout convention of 74 pads", and a table in `run-1/README.md` maps the bond pads on the die to pads on wafer.space's breakout board and to their Tiny Tapeout functions. Here are some of its rows:

Bond pad (die)	Breakout pad (board)	Tiny Tapeout function
0	1	`ctrl_ena`
8	9	GND IO
9–16	10–17	`uo[0–7]`
17	18	VDD IO
19–24	20–25	`analog[0–5]`
73	74	VDD IO

The board is drawn from a table like this one, so a mistake in a row becomes a wrong connection on the board. The die's pads are numbered from 0 and the board's from 1, so each breakout pad's number is one more than its bond pad's. When you review a board, check which of the two columns each connection was taken from.

Bond wires

Each bond pad is connected to the package, or directly to a board, by a **bond wire**, which is a fine wire that a wire-bonding machine welds in place at both ends. Bond wires have to follow layout rules, and wafer.space lists the rules for its breakouts in `run-1/wirebonding/README.md`:

- "No wires may cross."
- "Wire angle must be not greater than 45°."
- "Maximum wire length must be less than 3 mm — though a wire that long would already add a significant amount of inductance, so keep them as short as practical."
- "Minimum wire length is 1 mm from the die, or 2× the die thickness (whichever is greater)."
- "No solder mask is allowed in the area to be bonded."

The same file explains how wafer.space matches the pads on the die to the pads on the board. It says that "Wirebonding is still a very manual process — there are no real exchange formats yet that tie the die layout to the breakout PCB", so wafer.space wrote a script that reads the die's layout (its GDS file) and the breakout's KiCad file "and walks pin-for-pin around the padframe".

Services that package dies publish rules of the same kind for their packages. Europractice's ASIC Package Design Rules give limits on bond pad size and pitch, wire length and angle, crossing wires and die thickness for each package it offers. Every page of that document is marked "strictly confidential", even though it's posted publicly, so we link to it rather than repeat its numbers. One of its guidelines is useful for any wire-bonded package, though. You draw the die together with the package's bond pins, split the die's edge into equal segments, and keep each bond pad in the same segment as the bond pin it's wired to. Each pad then faces the pin it goes to, so its wire runs roughly straight out, which keeps the wires inside the angle limit and stops them crossing.

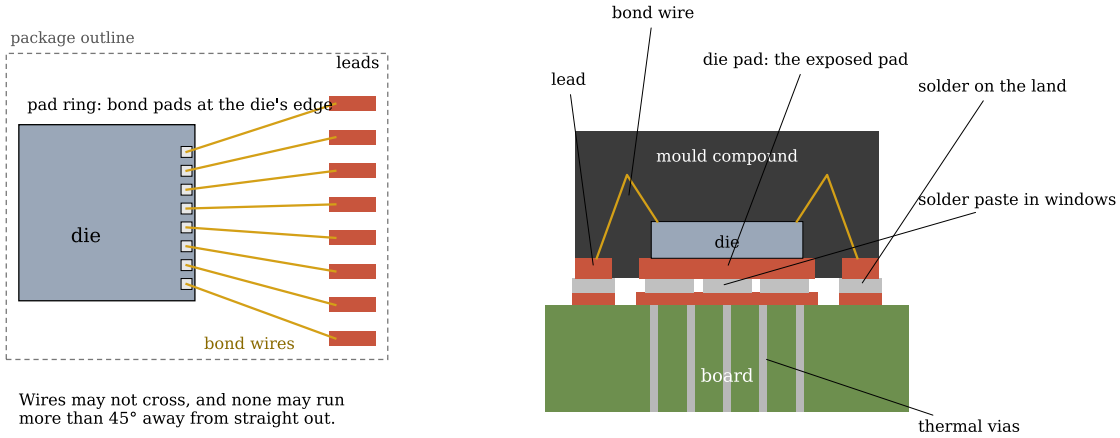


Figure 8.1: A die's pad ring wired out to a package's leads, and a cut through a QFN soldered to a board

Not to scale. Left: a die's pad ring wired out to the leads of its package. Right: a QFN soldered to a board. Drawn by `scripts/figures/qfn_package.py`.

Packages

Packages differ in how their connections leave the body. Of the four kinds below, only the QFN is on the boards in this course.

- A **QFN** (quad flat no-lead) has no legs. Its leads are flat pads around the edge of the body, flush with its bottom face, and the die sits on a metal **die pad**, whose underside shows in the middle of the package's bottom as the **exposed pad**. Europractice's notes on its open-cavity plastic packages describe the same construction, with a copper die attach pad and a lead frame, and its rules say that by default the die is attached to the frame with a conductive die attach. The exposed pad is soldered to the board, which gives the die's heat a path into the board. On the SKY130 breakout the exposed pad is a ground connection as well. It's pin 65 of the chip's symbol, `ThermalPad`, and the board puts that pin on the `GND` net. Tiny Tapeout's SKY130 and IHP breakouts both carry the chip in a QFN-64. The SKY130 breakout's footprint has a 7.65 mm exposed pad, and the IHP breakout's has a 4.7 mm one.
- A **QFP** (quad flat package) has gull-wing leads, which bend out and down from the sides of the body. Because the leads stick out past the body, you can see every solder joint and reach it with a soldering iron.
- A **BGA** (ball grid array) has its connections in a grid of solder balls across the whole of its underside, so even a small package can have a lot of connections. For example, KiCad's footprint generator describes `BGA-624_21x21mm_Layout25x25_P0.8mm` (from NXP's SOT1529-1 drawing) as having a 0.8 mm pitch, 0.5 mm balls and 0.4 mm pads. To get signals out from the inner balls you need vias between the pads or in them. JLCPCB's capabilities page says that "Vias can be placed within BGA pads using filled and plated-over vias", and that BGA pads of 0.2 to 0.25 mm require an ENIG finish. Laying out a BGA is beyond this course.

- **Chip-on-board** (COB) does without a package. The die is mounted on a small board and its bond wires land on pads on that board, which is why wafer.space forbids solder mask in the area to be bonded. Tiny Tapeout's FAQ says its GF180 shuttles work this way, with "the die bonded to a PCB directly (called chip-on-board, or COB)", so the chip "cannot be removed". Tiny Tapeout's TT08 chips were bonded this way too. The news post about them says "The bare dies are now mounted and wire-bonded", and that the first tests of the chip, which carries 135 designs, show working silicon. wafer.space's COB module is 14 × 16 mm and plugs into a motherboard through a 70-pin connector with 0.4 mm pitch (LCSC C19089236, mating part C19089262).

If the team ever has a die of its own to package, it could use the university's own wire bonder or send the die to CMC Microsystems. The University of Toronto's Nanofabrication Centre lists a "TPT HB16 Wire Bonder" in its January 2026 pricing chart at $49 an hour for UofT users. Staff service there costs $60 an hour from 9 am to 5 pm, and equipment training $60 an hour. CMC Microsystems packages dies in DIP 40, CFP 24, CQFP 44, 80 and 120, and CPGA packages from 69 to 209 pins, and says its services "are also accessible to non-subscribers", by quotation.

Package parasitics

Every signal to or from the die passes through the package, so the package's own electrical properties become part of the signal's path. Hall and Heck's *Advanced Signal Integrity for High-Speed Digital Designs* notes that bond wires, vias, lead frames, sockets and connectors are inductive and that CMOS inputs are capacitive (section 3.5.7). It also says that a package often adds 0.25 to 1.0 inch of transmission line to the path (section 3.5.5). wafer.space's warning that a 3 mm bond wire "would already add a significant amount of inductance" makes the same point. For an analog pin on a SKY130 Tiny Tapeout chip, the analog spec says that the whole path from the pad to your project, which includes an analog switch, has under 500 Ω of resistance and under 5 pF of capacitance. At the speeds in this course you'll mostly use these numbers in calculations, such as working out how long an analog input takes to settle.

Land patterns from a package drawing

A package's datasheet has a drawing of it, and most of the dimensions on that drawing are given as a minimum and a maximum, because no two packages come out exactly alike. (A few, such as the pitch, are basic dimensions, which are exact by definition.) The board's copper has to work for every package within those limits, and it also has to allow for a board maker and an assembly machine that are each a little off themselves. KiCad's own footprints follow IPC's method, which is set out in two standards: IPC-7351B, "Generic Requirements for Surface Mount Design and Land Pattern Standard", dated June 2010, and IPC-7352, "Generic Guideline for Land Pattern Design", dated May 2023, which supersedes the June 2010 IPC-7352. Both are sold by IPC, so we teach the method through KiCad's footprint generator, which implements it and publishes its numbers (`kicad-footprint-generator` at commit 090252c).

The method aims for a **solder fillet** of a chosen size at each edge of each lead, which means at its outer end (the **toe**), at its inner end (the **heel**) and along its sides. Each pad is the size of the lead, plus those goals, plus an allowance for tolerances. The generator's configuration copies the goals from IPC-7351B, with the comment "sadly no openly accessible document available", and gives them at three **density levels**, called least, nominal and most, for each kind of lead. For a flat no-lead part such as a QFN, they are:

Goal	Least	Nominal	Most
Toe	0.20 mm	0.30 mm	0.40 mm
Heel	0	0	0
Side	−0.04 mm	−0.04 mm	−0.04 mm
Courtyard	0.10 mm	0.25 mm	0.50 mm

For gull-wing leads at a pitch above 0.625 mm, such as a QFP's or an SOIC's, the toe goals are 0.15, 0.35 and 0.55 mm, the heel goals 0.25, 0.35 and 0.45 mm, and the side goals 0.01, 0.03 and 0.05 mm. (At a pitch of 0.625 mm or less the side goals drop to −0.04, −0.02 and 0.01 mm, because there's less room between the leads.) A negative side goal takes width off the pad, which leaves more room between pads at a fine pitch. In the example below it makes the pads 0.25 mm wide, for leads that can be up to 0.30 mm wide.

The generator then works out three numbers for each pair of opposite rows of pads. In the notation of its own comments, they are:

$$Z_{max} = L_{min} + 2J_T + \sqrt{C_L^2 + F^2 + P^2} \tag{8.1}$$

$$G_{min} = S_{max} - 2J_H - \sqrt{C_S^2 + F^2 + P^2} \tag{8.2}$$

$$X_{max} = W_{min} + 2J_S + \sqrt{C_W^2 + F^2 + P^2} \tag{8.3}$$

Z is the distance between the outer edges of two opposite pads, G the distance between their inner edges, and X a pad's width. L is the span across the leads' outer ends, S the span across their inner ends and W a lead's width, and each C is the tolerance of one of those (its maximum minus its minimum). J_T, J_H and J_S are the toe, heel and side goals. F and P are the tolerances of the board's fabrication and of the part's placement, which the generator sets at 0.1 mm and 0.05 mm. The square roots add the tolerances as a root sum of squares, which assumes that they're independent and rarely all at their worst at once. The generator rounds the results to 0.05 mm.

A worked example: the breakout's QFN-64

Tiny Tapeout's SKY130 breakout uses KiCad's footprint `QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm`, and the generator's entry for it takes its dimensions from page 415 of Atmel's ATmega2560 datasheet. The entry gives a body of 8.9 to 9.1 mm, leads 0.35 to 0.45 mm long and 0.18 to 0.30 mm wide, and an exposed pad of 7.5 to 7.8 mm (7.65 mm nominal). A QFN's leads end at the body's edge, so L is the body size, 8.9 to 9.1 mm, and $C_L = 0.2$ mm. The inner span S is the body less two lead lengths, 8.0 to 8.4 mm. The generator adds the tolerances of the body (0.2 mm) and of the two leads ($0.1 \times \sqrt{2}$ mm) as a root sum of squares, which gives C_S = 0.245 mm rather than the full 0.4 mm range. It then puts S_{max} at the nominal 8.2 mm plus half of C_S, which is 8.32 mm. With the nominal goals, the three numbers are:

$$Z = 8.9 + 2(0.30) + \sqrt{0.2^2 + 0.1^2 + 0.05^2} = 9.73 \rightarrow 9.75 \text{ mm} \tag{8.4}$$

$$G = 8.32 - 0 - \sqrt{0.245^2 + 0.1^2 + 0.05^2} = 8.05 \text{ mm} \tag{8.5}$$

$$X = 0.18 + 2(-0.04) + \sqrt{0.12^2 + 0.1^2 + 0.05^2} = 0.26 \rightarrow 0.25 \text{ mm} \tag{8.6}$$

Each pad is then $(Z - G)/2$ = 0.85 mm long and 0.25 mm wide, centred $(Z + G)/4$ = 4.45 mm from the middle. The footprint in KiCad 10.0.5's library has the same pads, 64 of them at 0.85 × 0.25 mm, with pad 1 at x = −4.45 mm. The exposed pad is drawn at its nominal size, 7.65 mm, which leaves (8.05 − 7.65)/2 = 0.20 mm between it and the inner ends of the pads. KLC rule F6.3 and the generator both ask for at least 0.2 mm, so this footprint meets the minimum exactly. (If the gap had come out smaller, the generator would have moved the pads' inner ends outward to make it 0.2 mm, and the pads would have come out shorter.)

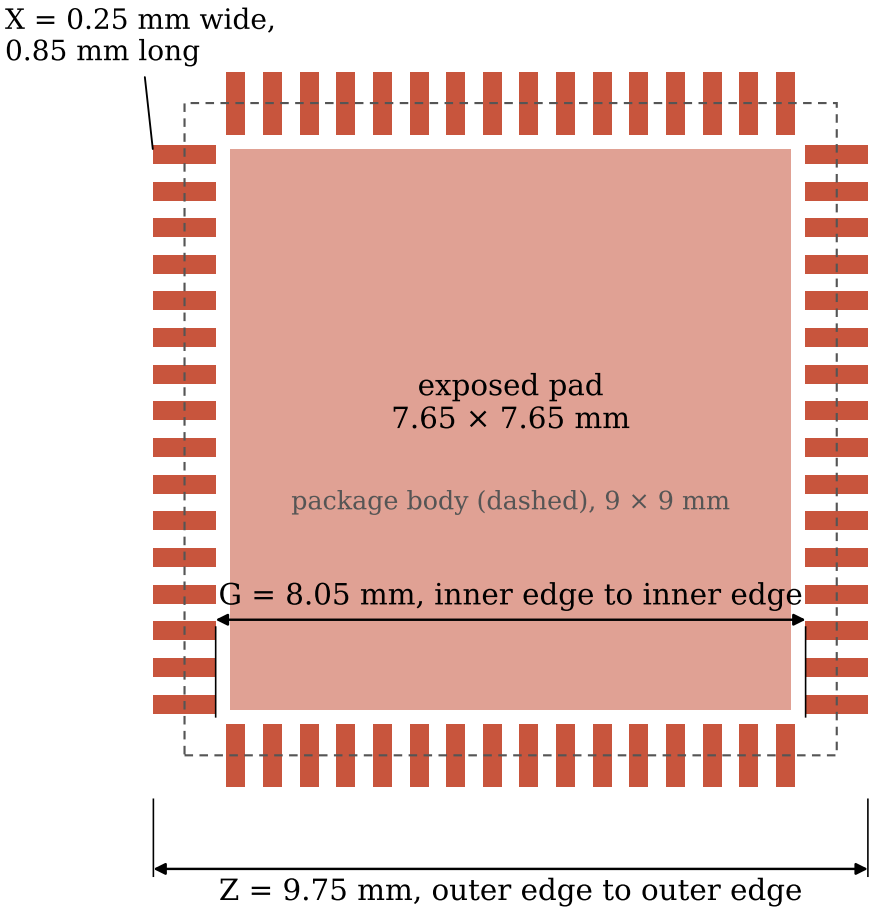


Figure 8.2: The QFN-64 land pattern with its three dimensions

The land pattern worked out above, drawn by `scripts/figures/qfn_land_pattern.py`, which also prints the working.

The generator's entry also gives the exposed pad 6 × 6 separate paste openings. A second version of the footprint, with `_ThermalVias` at the end of its name, adds a 7 × 7 grid of thermal vias with 0.2 mm drills, which carry heat down into the board as in the cut through the QFN above. Add the silkscreen, the outline on the fab layer and the courtyard, and you have the whole footprint that Symbols, footprints and libraries showed, which was the version without thermal vias. In Lab 02 you do the same calculation for a package that KiCad's library doesn't have, and then check your footprint against your own numbers. In the lab, the formula for G and the course's check both take the worst case for S, instead of adding its tolerances as a root sum of squares the way the generator does.

Check yourself

1. Work out Z for the same QFN-64 at the "most" density level, where the toe goal is 0.40 mm. How much longer does each pad get?

   Answer: A1, in Answers at the end.

2. Why might a padframe table number the die's pads from 0 and the board's pads from 1, and why does it matter to you?

   Answer: A2, in Answers at the end.

3. A signal leaves a QFN through a bond wire and goes into the input of another CMOS chip. Which of those two looks inductive to a fast edge, and which capacitive?

   Answer: A3, in Answers at the end.

4. On a chip-on-board module, what takes the place of a QFN's leads, and what rule does that put on the board's solder mask?

   Answer: A4, in Answers at the end.

Next: Lab 02: A footprint from a datasheet.

Answers

A1. Check yourself

$Z = 8.9 + 2(0.40) + 0.229 = 9.93$ mm, which rounds to 9.95 mm, 0.2 mm more than at nominal. G doesn't change, because the heel goal is 0 at every level, so each pad becomes (9.95 − 8.05)/2 = 0.95 mm long. That's 0.1 mm longer than at nominal, and all of the extra length is at the toe.

A2. Check yourself

The table doesn't give a reason, and you can use it without one, because it lists both numberings side by side and each column is right for its own side. It matters because the board is drawn from the table. If you read the wrong column, every connection moves along by one pad, and no check that looks only at the board can find that, because the schematic and the board were both drawn from the same wrong column.

A3. Check yourself

The bond wire is inductive and the CMOS input is capacitive (Hall and Heck, section 3.5.7). Lead frames, vias, sockets and connectors are inductive too.

A4. Check yourself

Pads on the module's own board take the place of the leads, because the wires go from the die straight to copper on the PCB. The wires are bonded to that copper, so wafer.space's rule is that "No solder mask is allowed in the area to be bonded."

9. Lab 02: A footprint from a datasheet

In this lab you make a KiCad footprint for TI's RHL0024A package, which KiCad's library doesn't have. The package is a 24-pin VQFN with a 3.5 × 5.5 mm body and an exposed pad underneath. You work out the size and position of each pad by hand from TI's package drawing, and then draw the footprint in KiCad. You also draw a symbol for the SN74LVC8T245, a level translator that TI sells in this package. The lab's check compares your numbers with the course's, your footprint with your numbers and your symbol with the datasheet's pin table. It then builds a small board with your part on it and runs KiCad's own checks on that board.

This lab takes about 2 h.

Package: `labs/lab-02-a-footprint-from-a-datasheet/`

Before you start

- Finish Lab 01, which installs KiCad and checks that `labs/common/kicad.sh` runs.
- Read Schematics other people can read, Symbols, footprints and libraries and From die to package.
- Download TI's SN74LVC8T245 datasheet. We used revision SCES584D (June 2005, revised May 2026), and the page numbers below are that revision's PDF pages. If TI has revised it since, find the package outline headed "RHL0024A" near the end; the drawing we used is numbered 4225250/C.

The SN74LVC8T245 passes eight signals between its A side and its B side, and each side has its own supply, anywhere from 1.65 V to 5.5 V. You could use one between a Tiny Tapeout chip's 3.3 V pins and a part that runs at another voltage. KiCad 10.0.5's `Package_DFN_QFN` library has 776 footprints. Only two have RHL in their names, `Texas_VQFN-RHL-20` and its thermal-via version, and both are for a 20-pin package, with 21 pads counting the exposed pad.

Part 1: Read the drawing

What does the drawing say about the package, and which of its numbers come with a tolerance?

1. Open the datasheet at page 36, "PACKAGE OUTLINE, RHL0024A". The view at the top of the drawing is the package seen from above. Under it is a side view, and under that is the package seen from below, which shows the 24 terminals and the exposed pad. The drawing numbers the exposed pad 25.
2. Write down these dimensions from the drawing, which are the only ones the lab uses. A dimension given as two numbers, such as 3.6 over 3.4, is a range, and a real part can measure anything between them. A number in a box is a basic dimension, which is exact by definition, and a number in parentheses is for reference only.

Dimension	From the drawing
Body, the short side	3.4 to 3.6
Body, the long side	5.4 to 5.6
Terminal length	0.3 to 0.5 (24X)
Terminal width	0.18 to 0.30 (24X)
Pitch	0.5, boxed
Span of the 10 terminals on each long side	4.5, boxed
Spacing of the 2 terminals on each short side	1.5, boxed
Exposed pad	2.05 ± 0.1 by 4.05 ± 0.1

3. In the view from below, pin 1 is at the bottom left. Where will pin 1 be in your footprint, which KiCad draws from above?

   Answer: A1, in Answers at the end.

Check: you have the eight dimensions above written down, and you know where pin 1 goes.

Part 2: Work out the pads

How big should each pad be, and where does it go?

1. Make a prediction first. TI's example land pattern on page 37 uses pads 0.6 mm long and 0.24 mm wide. Will pads worked out from the IPC method below come out longer or shorter than that?
2. For each pair of opposite sides, work out Z, G and X from the formulas below. The terminals on the long sides (pins 2 to 11 and 14 to 23) point along the short body dimension, so for them L is 3.4 to 3.6. The terminals on the short sides (1, 12, 13 and 24) point along the long one, so for them L is 5.4 to 5.6. T is the terminal length, W the terminal width, and each C is the difference between the maximum and the minimum of the quantity in its subscript:

$$S_{max} = L_{max} - 2T_{min}, \qquad S_{min} = L_{min} - 2T_{max} \tag{9.1}$$

$$Z = L_{min} + 2J_T + \sqrt{C_L^2 + F^2 + P^2} \tag{9.2}$$

$$G = S_{max} - 2J_H - \sqrt{C_S^2 + F^2 + P^2} \tag{9.3}$$

$$X = W_{min} + 2J_S + \sqrt{C_W^2 + F^2 + P^2} \tag{9.4}$$

   S is the distance between the inner ends of two opposite terminals. Z is the distance between the outer ends of two opposite pads, G the distance between their inner ends, and X the pad width. $J_T = 0.30$, $J_H = 0$ and $J_S = -0.04$ mm are IPC-7351B's solder-fillet goals at the toe, heel and side of a no-lead terminal, at nominal density, as KiCad's footprint generator records them. F and P allow for the board maker's and the assembler's tolerances, and this course uses F = 0.10 mm and P = 0.05 mm. Taking C_S as $S_{max} - S_{min}$ in the formula for G assumes that the tolerances that make up S are all at their worst at once, and the course's check does the same. KiCad's footprint generator instead combines those tolerances as a root-sum-square. Write each result to two decimal places in the Z, G and X rows of `pads.md`, in the lab folder.
3. KiCad's library convention asks for at least 0.2 mm between the exposed pad and every other pad. Make the exposed pad's land the drawing's nominal size, 2.05 × 4.05, and work out the gap between it and the inner ends of your pads at G. Where the gap is less than 0.2 mm, move the pads' inner ends out until it's 0.2 mm, which makes the pads shorter. Then work out each pad's length, and the distance from the footprint's centre to the pad's centre, and write them in the "Pad length" and "Pad centre" rows. Write the exposed pad's size on the "Exposed pad:" line at the bottom.
4. Check your numbers:

```
labs/common/kicad.sh     python3     labs/lab-02-a-footprint-from-a-datasheet/src/
check_footprint.py
```

   You should see an `ok` row for `pads.md` and MISMATCH rows for the other three parts, because you haven't drawn the footprint or the symbol yet:

```
  ok       pads.md    Z, G, X, pad lengths and centres within 0.02 mm of the course's
  MISMATCH  Footprint  no labs/lab-02-a-footprint-from-a-datasheet/pcb101.pretty/
RHL0024A.kicad_mod yet
  MISMATCH  Symbol     no symbol called SN74LVC8T245RHL in labs/lab-02-a-footprint-
from-a-datasheet/pcb101.kicad_sym
  MISMATCH  Fit        needs both the footprint and the symbol

FAIL  3 of 4 parts have a MISMATCH: Footprint, Symbol and Fit
```

If a row says a number is more than 0.02 mm from the course's value, recheck that one. Compare your pad length with your prediction.

Answer: A2, in Answers at the end.

Check: the `pads.md` row says `ok`.

Part 3: Draw the footprint

Can you turn your numbers into a KiCad footprint that matches them?

1. Open `labs/lab-02-a-footprint-from-a-datasheet/lab02.kicad_pro` in KiCad, then the Footprint Editor. The project's library tables add the lab folder's `pcb101.pretty` as a library called `pcb101`. Make a new footprint in `pcb101` called `RHL0024A`.
2. Place one pad with the pad tool on the right-hand toolbar, then press E on it to open its properties:

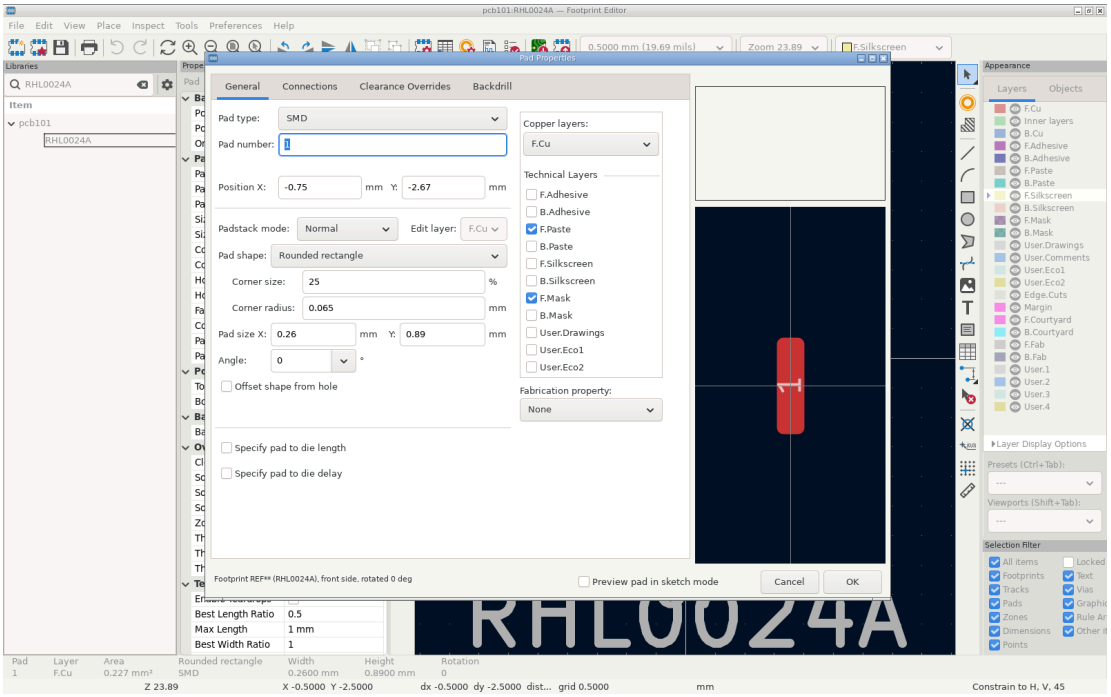


Figure 9.1: KiCad 10.0.5's Pad Properties dialog for pad 1 of the reference footprint: SMD, pad number 1, position −0.75, −2.67 mm, rounded rectangle with a 25 % corner, size 0.26 × 0.89 mm, on F.Cu with F.Paste and F.Mask.

The dialog above shows pad 1 of the reference footprint. Make yours pad 2: SMD, rounded rectangle, 0.89 mm in X by 0.26 mm in Y, at X = −1.67 and Y = −2.25, with your own numbers in place of these if they differ. Leave F.Paste and F.Mask ticked. KiCad's Y axis points down, so negative Y is the top of the footprint.

3. Make the rest of the left column with an array. Select pad 2 and press Ctrl+T:

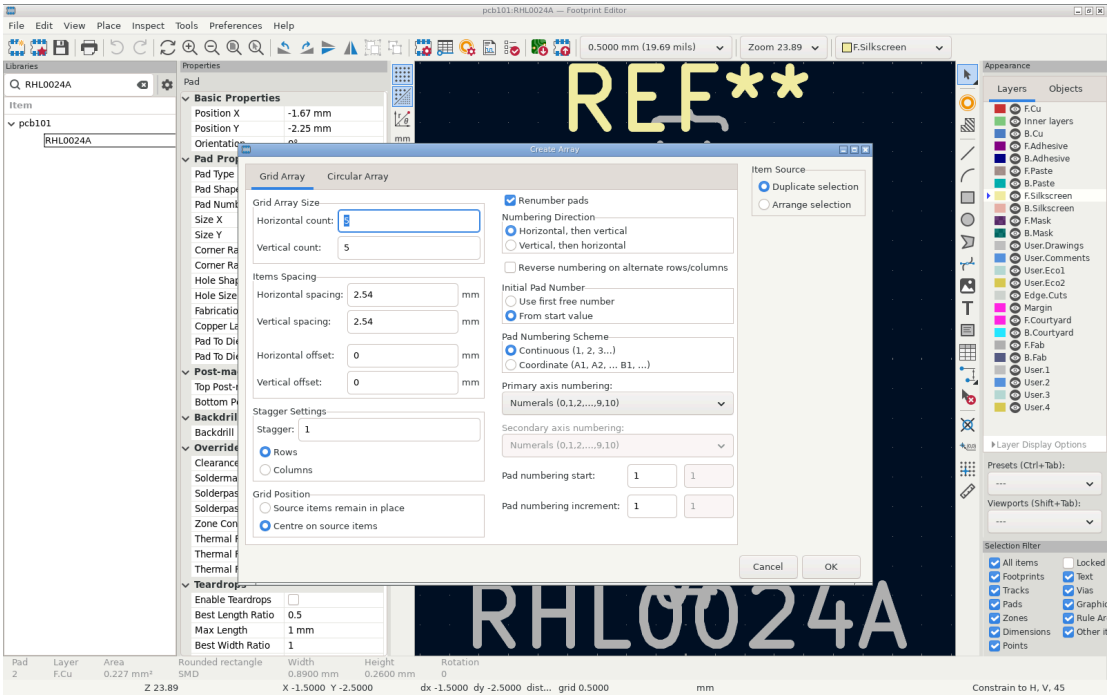


Figure 9.2: The Create Array dialog: Grid Array tab, horizontal and vertical counts, spacings, “Renumber pads”, a pad numbering start, and “Source items remain in place” or “Centre on source items” under Grid Position.

Set a horizontal count of 1, a vertical count of 10 and a vertical spacing of 0.5 mm, choose “Source items remain in place”, keep “Renumber pads” ticked and start the numbering at 2. The pad you selected becomes the first of the ten, so you get pads 2 to 11 down the left side.

4. Make pads 14 to 23 the same way, from a pad 14 at X = 1.67, Y = 2.25. The right column is numbered upward, so give this array a vertical spacing of −0.5 mm, and start its numbering at 14. Then place the four pads on the short sides, each 0.26 mm in X by 0.89 mm in Y. Pads 1 and 24 go at Y = −2.67, and 12 and 13 at Y = 2.67. Pads 1 and 12 are at X = −0.75, and 24 and 13 at X = 0.75.

5. Add the exposed pad: pad 25, a rectangle 2.05 × 4.05 mm at the origin, with F.Paste unticked. Its paste goes on separately, through smaller openings, as it does in KiCad’s own QFN footprints (`QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm`, for example, has 36 of them). You can make the openings in either of two ways. One is to add pads of type “SMD Aperture” with no number, on F.Paste only. The other is to set a negative “Solder paste relative clearance” on pad 25 and tick F.Paste again. On each side, KiCad then pulls the edge of the opening in by that percentage of the pad’s size along the same axis. We plotted a 2.05 × 4.05 mm pad with −10 % and got a 1.64 × 3.24 mm opening, which is 64 % of the pad’s area. Whichever way you choose, the paste should cover between 50 % and 80 % of the pad’s area, which is what KiCad’s library convention asks for. TI’s own stencil on page 38 prints 80 %. The reference uses eight 0.83 × 0.82 mm openings, two across and four down, for 66 %.

6. Draw the rest:

   - on F.Fab, the body outline, 3.5 × 5.5 mm, centred, ideally with the pin 1 corner cut off;
   - on F.Silkscreen, a mark at pin 1 and short lines at the body’s corners, all at least 0.2 mm clear of every pad;
   - on F.Courtyard, a rectangle 0.25 mm outside the pads and the body, which for the reference is ±2.37 × ±3.37 mm.

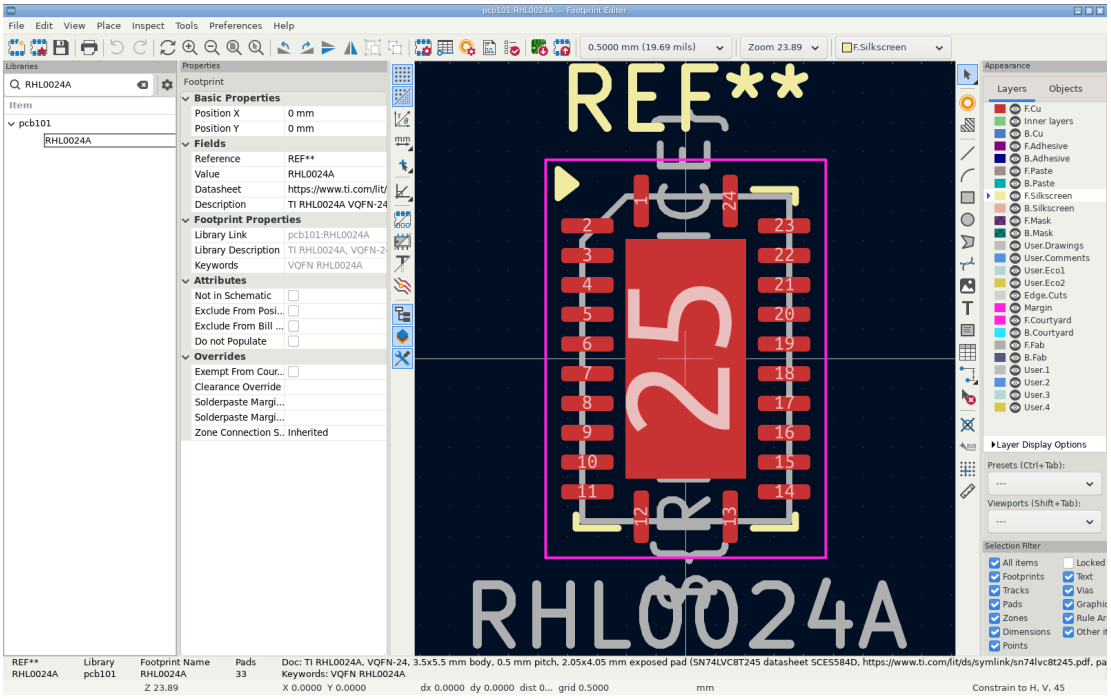


Figure 9.3: The reference footprint in the Footprint Editor, loaded from library pcb101: pads 1 to 24 around exposed pad 25, the pin 1 triangle on the silkscreen, the body outline with its cut corner on F.Fab, and the courtyard.

7. Save, and run the check from Part 2 again. You should see `ok  Footprint  25 pads where your numbers put them, paste 66 %, and matches none of KiCad's 776 QFN footprints`, with your own paste percentage in place of the reference's 66. The check compares every pad with your `pads.md` to within 0.01 mm, and if a pad is off, it names the pad and says where your numbers put it.

Check: the Footprint row says `ok`.

Part 4: Draw the symbol

Which pins does the schematic need, and what kind is each one?

1. KiCad's own library has a symbol called `SN74LVC8T245`. Open it in the Symbol Editor (it's in `Logic_LevelTranslator`) and look at its pins and its footprint filters. Why can't you use it with your footprint as it stands?

   Answer: A3, in Answers at the end.

2. In the Symbol Editor, make a new symbol in the project library `pcb101` called `SN74LVC8T245RHL`. Give it the pins in Table 4-1 on page 5: VCCA on 1, DIR on 2, A1 to A8 on 3 to 10, GND on 11, 12 and 13, B8 to B1 on 14 to 21, OE on 22 and VCCB on 23 and 24. Add pin 25 for the thermal pad, called `EP`. The Pin Table lets you type the pins into a grid rather than placing each one.

3. Set each pin's electrical type, because ERC decides what to report from the pin types. Make the supplies power input, DIR and OE input, and the A and B pins bidirectional. GND has three pins and VCCB has two, and a schematic should have to connect each supply only once, so stack each supply's pins. In KiCad 10 a stack can be a single pin with a list of numbers. Number one GND pin `[11,12,13]` and one VCCB pin `[23,24]`, and KiCad gives that pin's net to every pad in the list. KiCad's library convention (KLC S4.3) asks for these pin stacks from KiCad 10 on, in place of the older way, in which several pins are drawn on top of each other and all but one are hidden. The check accepts either, and the reference was drawn the older way. If you draw it that way, make the hidden pins passive, because KiCad joins a hidden power-input pin to a global net named after it, and then in Part 5 the pads' nets wouldn't match the schematic. If you leave one as a power input, the check's Symbol row names the pin to change. Table 4-1 gives the thermal pad no type. The reference makes it passive, and the check accepts passive, power input, input, bidirectional or unspecified for it.

4. Set the symbol's Footprint field to `pcb101:RHL0024A`, and save.
5. Table 4-1 lists the thermal pad with no pin number, type or description, and note 3 on page 36 says only that it must be soldered to the board. Neither says what to connect it to. Where would you find out?

   Answer: A4, in Answers at the end.

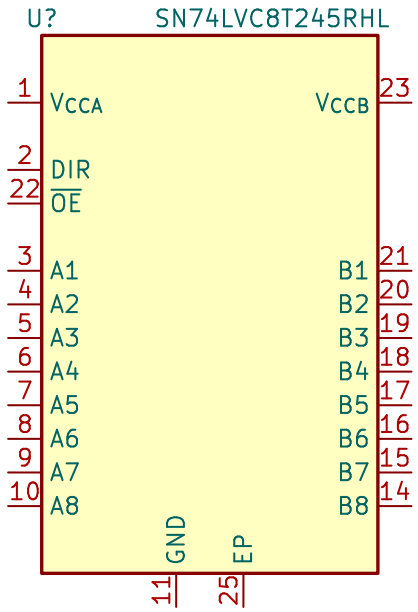


Figure 9.4: The reference symbol SN74LVC8T245RHL, plotted with kicad-cli: VCCA, DIR, OE and A1 to A8 on the left, VCCB and B1 to B8 on the right, GND (pin 11) and EP (pin 25) underneath.

Check: the check's Symbol row says `ok  Symbol  25 pins with Table 4-1's names and types, footprint pcb101:RHL0024A`.

Part 5: Put it on a board

Do your symbol and footprint work together the way KiCad expects?

1. Run the check once more. With both files in place, it now builds a KiCad project with one part in `results/fit/`. The project's schematic has your symbol, a label on every pin, and a PWR_FLAG on each net that has a power-input or input pin, so that ERC sees each of those nets driven. Its board has your footprint, with every pad given the net of its pin. The check then runs `labs/common/check_board.py` on that project. In our three runs, with the container limited to one CPU, the whole check took 18 to 43 s. On the reference solution it ends like this:

```
  ok       pads.md    Z, G, X, pad lengths and centres within 0.02 mm of the course's
  ok        Footprint  25 pads where your numbers put them, paste 66 %, and matches
none of KiCad's 776 QFN footprints
  ok      Symbol    25 pins with Table 4-1's names and types, footprint pcb101:RHL0024A
  ok         Fit        ERC, DRC and parity clean on a board with your part (17 ERC
and 0 DRC warnings)

PASS  labs/lab-02-a-footprint-from-a-datasheet/solutions
```

   The 17 ERC warnings are for the labels on A1 to A8, B1 to B8 and EP. Each of those labels is connected to only one pin, because the schematic has no other part for it to reach. `results/fit/check_board.txt` has the board check's full output.

2. To see what the board check adds, we made every signal pad 0.32 mm wide instead of 0.26, in `pads.md` and in the footprint. The `pads.md` row failed, because 0.32 is more than 0.02 mm from the course's X. The Footprint row still passed, because the pads matched the numbers in `pads.md`, but the board check in the Fit row failed:

```
  MISMATCH  pads.md   X, long sides: 0.32 is more than 0.02 mm from the course's value
  MISMATCH          X, short sides: 0.32 is more than 0.02 mm from the course's value
   ok        Footprint  25 pads where your numbers put them, paste 66 %, and matches
none of KiCad's 776 QFN footprints
  ok      Symbol    25 pins with Table 4-1's names and types, footprint pcb101:RHL0024A
   MISMATCH  Fit        drc.error 18 (clearance 18)
```

   Each of the 18 is `Clearance violation ( clearance 0.2000 mm; actual 0.1800 mm)` between neighbouring pads. Why 18, and why 0.18 mm?

   Answer: A5, in Answers at the end.

3. We also deleted pin 25 from the reference symbol. The board check still passed, because pad 25 just had no net and KiCad's schematic parity check reported nothing. Only the Symbol row failed, with `no pin 25`. That's why the lab's check reads your symbol's pins itself rather than leaving it all to KiCad.

Check: the check prints `PASS`.

Expected results

Check	Expected	
Z, G, X, long sides	4.23, 2.39, 0.26 mm	within 0.02 mm
Z, G, X, short sides	6.23, 4.39, 0.26 mm	within 0.02 mm
Gap to the exposed pad at G	0.17 mm, so the pads are trimmed	exact
Pads	0.89 × 0.26 mm, centres at ±1.67 and ±2.67 mm	within 0.02 mm
Paste on the exposed pad	50 to 80 %	rule
Fit project	0 ERC errors, 0 DRC errors, 0 parity items	exact
`check_footprint.py` run time	18 to 43 s in our runs, with both files in place and one CPU	ballpark

Finishing

You're done when `check_footprint.py` prints `PASS  labs/lab-02-a-footprint-from-a-datasheet` on your own `pads.md`, footprint and symbol.

Questions:

1. Your pads stick out about 0.37 mm past the body; TI's stop at 0.2 mm. Which would you rather solder by hand, and which leaves more room for parts packed around the chip?

   Answer: A6, in Answers at the end.

2. In the fit project, pads 11, 12 and 13 are all on the net GND, and 23 and 24 are on VCCB, but no track connects them, and the check doesn't count that against you. Why is that fine here, and not on your Lab 03 board?

   Answer: A7, in Answers at the end.

Submit: nothing for this lab.

Going further (optional)

- Compare your pads with the QFN-64 on Tiny Tapeout's SKY130 breakout from Lab 01. That board uses KiCad's `QFN-64-1EP_9x9mm_P0.5mm_EP7.65x7.65mm`, which has 0.85 × 0.25 mm pads centred at ±4.45 mm. The footprint's description cites page 415 of the ATmega2560 datasheet, so the footprint under the Tiny Tapeout chip was drawn from another chip's package drawing.
- Run `labs/common/kicad.sh python3 labs/lab-02-a-footprint-from-a-datasheet/src/selftest.py`. It runs the check on seven folders whose verdicts are known in advance. One of them holds KiCad's `QFN-24-1EP_4x4mm_P0.5mm_EP2.6x2.6mm` renamed to `RHL0024A`, and the check fails it with `these pads are the same as KiCad's ....`

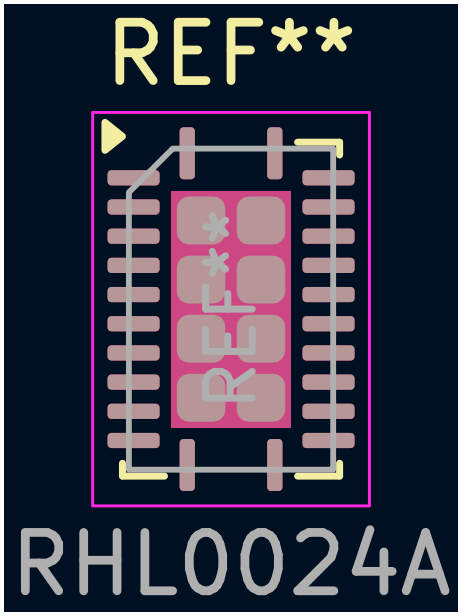


Figure 9.5: The reference footprint RHL0024A, plotted with kicad-cli: copper pads in red, the paste openings on the exposed pad, silkscreen in yellow, the body outline and the courtyard.

Next on the Course path: read Layers and fabricator rules, Placement and routing, Pins, Pmods and planning a pinout and Parts you can solder by hand, then do Lab 03: A Pmod for one project's pins.

Answers

A1. Part 1: Read the drawing

Pin 1 goes at the top left. The view from below is the view from above turned over, and on this drawing it's turned top to bottom, so pin 1 moves from the bottom left to the top left. Seen from above, the numbering then runs counterclockwise. Pin 1 is on the top short side, 2 to 11 run down the left side, 12 and 13 are on the bottom, 14 to 23 run up the right side, and 24 is on the top next to pin 1. TI's example board layout on page 37 is drawn from above and numbers its pads the same way, and KiCad's library convention also puts pin 1 at the top left.

A2. Part 2: Work out the pads

On the long sides, Z = 4.23, G = 2.39 and X = 0.26. On the short sides, Z = 6.23, G = 4.39 and X is 0.26 again. At G, the gap to the exposed pad is 0.17 mm on both pairs of sides, which is less than 0.2 mm, so the inner ends move out to 1.225 mm (long sides) and 2.225 mm (short sides) from the centre. Every pad ends up 0.89 × 0.26 mm, and its centre is 1.67 mm (long sides) or 2.67 mm (short sides) from the middle of the footprint. That's longer than TI's 0.6 mm. The formula for Z, with its 0.30 mm toe goal, puts the outer end of each pad about 0.37 mm beyond the nominal edge of the body, so that a solder fillet can form there. TI's pads end 0.2 mm beyond it. The course's check holds you to the IPC method so that everyone's numbers can be checked against the same answer.

A3. Part 4: Draw the symbol

It has 24 pins and no pin for the exposed pad, so pad 25 would get no net. Its footprint filters include `Texas*VQFN*RHL*`, which in KiCad 10.0.5's library matches only `Texas_VQFN-RHL-20` and `Texas_VQFN-`

`RHL-20_ThermalVias`. Both are footprints for a 20-pin package, and the description of each cites the datasheet of a different part. In a filtered list, a footprint for the wrong pin count can look plausible, and that's one reason to read the drawing yourself.

A4. Part 4: Draw the symbol

Note 4 on page 37 points to TI's application note SLUA271 on soldering a thermal pad, and you can ask about this part on TI's support forum. Giving the pad its own pin, EP, puts whatever you decide in the schematic, where a reviewer can see it.

A5. Part 5: Put it on a board

Pads 0.32 mm wide at a 0.5 mm pitch leave 0.5 − 0.32 = 0.18 mm of copper-free space between neighbours, and the board's default clearance is 0.2 mm. Each long side has 10 pads and so 9 gaps, and there are two long sides. The two pads on each short side are 1.5 mm apart, centre to centre, which leaves far more than the clearance between them.

A6. Finishing

The longer IPC pads are easier to solder by hand, because there's more copper outside the body for the iron to heat and for you to see the fillet on. TI's shorter pads make the courtyard smaller, which matters when other parts are packed close around the chip. Neither is wrong, and the drawing doesn't tell you which to use, so it's a decision you make for each board.

A7. Finishing

The fit project is a board with one part and no tracks, made only to test the part, so pads that share a net are bound to be unconnected. On the reference, `check_board.py` reported 3 unconnected items, and the project's `check-expect.json` leaves unconnected items out of the PASS or FAIL. On a real board every net has to be routed, and unless a `check-expect.json` says otherwise, the board check counts unconnected items as failures.

III 3. A board of your own

10 **Layers and fabricator rules 71**
10 Stackups 71
10 The copper layers in use 72
10 Fabricators' minimums 73
10 Telling KiCad the rules 74
10 The DIP dock against JLCPCB's rules 75
10 Check yourself 76
10 Answers 76

11 **Placement and routing 77**
11 Connectors first 77
11 Decoupling next to the pin it serves 78
11 Track widths, clearances and vias 79
11 Return paths and the ground plane 80
11 When a trace stops being a wire 80
11 Check yourself 81
11 Answers 82

12 **Pins, Pmods and planning a pinout . 83**
12 A project's pins 83
12 The three I/O Pmods 83
12 What else is on each pin 84
12 The RP2350's UARTs and the recommended pinouts 84
12 SimProc's pins 85
12 Planning a pinout before tapeout 85
12 Check yourself 85
12 Answers 86

13 **Parts you can solder by hand 87**
13 Sizes and their codes 87
13 KiCad's hand-soldering footprints 88
13 Part sizes on Tiny Tapeout's boards 88
13 Packages with pads you can't reach 88
13 Through-hole parts 88
13 Fabricator assembly and safety 89
13 Check yourself 89
13 Answers 89

14 **Lab 03: A Pmod for one project's pins 91**
14 Before you start 91
14 Part 1: Start your board repository 91
14 Part 2: Draw the schematic 92
14 Part 3: Place the connectors 94
14 Part 4: Place the rest and route 95
14 Part 5: Check the connections against SimProc's pin list 96
14 Finishing 97
14 Going further (optional) 97
14 Answers 97

10. Layers and fabricator rules

This page answers: *what is a circuit board made of, and how do I tell KiCad what my fabricator can actually make?*

A board design in KiCad is a description that a fabricator turns into fibreglass and copper. It has to describe something the fabricator can make, with no track narrower, no gap smaller and no hole smaller than the fabricator's machines allow, and it has to say what goes on each layer. This page starts with the layers of two of Tiny Tapeout's boards, as their board files record them. Next it sets what three fabricators say they can make beside the numbers that KiCad's own footprint libraries use. The last two sections show how you write a fabricator's limits into a board so that DRC checks them, and what DRC found when we did that to Tiny Tapeout's DIP dock. In Lab 03 you draw a 2-layer board that already has JLCPCB's limits set in it.

The fabricators' numbers on this page come from their own pages, read on November 16, 2026. They change from time to time, so check the page of the fabricator you're ordering from before you rely on them.

Stackups

A board is a stack of copper layers separated by sheets of dielectric, with solder mask over the outer copper and silkscreen printed over the mask. KiCad records this stack, which is called the **stackup**, in the board file. Here are the stackups of two of Tiny Tapeout's boards, drawn from their board files:

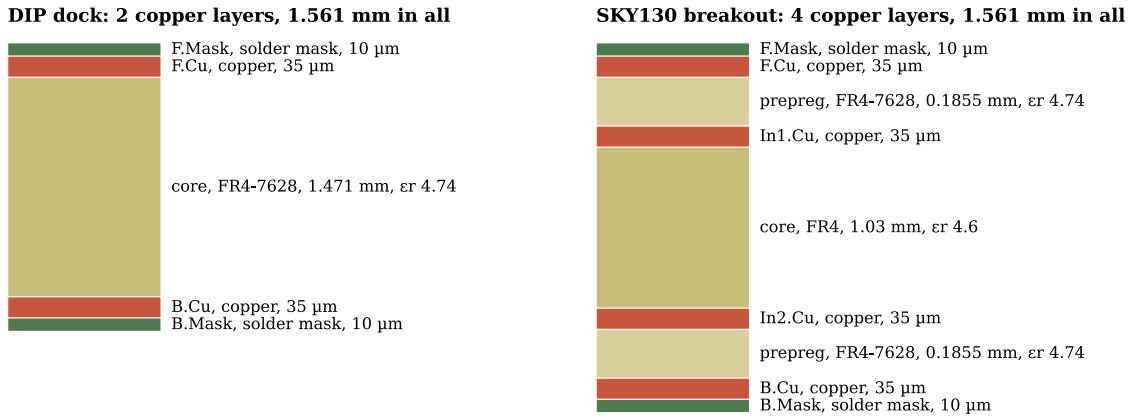


Figure 10.1: The stackups of the 2-layer DIP dock and the 4-layer SKY130 breakout

The stackups of Tiny Tapeout's DIP dock and SKY130 breakout, from the `(stackup …)` section of each board file at `breakout-pcb` 91a2bf1. Copper and mask are drawn thicker than they are, so that you can see them. Drawn by `scripts/figures/stackups.py`.

Both boards are 1.561 mm thick in all, which is the sum of the thicknesses of their layers. The copper on every layer is 35 µm thick, which PCBWay's capabilities page calls "1oz", because fabricators give copper thickness as a weight per square foot. The dielectric between the copper layers is FR4, a laminate of glass fibre and epoxy. A fabricator makes a board by pressing together two kinds of FR4 sheet, **core** and **prepreg**, and the board files list each dielectric layer as one or the other, with its relative permittivity, εr. A 2-layer board is one core with copper on both sides. The 4-layer breakout has a 1.03 mm core in the middle and a thin 0.1855 mm prepreg between each outer layer and the inner layer next to it, so each outer layer has an inner layer only 0.1855 mm away.

That closeness is the main reason to use a 4-layer board. At high frequencies, a signal's return current flows in the nearest conductor under its track. When a gap in that conductor makes the return current detour, the loop that the current flows around gets bigger. Hall and Heck (*Advanced Signal Integrity for High-Speed Digital Designs*, chapter 10) say that the most fundamental effect of such a detour is an increase

in the series inductance of the path, by an amount that depends on how far the current has to go out of its way. A solid ground layer 0.19 mm under the top layer gives every top-layer signal a return path right below it.

The copper layers in use

On the 2-layer DIP dock, tracks run on both copper layers (the board file has 353 track segments on F.Cu and 182 on B.Cu), and a zone on the `GND` net fills the space that's left on both:

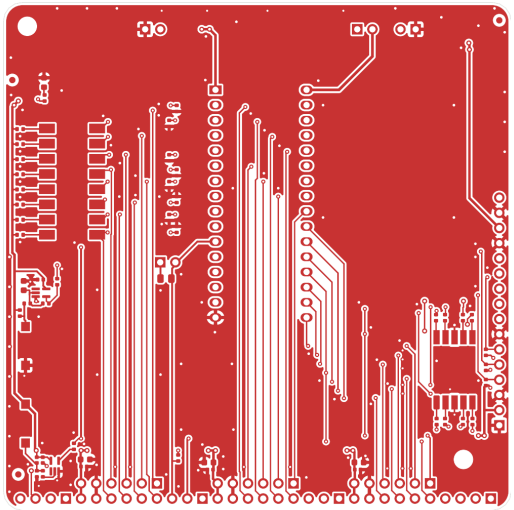


Figure 10.1: F.Cu, the DIP dock's top copper

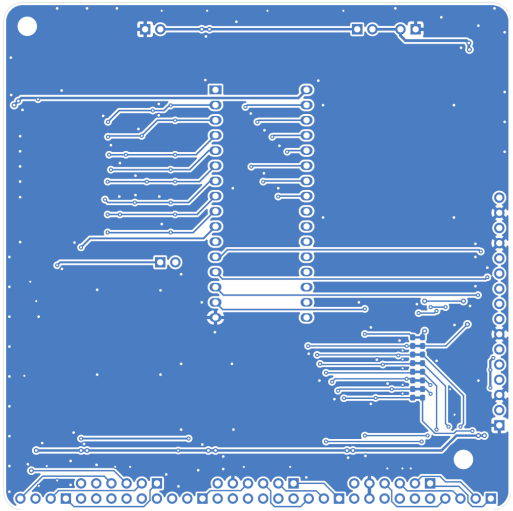


Figure 10.2: B.Cu, the DIP dock's bottom copper

The two copper layers of the DIP dock, plotted by `kicad-cli` 10.0.5 with the board outline. Made by `scripts/figures/kicad_figures.sh`, as is the next plot.

On the 4-layer SKY130 breakout, most of the tracks are on the outer layers (590 segments on F.Cu and 404 on B.Cu), and a few are on the inner ones (104 on In1.Cu and 41 on In2.Cu). A ground zone fills all four layers around the tracks. The first inner layer, In1.Cu, lies 0.19 mm under the top layer, which is where the chip is, and it's almost solid ground:

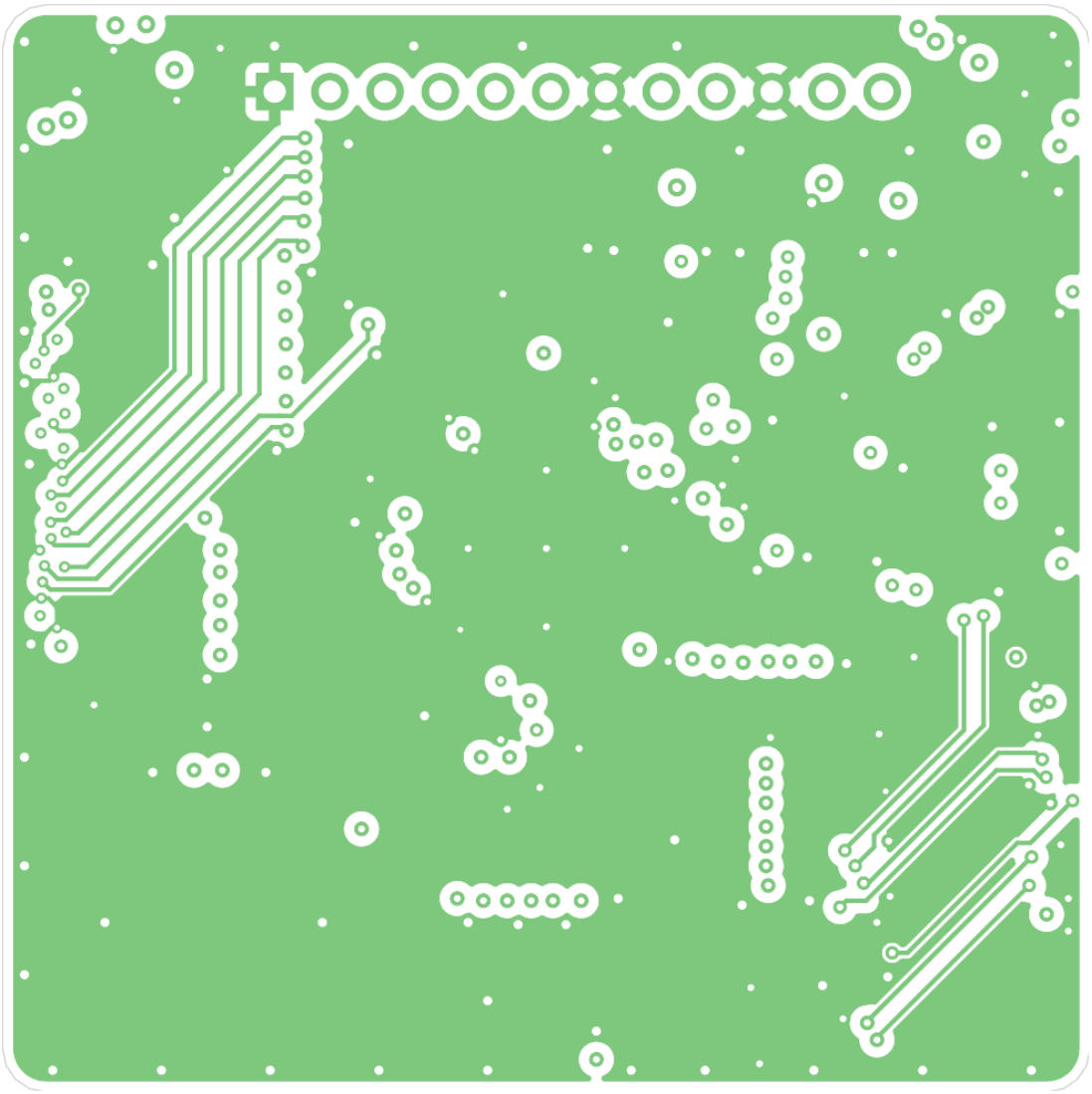


Figure 10.2: In1.Cu, the SKY130 breakout's first inner layer

In1.Cu of the SKY130 breakout: a ground plane, broken only by vias, the through-hole header along the top edge, and a few tracks.

The fabricator works from more layers than the copper. As Symbols, footprints and libraries showed, the mask layers say where the solder mask is opened, the paste layers become the stencil, and the silkscreen layers are printed onto the board. `Edge.Cuts` holds the board's outline, which the fabricator routes out. The fab and courtyard layers aren't made into anything on the board. The fab layers are drawings for people, and DRC uses the courtyards to check that parts don't overlap.

Fabricators' minimums

Each fabricator publishes its own minimums, and they differ. The table gives them for three fabricators that students often use, beside the numbers that the KiCad Library Convention (KLC) sets for KiCad's own footprints:

	JLCPCB	PCBWay	OSH Park	KLC footprints
Track width and spacing	0.10 / 0.10 mm on 1 and 2 layers, 0.09 / 0.09 mm on	0.1 mm / 0.1 mm	2 layers: 6 mil (0.1524 mm) / 6	

	JLCPCB	PCBWay	OSH Park	KLC footprints
	more (1 oz copper)		mil; 4 layers: 5 mil (0.127 mm) / 5 mil	
Smallest drill	0.15 mm (0.1 mm only for boards ≤ 1 mm thick with an ENIG or OSP finish, "more costly")	0.15 mm	10 mil (0.254 mm)	
Plated through-hole annular ring	2 layers: "Recommended 0.25 mm or above; absolute minimum 0.18 mm"; more layers: 0.20 mm recommended, 0.15 mm minimum	0.15 mm	2 layers: 5 mil (0.127 mm); 4 layers: 4 mil (0.1016 mm)	at least 0.15 mm (F7.5)
Via	0.15 mm hole in a 0.25 mm pad at least			
Silkscreen line	"≥0.15mm"	character width 0.15 mm		0.10 to 0.15 mm, nominally 0.12 mm (F5.1)
Silkscreen text height	1.0 mm	0.8 mm		reference text 1.00 mm (F5.1)
Pad to silkscreen	0.15 mm			0.2 mm recommended (F5.1)

Sources: JLCPCB's capabilities page, PCBWay's capabilities page, OSH Park's services page, and KLC's source at commit 6e8cbad. An empty cell means that page doesn't give the number, not that there's no limit.

In two of the rows, KLC is looser than a fabricator. KLC's footprints draw silkscreen at 0.12 mm, under JLCPCB's 0.15 mm minimum line width and PCBWay's 0.15 mm minimum character width, and JLCPCB's page adds that "Characters width less than 0.15mm will be unidentifiable". KLC's minimum annular ring of 0.15 mm is also under JLCPCB's 2-layer absolute minimum of 0.18 mm. So a board made from KiCad's own footprints can still break a fabricator's rules. To find out where, you give DRC the fabricator's rules and let it check the board against them, as the next two sections show.

Telling KiCad the rules

KiCad keeps a board's rules in two places. **Board Setup** has a Constraints section with the board-wide minimums: clearance, track width, via size and annular ring, hole sizes and clearances, copper-to-edge clearance, and silkscreen text height and thickness. They're saved in the project's `.kicad_pro` file, together with the **net classes**, which set the clearance, track width and via size for groups of nets. For example, the DIP dock's constraints ask for clearances of at least 0.16 mm, tracks at least 0.16 mm wide, vias at least 0.5 mm across with an annular ring of at least 0.125 mm, holes at least 0.25 mm across, 0.5 mm between copper and the board edge, and silkscreen text at least 0.8 mm high and 0.08 mm thick. Its Default net class asks for 0.2 mm clearance and 0.22 mm tracks, and a Power class for 0.25 mm of each.

Custom rules go in a `.kicad_dru` file beside the project, written in KiCad's rules language. Each rule has a name, an optional condition that picks the objects it applies to, and one or more constraints. Here is one of the SKY130 breakout's own rules. It's meant to let tracks and gaps shrink to 0.16 mm where they have to squeeze between the pins of a fine-pitch QFN, while the rest of the board keeps the wider values of its net classes:

```
(rule "neckdown QFN48"
   (constraint track_width (min 0.16mm) (opt 0.2mm) (max 0.25mm))
   (constraint clearance (min 0.16mm) (opt 0.16mm) (max 0.2mm))
   (condition "A.intersectsCourtyard('Package_DFN_QFN:QFN-48*') && A.Type != 'Zone'")
)
```

Its condition limits it to objects, other than zones, that overlap the courtyard of a footprint whose name starts with `QFN-48`. The breakout's chip is a QFN-64, so on this board the rule matches nothing. Rules files get copied from board to board. The DIP dock's `.kicad_dru` is byte for byte the same as the demo board's, and it has rules for a QFN-80 and a USB-C connector, neither of which is on the dock. A rule that matches nothing is harmless, but it isn't checking what its name says, so read the rules you inherit before you rely on them.

KiCad 10 added a graphical editor for these rules. The announcement of KiCad 10.0.0 describes it as "a new rule editor dialog that enables users to create custom design rules in a graphical workflow". The KiCad 10 manual says that you open it from **Tools → Design Rule Editor**, and that it stores its rules in the same `.kicad_dru` file as rules written by hand, so you can write rules either way.

The DIP dock against JLCPCB's rules

To see what a fabricator's numbers do to a real board, we added JLCPCB's 2-layer minimums to a copy of the DIP dock's `.kicad_dru` and ran DRC on it in the course's pinned image. These are the rules we added, after the dock's own:

```
(rule "JLCPCB track width"
   (constraint track_width (min 0.10mm)))
(rule "JLCPCB clearance"
   (constraint clearance (min 0.10mm)))
(rule "JLCPCB copper to board edge"
   (constraint edge_clearance (min 0.2mm)))
(rule "JLCPCB PTH annular ring"
   (condition "A.Type == 'Pad' && A.Pad_Type == 'Through-hole'")
   (constraint annular_width (min 0.18mm)))
(rule "JLCPCB silkscreen text"
   (condition "A.Layer == 'F.Silkscreen' || A.Layer == 'B.Silkscreen'")
   (constraint text_height (min 1.0mm))
   (constraint text_thickness (min 0.15mm)))
(rule "JLCPCB silkscreen lines"
    (condition "A.Type == 'Graphic' && (A.Layer == 'F.Silkscreen' || A.Layer ==
'B.Silkscreen')")
   (constraint assertion "A.Line_Width >= 0.15mm"))
```

KiCad's rules language has constraints for track width, clearance, edge clearance, annular width and silkscreen text, but none for the width of a silkscreen line, so the last rule uses an **assertion** instead. An assertion is a constraint that, in the manual's words, "Checks that the boolean expression is true". Here the expression tests the line width of each silkscreen shape.

Without the added rules, DRC finds no errors on the dock apart from six courtyard overlaps that its designer marked as excluded. With them, it reports:

- No track, clearance or edge-clearance violations. The dock's own minimums (0.16 mm tracks and gaps, 0.5 mm from the edge) are all well inside JLCPCB's.
- 4 `annular_width` errors, each "Annular width (rule 'JLCPCB PTH annular ring' min annular width 0.1800 mm; actual 0.1600 mm)", on four stitching vias that the board draws as footprints (`StitchingVia-0.3mm`, a 0.62 mm pad with a 0.3 mm hole). They work as vias, and JLCPCB's via rule (a 0.15 mm hole in a 0.25 mm pad) would pass them, but the board file records them as through-hole pads, and DRC can only check them as what the file says they are.
- 101 `text_thickness` warnings, 98 of them for text 0.14 mm thick and the other 3 for "TrueType font characters with insufficient stroke weight". There are also 38 `text_height` warnings, for text 0.8 mm (29) or 0.9 mm (9) high, which is within PCBWay's 0.8 mm minimum but under JLCPCB's 1.0 mm.
- 199 `assertion_failure` errors for silkscreen shapes thinner than 0.15 mm. DRC lists no more than 199 violations of one kind, and the board has more of these than that. When we ran the same assertion on smaller groups of footprints, each with fewer than 199 such shapes, every group reported all of its

shapes, and together they came to 486. One of them is drawn on the board itself, 0.1 mm wide. 284 are in footprints from Tiny Tapeout's library: 168 in the three Pmod sockets, 64 in the eight solder jumpers, 30 in the DIP switch and 22 in the 7-segment display. 167 are in footprints from KiCad's own libraries: 74 in resistors, 42 in pin headers, 20 in capacitors, 12 in pin sockets, 8 in SOT packages, 8 in the DIP socket and 3 in an LED. Every one of those footprint shapes is 0.12 mm wide, KLC's nominal width. The last 34 are the filled shapes of two logos, each of which is on both sides of the board. They have an outline 0.01 mm wide or none at all, and the assertion counts them too.

None of this means that the DIP dock can't be made. JLCPCB's page warns that characters thinner than 0.15 mm will be "unidentifiable", but it doesn't say that it will refuse a board that has them, and the dock was designed with Tiny Tapeout's own rules. With a fabricator's rules in the file, DRC lists the places where the board falls short of that fabricator's numbers, up to 199 of each kind, and you can fix each one or accept it with a reason. Without those rules, you'd be finding them by eye.

Check yourself

1. A through-hole pad is 1.4 mm across with a 1.0 mm hole. What's its annular ring, and is it within JLCPCB's 2-layer numbers?

   Answer: A1, in Answers at the end.

2. Your Lab 03 board uses KiCad's 0805 hand-soldering resistor footprints, whose silkscreen lines are 0.12 mm wide, and you plan to order from JLCPCB. Will DRC warn you that those lines are too thin?

   Answer: A2, in Answers at the end.

3. On the SKY130 breakout, why does it help that In1.Cu is almost solid ground and sits 0.19 mm below F.Cu, rather than at the middle of the board?

   Answer: A3, in Answers at the end.

4. You inherit a `.kicad_dru` with a rule whose condition is `A.intersectsCourtyard('Package_DFN_QFN:QFN-48*')`, and your board has no QFN-48. What does the rule do?

   Answer: A4, in Answers at the end.

Next: Placement and routing.

Answers

A1. Check yourself

(1.4 − 1.0) / 2 = 0.2 mm. That's above JLCPCB's 2-layer absolute minimum of 0.18 mm but below its recommended 0.25 mm, so it can be made, but JLCPCB would rather you made the pad 1.5 mm or more.

A2. Check yourself

Only if your rules ask it to. KiCad has no built-in check of silkscreen line width, so DRC says nothing about it unless your `.kicad_dru` includes an assertion like the one on this page. The Lab 03 template's `pmod.kicad_dru` has only the annular-ring rule, so as shipped it won't warn you. If you add the assertion and it does warn you, you can thicken the lines in a copy of the footprint, or accept that JLCPCB may print them poorly.

A3. Check yourself

At high frequencies, a signal's return current flows in the nearest conductor under its track. With a ground plane 0.19 mm below the top layer, every top-layer track has its return path right underneath it, and the loop that the current makes is small. If the plane were at the middle of the board, about 0.7 mm below F.Cu, or if it were broken by gaps, the return current would spread out or detour, which adds inductance to the path.

A4. Check yourself

Nothing. No object on your board matches its condition, so its constraints are never applied. The rule is harmless, but it isn't checking what its name suggests, so read the conditions of the rules you inherit.

11. Placement and routing

This page answers: *in what order do the parts go onto a board, and what decides how wide each trace is, where it runs, and which path its current takes back?*

You'll lay out your first board, a 2-layer Pmod for one Tiny Tapeout project, in Lab 03. Layers and fabricator rules covers the numbers a fabricator will hold you to and how they become KiCad's design rules, and this page is about the choices you make inside those rules. Its worked examples are two of Tiny Tapeout's own boards. The 2-layer DIP dock is the closest thing to the board you'll draw, and the 4-layer SKY130 breakout is the board that carries the chip.

Connectors first

Place the parts whose positions someone else has already decided, and lock them, before you place anything else. On a Pmod that means the connectors, because they have to line up with the sockets on the demo board. Tiny Tapeout's Pmod library says that when a board uses more than one Pmod connector, the connectors sit 0.9 in (22.86 mm) apart, with the input connector on the left, the bidirectional one in the middle and the output one on the right, looking at the board from the top. Its README suggests KiCad's Position Relative To (Shift+P) with an X offset of 22.86 mm to get the spacing exactly.

A Pmod plugs into either the demo board or the DIP dock, and the two boards put their sockets in the same places. In the demo board's file, the three I/O sockets J14, J15 and J16 sit at x = 92.70, 115.56 and 138.42 mm, all at y = 93 mm, and in the DIP dock's file its sockets J5, J6 and J7 are at exactly the same coordinates, so a Pmod drawn for one board fits the other.

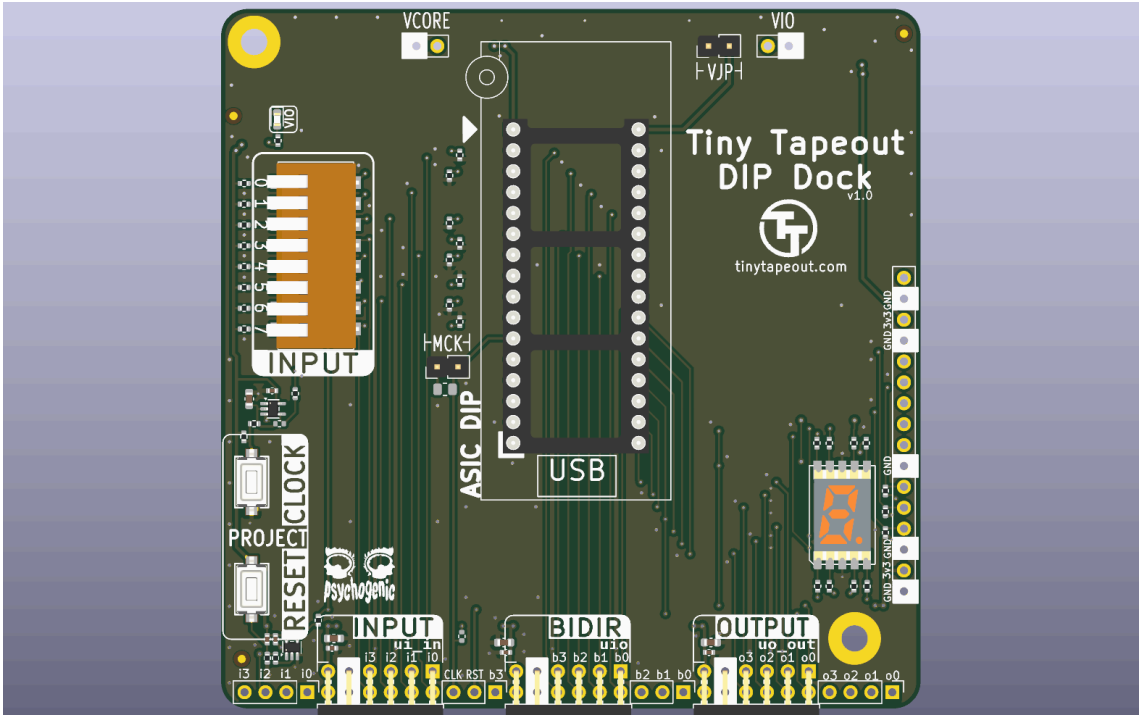


Figure 11.1: The DIP dock from above

Tiny Tapeout's DIP dock (`TinyTapeout/breakout-pcb` at 91a2bf1, Apache-2.0), rendered from above by `kicad-cli` 10.0.5, with the parts' 3D models. The board is 85 × 85 mm with two copper layers. The three Pmod sockets along the bottom edge are labelled INPUT, BIDIR and OUTPUT on the silkscreen and are 22.86 mm apart.

After the connectors, place the parts their signals go to, in the order that keeps each signal's path short. Then place the small parts that serve those parts: decoupling capacitors next to the pins they decouple, and the buffers, series resistors and ESD parts that Protecting the chip's I/O and Supplies, back-powering and ESD add. Take your time over this, because routing can't fix a placement that leaves a signal no short path.

Decoupling next to the pin it serves

A CMOS chip draws its supply current in short pulses, one at every edge, and each pulse has to come through the inductance of the wiring between the chip and wherever the charge is stored. Horowitz and Hill work an example (*The Art of Electronics*, 3rd ed., p. 857) in which a 74LVC output at 3.3 V driving 25 pF with 2 ns edges draws 40 mA. A ground wire has roughly 5 nH/cm of inductance, so one inch of it turns that current step into about 0.2 V of noise on the chip's ground. Their remedy (p. 856) is a ground plane and plenty of bypass capacitors. They write that "it is best to use a 0.1 µF ceramic capacitor near each IC, although one capacitor for every two or three ICs may suffice", and they recommend a few larger capacitors scattered through the circuit as well. The capacitor supplies the pulse from close by, so the pulse's loop is the short one from the capacitor through the chip and back, and the rest of the supply only has to deliver the average current.

To see how this is done around a QFN, we measured the SKY130 breakout's board file, from the centre of each capacitor's pad to the centre of the nearest chip pad on the same rail. Nine of its twelve 100 nF capacitors are within 2.6 mm of the chip pin they serve, and two more, C21 and C22, are on the regulator outputs. The twelfth, C25, is on `VDD_IO` too, but the schematic draws it beside U3, the footprint for an EEPROM that isn't fitted, and on the board it's 1.9 mm from U3′s supply pin and 18.5 mm from the chip. The 1 µF capacitors are further out. C2, C8 and C5 are 3.7 to 5.5 mm from the chip's pins, six more sit beside the 0 Ω links at the top of the board, and the last two, C18 and C19, are on the regulator outputs with C21 and C22. The two 10 µF capacitors sit at the regulator inputs.

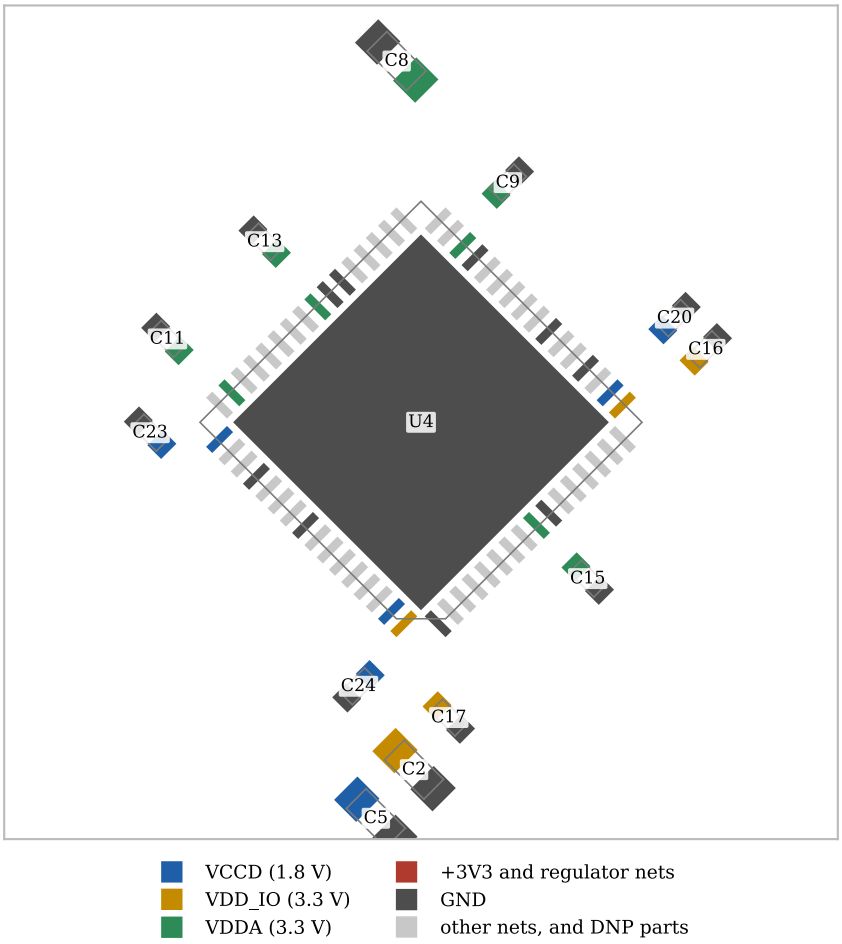


Figure 11.2: The SKY130 breakout around the chip U4

A 24 × 24 mm patch of Tiny Tapeout's SKY130 QFN breakout, rev 3.2 (`TinyTapeout/breakout-pcb` at 91a2bf1), drawn to scale from the board file by `scripts/figures/sky_breakout_map.py`. Pads are coloured by the supply they're on; tracks and copper pours are left out. The chip U4 is placed at 135°, so its rows of pads run diagonally, and the 0402 capacitors C9, C11, C13, C15, C16, C17, C20, C23 and C24 sit just off its edges, each by a pin of its own colour. C2, C5 and C8 are 1 µF.

Track widths, clearances and vias

A fabricator's capability page gives the smallest track and gap it can make, and you shouldn't design at that limit everywhere. JLCPCB's page, read on November 16, 2026, gives a minimum track width and spacing of 0.10 mm (4 mil) on 1- and 2-layer boards with 1 oz copper, and a minimum via of 0.15 mm hole and 0.25 mm diameter. Tiny Tapeout's 2-layer dock draws at roughly twice that. Here are the numbers we read from the two boards' project and board files:

	DIP dock (2 layers)	SKY130 breakout (4 layers)
Default track width	0.22 mm	0.2 mm
Default clearance	0.2 mm	0.2 mm
Power tracks	0.25 mm (`Power` class)	0.25 mm (`PowerRail` class)
Default via (diameter / drill)	0.65 / 0.3 mm	0.6 / 0.3 mm
Track widths used	0.22 to 0.45 mm	0.16 to 0.55 mm; 693 of 1,139 tracks are 0.2 mm

	DIP dock (2 layers)	SKY130 breakout (4 layers)
Board-wide minimum, track and clearance	0.16 mm	0.16 mm

The dock's defaults are the same as the demo board's, and so is its custom rules file. The rules in it that let copper sit closer together apply only in and around two fine-pitch parts on the demo board, the connector the breakout plugs into and the RP2350′s QFN-80, and there they let the gap shrink from the net class's 0.2 mm to as little as 0.16 mm. (On the dock, which has neither part, those rules match nothing.) Outside those parts, DRC holds a track's width only to the board-wide 0.16 mm minimum, so the net class's 0.22 mm is the width you route with by default rather than a limit, and every track on the dock is 0.22 mm or wider. We suggest you do the same, drawing comfortably wide by default and going narrower only where the pads leave no room.

Wider tracks also have less resistance and less inductance, which matters for supply tracks. Hall and Heck model a via as the inductance of its barrel between the capacitances of its two pads, and in their worked example (*Advanced Signal Integrity for High-Speed Digital Designs*, Example 9-9) the barrel is about 0.7 nH and each pad about 0.3 pF. For a decoupling capacitor, the vias from its pads to the planes are part of the loop you're trying to keep small, so they belong right at the pads.

Return paths and the ground plane

Every current returns to its source, and at high frequencies it takes the path of least impedance, which runs through the reference plane directly under the trace that carried it out. Hall and Heck (§10.2) call any other path a nonideal return path. Their examples include a gap in the plane or a crowd of via holes that makes the current detour, and a bond wire that it has to return through. A detour makes the current's loop bigger, and the most basic effect of that is extra series inductance in the signal's path, which grows with the length of the detour.

On a 2-layer board, the bottom layer is usually the plane. The DIP dock pours ground over both layers from one zone, `GNDPOUR`, which covers F.Cu and B.Cu with 0.35 mm clearance to pads. It joins the two pours with 107 stitching-via footprints and with the 21 of its 111 vias that are on GND. Its other 90 vias carry +3V3 and signals. When you route a signal on the bottom layer, you cut a slot in that pour, and every signal crossing the slot on top has its return current pushed around the end of it. Route on the top where you can, keep any run on the bottom short, so that the detour it causes is short too, and don't let a pour be cut into islands.

When a trace stops being a wire

A trace behaves like a plain wire only while its delay is short compared with the edges it carries. Hall and Heck give two rules of thumb:

- A piece of line can be modelled as one lumped inductance and capacitance when its delay is at most $t_r/10$ (§3.2.3).
- Reflections from the line's ends are hidden inside the edge when $t_r > 2t_d$, where t_d is the line's delay (§3.5.6).

A signal in FR-4 travels at $v = c/\sqrt{\varepsilon_r}$. With $\varepsilon_r = 4.5$, the value in Hall and Heck's Example 3-1, that's 141 mm/ns, or about 7 ps per millimetre. A trace on an outer layer runs partly in air and is a little faster, so lengths worked out this way are on the safe side.

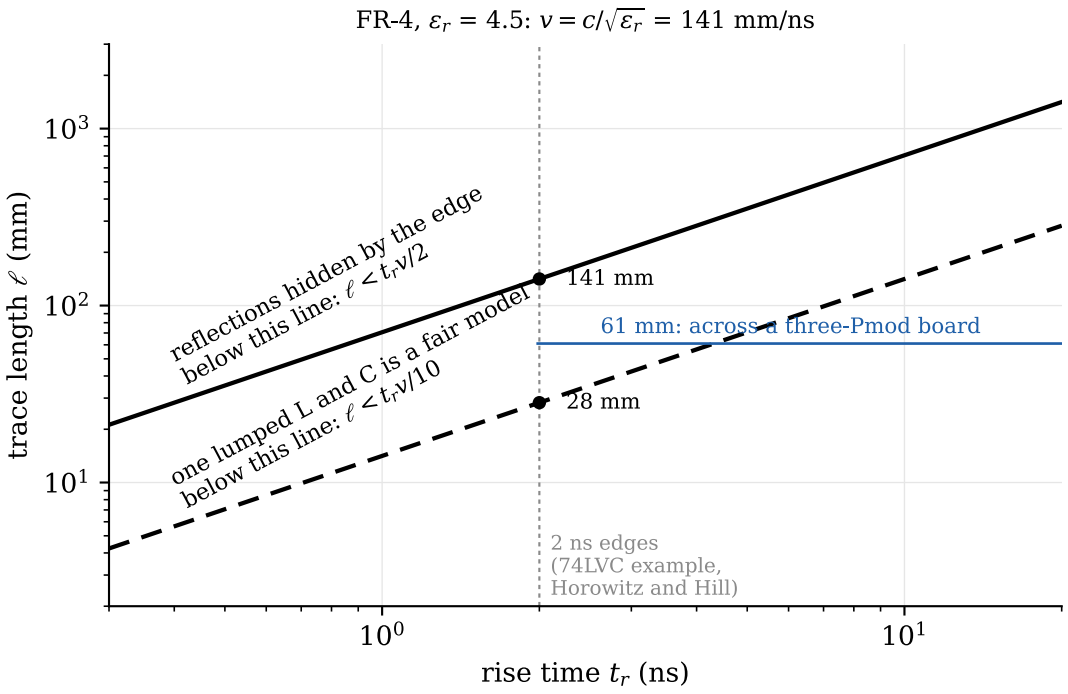


Figure 11.3: Trace length limits against rise time

The two rules turned into lengths of trace. For 2 ns edges, as in the 74LVC example above, a trace up to 28 mm long is a lumped element by the $t_r/10$ rule, and its reflections stay hidden up to 141 mm. The blue line marks 61 mm, about the width of a board that spans all three Pmods.

Tiny Tapeout's GPIO page rates the SKY130 pads at 33 MHz for outputs and 66 MHz for inputs, and says the pads have a slow slew rate, but it gives no rise time. An edge's rise time sets its bandwidth, about $0.35/t_r$ (Hall and Heck, eq. 8-8), so a 2 ns edge has content up to about 175 MHz even when the signal toggles at a few MHz. On a Pmod, whose traces are a few centimetres long, every trace counts as short by the reflection rule for edges of a nanosecond or two. A cable can be long compared with the edge, though, whether it's a wire to a USB-UART dongle or a ribbon to another board. The standard cure for a logic output driving a cable is a series resistor at the driver, of about the line's impedance, which Horowitz and Hill call "series termination" (p. 859). In Lab 04 you'll protect each UART line with a buffer and a series resistor after it. On the line out to the dongle, that resistor sits at the driving end of the cable, and it slows the edges as well, which helps here.

Check yourself

1. Why do the Pmod connectors go down before anything else, and why lock them?

   Answer: A1, in Answers at the end.

2. On the SKY130 breakout, C25 is a 100 nF capacitor on `VDD_IO`, about 18.5 mm from the chip. Suppose the chip had to draw its current pulses from C25, and the path there and back added 3.6 cm of conductor. Using Horowitz and Hill's 5 nH/cm and their 40 mA in 2 ns, how much voltage would that path's inductance develop?

   Answer: A2, in Answers at the end.

3. With 1 ns edges in FR-4 ($\varepsilon_r = 4.5$), up to what length can a trace be treated as one lumped element, and up to what length are its reflections hidden by the edge?

   Answer: A3, in Answers at the end.

4. On a 2-layer board whose bottom layer is a ground pour, you route one signal along the bottom for 20 mm. What does that do to the signals on the top layer that cross it?

   Answer: A4, in Answers at the end.

Next: Pins, Pmods and planning a pinout.

Answers

A1. Check yourself

The board they plug into fixes their positions. They have to be 22.86 mm apart, with inputs on the left, bidirectional in the middle and outputs on the right, so that they line up with the sockets on the demo board and the DIP dock. Every other part is placed around them, and locking them stops a later drag from moving one off that spacing.

A2. Check yourself

3.6 cm × 5 nH/cm is 18 nH, and 40 mA in 2 ns is 20 mA/ns, so $V = L\,dI/dt$ = 18 nH × 20 mA/ns = 0.36 V. That's why the breakout puts nine of its 100 nF capacitors within 2.6 mm of the chip's pins. C25 sits beside U3, the EEPROM footprint, 1.9 mm from U3's supply pin.

A3. Check yourself

At 141 mm/ns, $t_r/10$ is 0.1 ns, or about 14 mm, and $t_r/2$ is 0.5 ns, or about 71 mm.

A4. Check yourself

The bottom track cuts a 20 mm slot in the pour. A signal crossing the slot on top can't have its return current directly underneath, so the return goes around the end of the slot, the loop gets bigger, and the crossing adds series inductance that grows with the detour. Keeping bottom-layer runs short keeps the detours short.

12. Pins, Pmods and planning a pinout

This page answers: *which of a Tiny Tapeout project's pins can a board of yours reach, through which connector pin, and what else on the demo board is already connected to each one?*

A Pmod is the simplest board you can make for a Tiny Tapeout chip. It plugs into headers on the demo board and never touches the chip's breakout. In Lab 03 you'll draw one for a single project, which by default is the team's SimProc, and this page uses SimProc as its example. To draw it, you need to know which socket pin each of the project's pins comes out on, what the demo board already has connected to each of them, and how the project's choice of pins makes the board easier or harder to design. A team can still change its choice of pins until its chip is taped out, so this page ends with that.

A project's pins

Tiny Tapeout's GPIO page lists 26 pins for each project: the clock `clk`, the active-low reset `rst_n`, eight inputs `ui_in[7:0]`, eight outputs `uo_out[7:0]` and eight bidirectional pins `uio[7:0]`. The project sets the direction of each `uio` pin with its `uio_oe` output, 1 for output and 0 for input. The demo board's RP2350 microcontroller usually supplies the clock and drives the reset. The board's three Pmod sockets carry only the other 24 pins, so those are the ones a Pmod can reach. Their electrical limits are on Protecting the chip's I/O.

The three I/O Pmods

The demo board's README says the three Pmod headers along its bottom edge "provide access to all the project I/O in a logical fashion, with inputs, bidirectional pins and outputs available on their own distinct headers", and that "extension boards will often need to span at least two, and sometimes three, distinct headers." Tiny Tapeout's guide to the ETR demo board (ETR is the name printed on the board) adds that the headers "follow the digilent PMOD spec and spacing".

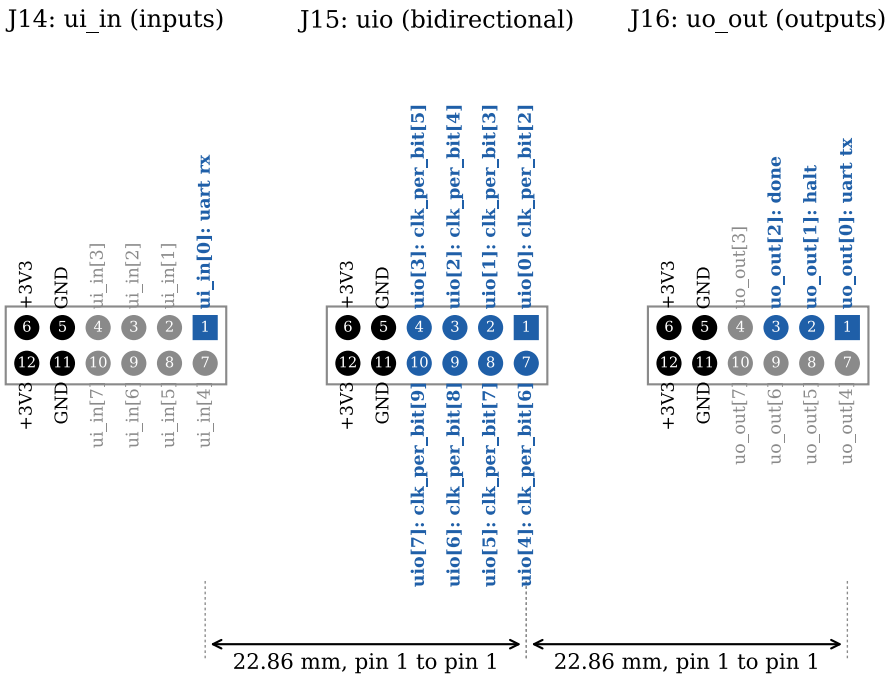


Figure 12.1: The demo board's three I/O Pmod sockets, with SimProc's pins

The demo board's sockets J14, J15 and J16, seen from above, drawn from the board file (`TinyTapeout/tt-demo-pcb` at bbf3dd1) by `scripts/figures/pmod_pinout.py`. Each pad is labelled with the net that the board file gives it, and pin 1 is drawn square. SimProc's pins are in blue, labelled with what each one does, from its `info.yaml` and RTL. Each `uio` pin sets one bit of SimProc's `clk_per_bit`.

Each socket has twelve pins in two rows of six. Pins 1 to 4 carry bits 0 to 3 of the socket's bus, pins 7 to 10 carry bits 4 to 7, pins 5 and 11 are ground, and pins 6 and 12 are the demo board's +3V3. A Pmod takes its power from pins 5, 6, 11 and 12, at the same 3.3 V that the chip's pads run from. Pin 1 of each socket is 22.86 mm from pin 1 of the next. That's the spacing Tiny Tapeout's Pmod library asks for, and the one Placement and routing uses for a Pmod's connectors.

On your own board, the matching part comes from that Pmod library. Its footprint, `PinHeader_2x06_P2.54mm_PMODPeriph2B`, is a through-hole angled header with "digilent numbering". Its symbol, `PMOD_PERIPH_2x6`, names the eight signal pins `IO1` to `IO8`, with `IO1` to `IO4` on pins 1 to 4 and `IO5` to `IO8` on pins 7 to 10, and it has `GND` on pins 5 and 11 and `VCC` on 6 and 12. The library's README asks for the input connector on the left, the bidirectional one in the middle and the output one on the right, looking at the board from the top, which is the order of the demo board's sockets in the figure.

What else is on each pin

Every pin your Pmod reaches is connected to the chip, through the connector the breakout plugs into, and also to other parts on the demo board. Counting only the parts that are fitted, the board file connects them like this:

- **`ui_in[0..7]`** go to the RP2350 (GPIO17 to GPIO24) and to an 8-way DIP switch. Each switch connects its line to +3V3 through a 1 kΩ resistor, so a switch that's on pulls its input high.
- **`uo_out[0..7]`** go to the RP2350 (GPIO33 to GPIO40), and each one drives a segment of the 7-segment display through a bridged solder jumper (JP2 to JP9) and a 510 Ω resistor. `uo_out[0]` to `uo_out[6]` drive segments A to G, and `uo_out[7]` drives the decimal point.
- **`uio[0..7]`** go to the RP2350 (GPIO25 to GPIO32) and to nothing else that's fitted.

On an input, then, your Pmod is one of up to three drivers. The demo board's firmware has a mode, `ASIC_RP_CONTROL`, in which the RP2350 drives the `ui_in` pins, and its example configuration starts in that mode. A DIP switch that's on drives its pin as well. If the browser app in the ETR guide reports the shuttle as "unknown", the guide says the usual cause is "driving the ui_in pins through the INPUT Pmod port". On an output, your Pmod is a second load, because the display segment and its 510 Ω resistor are already there. Protecting the chip's I/O and Supplies, back-powering and ESD deal with both in detail.

The RP2350′s UARTs and the recommended pinouts

Tiny Tapeout's pinouts page says "TT advocates to use common pinouts wherever possible", so that boards made for one project work for others. For a serial console it recommends one of two pairs, "ui_in[3] - RX" with "uo_out[4] - TX", or "ui_in[1] - RX" with "uo_out[0] - TX", and says you can reach either "via the builtin RP2040 on the demo board". The current demo board has an RP2350 instead, and its README's pin table shows which of the RP2350′s pins have a hardware UART:

Project pin	RP2350 pin	UART function in the README's table
`ui_in[0]`	GPIO17	none
`ui_in[1]`	GPIO18	none
`ui_in[3]`	GPIO20	UART1.tx
`uo_out[0]`	GPIO33	UART0.rx
`uo_out[4]`	GPIO37	UART1.rx

The pinouts page was written for the RP2040 board, and on the RP2350 board only its first pair has a hardware UART in both directions. With that pair, UART1 transmits into the project's RX on `ui_in[3]` and receives the project's TX on `uo_out[4]`. With the second pair, UART0 can receive the project's TX on `uo_out[0]`, but `ui_in[1]` has no UART function, so the RP2350 has no hardware UART to transmit to the project with.

SimProc's pins

SimProc's `info.yaml` and its top-level RTL (`uoftasic/simproc` at 1d97772) give this pinout:

Pin	Use
`ui[0]`	UART receive
`uo[0]`	UART transmit
`uo[1]`	halt flag
`uo[2]`	done flag
`uio[0..7]`	inputs (`uio_oe` is 0) that set the UART's bit period: `clk_per_bit = {uio_in[7:0], 2'b00}`

The UART's bit period is `clk_per_bit` cycles of the project's clock. SimProc's `info.yaml` gives that clock as 0 Hz, which means "not applicable", so the bit period in seconds depends on the clock that the demo board supplies. Its documentation says to "Send commands using the UART RX pin (using a USB to UART conveter)" and to "Set UIO7-UIO0 to set the BAUD rate of the internal UART module."

On the demo board, SimProc's transmit pin, `uo_out[0]`, lands on GPIO33, which the README's table lists as a UART0 receiver. Its receive pin, `ui_in[0]`, lands on GPIO17, which has no UART function in the table. So to talk to SimProc from a computer without going through the RP2350, a Pmod needs a header for a 3.3 V USB-UART dongle on `ui[0]` and `uo[0]`, which is how SimProc's documentation says to send it commands, and eight switches with pull-down resistors on `uio[0..7]` to set the baud rate. Those pins are on all three sockets, so the Pmod spans all three, the "sometimes three" case in the demo board's README. It needs no LEDs for the halt and done flags on `uo_out[1]` and `uo_out[2]`, because those pins already light segments B and C of the display, and an LED would be a second load on each one.

Planning a pinout before tapeout

While a design is still RTL, moving a signal to another pin costs nothing. Once the chip is made, its pins can't move, and they decide what board or firmware the chip needs. Had SimProc used the first recommended pair, `ui_in[3]` and `uo_out[4]`, for its UART, the RP2350's UART1 could talk to it in both directions. Four things are worth doing before a team submits a design:

- Check each pin against the demo board's current RP2 pin table, not only against the pinouts page, which still describes the RP2040 board.
- Use a recommended pinout when one fits, so that existing Pmods work with your project. The pinouts page gives them for UART, SPI, I2C, QSPI flash and PSRAM, VGA, audio and game controllers.
- Keep a peripheral's signals within one Pmod row when you can. The page's pinouts for SPI, for UART with optional hardware flow control, and for I2C each use one row, "preferably the upper row".
- Say in `info.yaml` what every used pin does, so that whoever draws the board doesn't have to read the RTL to find out.

Check yourself

1. Which pins of a Pmod socket carry power, and at what voltage?

   Answer: A1, in Answers at the end.

2. On the SimProc Pmod, which socket and pin does the dongle's TX line connect to, and which one does its RX line connect to?

   Answer: A2, in Answers at the end.

3. You want an LED on your Pmod that shows `uo_out[1]`. What else does that pin already drive?

   Answer: A3, in Answers at the end.

4. Why can the RP2350 receive what SimProc transmits with a hardware UART, but not transmit to it the same way?

   Answer: A4, in Answers at the end.

Next: Parts you can solder by hand.

Answers

A1. Check yourself

Pins 5 and 11 are ground, and pins 6 and 12 are the demo board's +3V3, so a Pmod's parts run from 3.3 V.

A2. Check yourself

The dongle's TX drives SimProc's receive pin, `ui_in[0]`, which is pin 1 of J14, the input socket. The dongle's RX listens to SimProc's transmit pin, `uo_out[0]`, which is pin 1 of J16, the output socket.

A3. Check yourself

Segment B of the demo board's display, through the solder jumper JP3 and a 510 Ω resistor, and an input of the RP2350. The pad is rated for 4 mA, and your LED and the segment would share that 4 mA.

A4. Check yourself

SimProc transmits on `uo_out[0]`, which is GPIO33, listed as UART0.rx. It receives on `ui_in[0]`, which is GPIO17, and the README's table lists no UART function for that pin.

13. Parts you can solder by hand

This page answers: *which parts can you solder to a board yourself with an iron, and how do you choose footprints so that the board you order is one you can put together?*

PCB101 doesn't ask you to solder anything, and it doesn't require you to order your board either. If your team does order the board you draw in Lab 03, though, someone will have to assemble it, quite possibly you, and the footprints you choose in the lab decide how hard that job is.

Sizes and their codes

Resistors, capacitors and other two-terminal surface-mount parts come in standard sizes named by a four-digit code, which gives the length and width in units of 0.010 inch. Horowitz and Hill's example is the "0603" package, 0.06 × 0.03 inch, or 1.5 × 0.75 mm (*The Art of Electronics*, 3rd ed., p. 1104). KiCad's footprint names add the metric code, which counts in tenths of a millimetre, so in `R_0603_1608Metric` the same 0603 part is called 1608, for 1.6 × 0.8 mm.

Horowitz and Hill favour 0603, "or the larger 0805 package, for general prototyping of surface-mount circuits". They say the smaller sizes, 0402 and below, "are a major pain – you basically have to work under a microscope (and don't sneeze)". We follow their advice in this course, so on a board you'll assemble yourself, use 0603 or 0805 for resistors and capacitors.

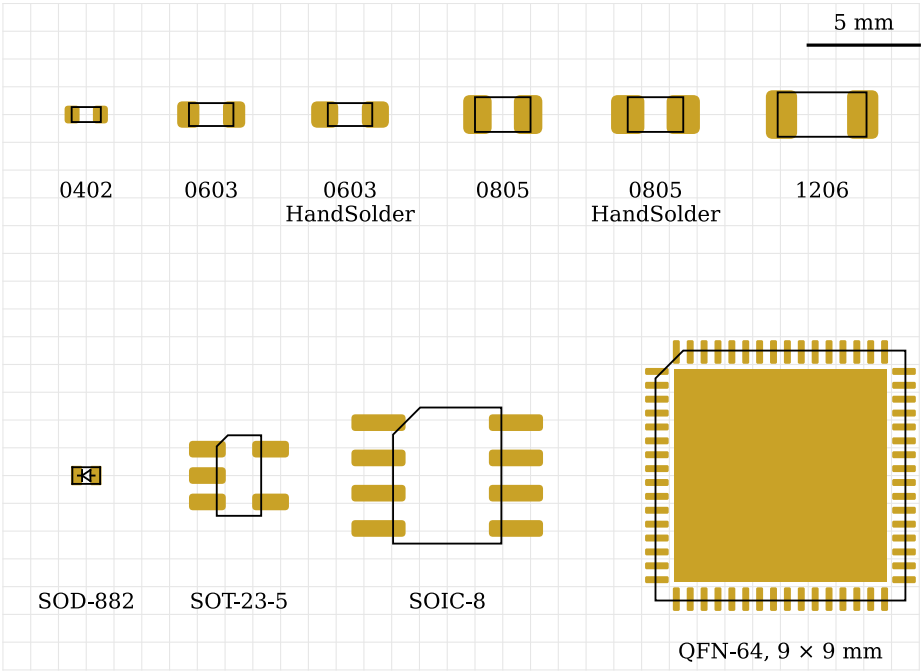


Figure 13.1: KiCad footprints for common small parts, drawn to one scale

Footprints from KiCad 10.0.5's own libraries, drawn to one scale on a 1 mm grid by `scripts/figures/footprint_sizes.py`. The copper pads are gold, and each part's body is drawn as the footprint's fabrication layer draws it. The top row is resistors, from left to right 0402, 0603 and its hand-soldering version, 0805 and its hand-soldering version, and 1206. The bottom row is SOD-882, a two-pin diode package; SOT-23-5; SOIC-8; and the 9 × 9 mm QFN-64 that Tiny Tapeout's SKY130 chip comes in.

KiCad's hand-soldering footprints

KiCad's resistor and capacitor libraries have a second footprint for every size, with `_HandSolder` at the end of its name. Its description reads "IPC-7351 nominal with elongated pad for handsoldering". It's for the same part, with longer pads that leave room for the iron's tip and for a fillet of solder that you can see.

Footprint	Pad (length × width)	Pad centres from the middle
`R_0603_1608Metric`	0.8 × 0.95 mm	±0.825 mm
`R_0603_1608Metric_Pad0.98x0.95mm_HandSolder`	1.4 × 0.95 mm	±0.9125 mm
`R_0805_2012Metric`	1.025 × 1.4 mm	±0.9125 mm
`R_0805_2012Metric_Pad1.20x1.40mm_HandSolder`	1.2 × 1.4 mm	±1.0 mm

Each pad ends half its length beyond its centre, so the 0603 hand-soldering pad reaches 1.4 mm from the middle of the part instead of 1.225 mm, and the 0805 one reaches 1.6 mm instead of 1.425 mm. The inner ends of the pads stay where they were, so all of the extra length is on the outside. The capacitor footprints are named the same way (`C_0603_1608Metric_Pad1.08x0.95mm_HandSolder`, for example), and there are hand-soldering versions of some three- and five-pin packages too, such as `SOT-23-5_HandSoldering`.

Part sizes on Tiny Tapeout's boards

Many of the parts on Tiny Tapeout's boards are smaller than this page recommends. On the SKY130 breakout, every 100 nF capacitor is 0402, while the 1 µF capacitors are 0603 and the 10 µF ones 0805. The DIP dock uses 0402 for 36 of its 37 resistors (the 37th, R15, is a 0 Ω link in an 0805 footprint, and it isn't fitted) and for its 100 nF capacitors, 0603 for its 1 µF capacitors and its LED, and SOT-353 (SC-70-5) for its two logic gates. Tiny Tapeout can use parts that small because a machine assembles its boards. Its archive of production files has the breakout's bill of materials, prepared for JLCPCB's assembly service.

On the breakout, three of the 0 Ω links are 0805: R13, R14 and R15, one on each of the chip's three supplies. The board's other fitted 0 Ω links, R2, R4, R5 and R8, are 0603 or 0402. A note on the schematic says that each supply pin "may be monitored or fed through these headers by popping off the relevant resistor (0805s for easier manip)", so these three links were sized for the person who takes them off by hand. (Measuring current comes back to them.)

Packages with pads you can't reach

Whether you can solder a part depends as much on whether the iron can reach the joint as on the part's size.

- **SOT-23-5**, the package of the breakout's AP2112K regulator, has its leads out beside the body, on 1.325 × 0.6 mm pads at 0.95 mm pitch. You can see and reach every joint.
- **SOIC-8** also has its leads beside the body, on a 1.27 mm pitch.
- **SOD-882**, the package of 22 of the TVS diodes on the demo board, has a body of 1.0 × 0.6 mm and two pads of 0.4 × 0.7 mm, 0.7 mm apart between centres. Most of each pad is under the body, and only 0.05 mm of it reaches past the end. The whole part is about the size of an 0402, the size at which Horowitz and Hill say you have to work under a microscope.
- **QFN-64**, 9 × 9 mm with a 0.5 mm pitch, has pads 0.25 mm wide. Each one reaches only 0.375 mm beyond the edge of the body, and the 7.65 × 7.65 mm exposed pad in the middle is entirely underneath the part, where no iron can reach it. The Tiny Tapeout chip comes already mounted on its breakout, so although you'll draw a footprint for a QFN in Lab 02, you won't have to solder one.

When you add an ESD array to your Pmod's dongle header and a buffer to its UART lines in Lab 04, choose parts in packages with their leads beside the body, like SOT-23-5, SOT-23-6 or SOIC-8, and not in a package like SOD-882.

Through-hole parts

The Pmod connectors, like the other through-hole parts on your board, will be the easiest joints to solder. Tiny Tapeout's Pmod footprint, `PinHeader_2x06_P2.54mm_PMODPeriph2B`, is for a through-hole angled

header, with 1.7 mm pads and 1.0 mm holes on a 2.54 mm pitch. Its joints are large and easy to see, and the pins hold the header in place while you solder them.

Fabricator assembly and safety

If you'd rather not solder, fabricators also assemble boards, as JLCPCB does for Tiny Tapeout's. Assembly costs more than bare boards, and the fabricator needs your board's bill of materials and position file, which Fabrication outputs describes. Reviewing and ordering a board covers ordering, and what assembly costs at JLCPCB. The course doesn't teach soldering or how to do it safely, so if you do solder, do it in a room that's set up for it and follow that room's rules.

Check yourself

1. You need a 100 nF capacitor on a Pmod that you'll assemble yourself. Which KiCad footprint would you pick?

   Answer: A1, in Answers at the end.

2. Why are R13, R14 and R15 on the SKY130 breakout 0805, when its other fitted 0 Ω links are 0603 or 0402?

   Answer: A2, in Answers at the end.

3. Why can't you solder a QFN's exposed pad with an iron, and why does that matter less for the Tiny Tapeout chip than for a QFN on your own board?

   Answer: A3, in Answers at the end.

Next on the course path: Lab 03: A Pmod for one project's pins.

Answers

A1. Check yourself

An 0603 or 0805 capacitor footprint, preferably the hand-soldering version, such as `C_0603_1608Metric_Pad1.08x0.95mm_HandSolder`. Avoid 0402, the size Tiny Tapeout's boards use for 100 nF, because it's one of the sizes that Horowitz and Hill call "a major pain" to work with.

A2. Check yourself

They're the links a person is expected to remove by hand, to measure or feed one of the chip's supplies, and the schematic says they're 0805 "for easier manip". The rest of the board is assembled by machine.

A3. Check yourself

The exposed pad lies entirely under the body of the part, so an iron can't reach it. The Tiny Tapeout chip comes already mounted on its breakout, so you never solder it. A QFN on your own board would have to be soldered some other way, which this course doesn't cover, or assembled by a fabricator.

14. Lab 03: A Pmod for one project's pins

In this lab you start a board repository of your own and draw your first board in it. The board is a 2-layer Pmod that plugs into the three I/O sockets on Tiny Tapeout's demo board and connects one project's pins to the parts that project needs. This page uses the team's SimProc, which is also the project the check uses by default. SimProc needs a header for a 3.3 V USB-UART dongle on `ui[0]` and `uo[0]`, and eight switches on `uio[0..7]` that set its baud rate. You'll draw the schematic, place the three connectors at the spacing of Tiny Tapeout's sockets, place and route the rest of the board, and then check its connections against SimProc's pin list. Labs 04 to 06 continue with this board, in the same repository. The capstone does too if you choose its Brief D; Brief A starts a new board.

The lab's files are in `labs/lab-03-a-pmod-for-one-projects-pins/` in the course repo.

Before you start

- Finish Lab 01, Part 1, so that KiCad 10.0 is installed, the course repo is cloned (the commands below assume `~/pcb101`; use your own path if it's elsewhere) and Docker can run the checks.
- Read the pages on row 5 of the course path: Layers and fabricator rules, Placement and routing, Pins, Pmods and planning a pinout and Parts you can solder by hand.
- Have git and a GitHub account working, as you did in IC102.

The check scripts are bash scripts, so on Windows you run them in WSL. If you don't have bash, the lab's README has a plain `docker run` command that does the same; we have run it on Linux but not on Windows.

The lab takes about 4 hours: roughly 15 minutes for Part 1, an hour for the schematic, half an hour to place the connectors, an hour and 45 minutes to place and route the rest, and half an hour for Part 5. Those times are our estimates for a first board. We did time the check itself, which took 20 seconds in our run once Docker had the image.

Part 1: Start your board repository

From this lab on, your board lives in a repository of your own, and the check you run on it stays in the course repo.

1. Copy the board template out of the course repo and make it a git repository:

   ```
   cp -r ~/pcb101/labs/lab-03-a-pmod-for-one-projects-pins/template ~/simproc-pmod
   cd ~/simproc-pmod
   git init -b main
   git add .
   git commit -m "Start from the PCB101 board template"
   ```

   The template is a KiCad 10 project named `pmod`. Its Board Setup already holds JLCPCB's published limits for a 2-layer board (0.10 mm tracks and gaps, 0.15 mm holes, 0.25 mm vias, and silkscreen text at least 1.0 mm high and 0.15 mm thick), and a custom rule in `pmod.kicad_dru` adds their 0.18 mm minimum annular ring. By default the tracks you draw are 0.25 mm wide, with 0.2 mm clearance. `lib/` holds Tiny Tapeout's Pmod symbol and footprint (Apache 2.0), and the project's library tables point at it, so the project opens the same way on any computer.

2. Make a public repository called `simproc-pmod` in your own GitHub account and push to it, the way you did in IC102. With the GitHub CLI, that's `gh repo create simproc-pmod --public --source . --push`.

3. Before you run the check on the empty project, write down which of its parts you expect to fail. Then run it from your repository:

```
~/pcb101/labs/lab-03-a-pmod-for-one-projects-pins/check.sh
```

You should see the board check, then the Pmod check, both failing:

```
                errors  warnings  excluded
  ERC                0         0         0
  DRC                1         0         0
  unconnected        0         0         0
  parity             0         0         0

  DRC errors (1):
    invalid_outline         Board has malformed outline (no edges found on Edge.Cuts
layer)  [PCB]

FAIL  drc.error: 1 (want 0)
...
  FAIL  connectors   the schematic has 0 PMOD_PERIPH_2x6 connectors; this project
needs 3, one per socket (input, bidir, output)
  FAIL  pins         not checked until the connectors pass

FAIL  pmod.kicad_pro: board check FAIL; 0 of 2 Pmod checks ok
```

ERC finds nothing to object to in an empty schematic, so a clean ERC row on its own doesn't tell you that a board does anything. The Pmod check reads SimProc's pin list, `labs/common/pins/simproc.json`, and it keeps failing until your board connects each pin in that list the way the list says.

KiCad and `kicad-cli` write a few files of their own next to the project. `kicad-cli` writes a `.kicad_prl` of local settings every time it runs on the project, which it does in every run of the check, and saving in KiCad adds a `.history` folder of automatic snapshots. `git status` lists them as untracked. Leave them out of your commits for now by committing with `git commit -am`, which commits changes to the files git already tracks and adds no new ones. Lab 05 adds the `.gitignore` that keeps them out of git.

Part 2: Draw the schematic

What does SimProc need from a board, pin by pin?

1. Open `pmod.kicad_pro` in KiCad, then the Schematic Editor. Place three Pmod connectors with Place > Place Symbols (A): type `PMOD_PERIPH` in the filter, choose `PMOD_PERIPH_2x6` from `TinyTapeout_PMOD`, and place it three times, as J1, J2 and J3.

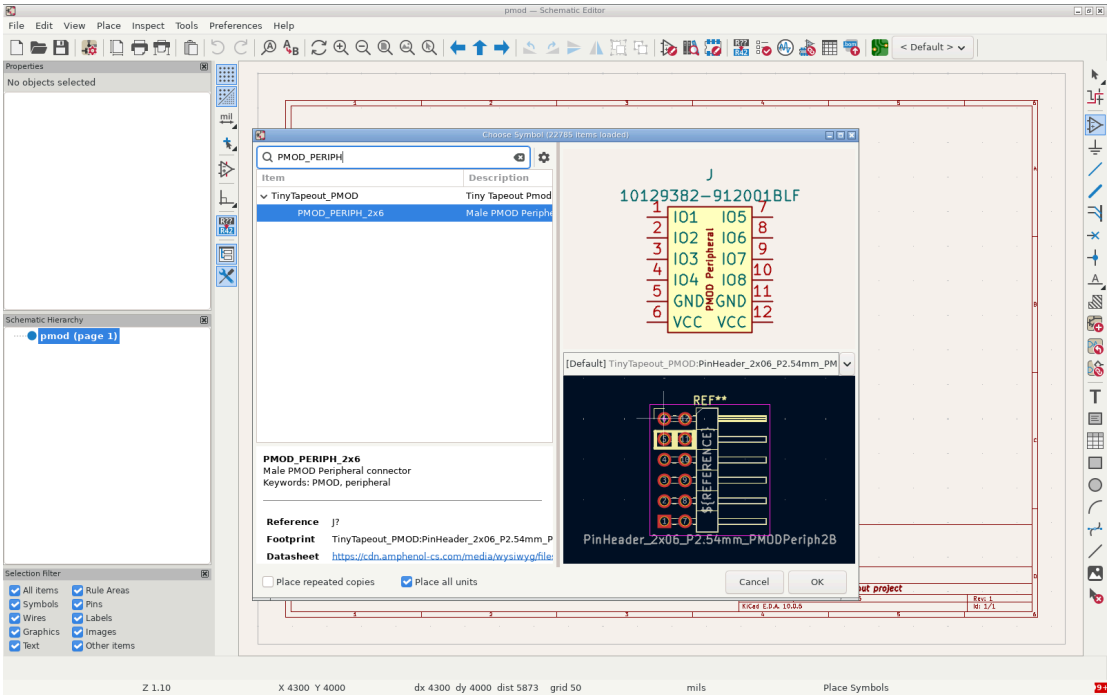


Figure 14.1: The Choose Symbol dialog filtered to PMOD_PERIPH

The Choose Symbol dialog in KiCad 10.0.5, opened with A in the template's schematic and filtered to `PMOD_PERIPH`. The symbol already names its footprint, `TinyTapeout_PMOD:PinHeader_2x06_P2.54mm_PMODPeriph2B`.

J1 will go in the demo board's input socket, J2 in the bidirectional one and J3 in the output one. The symbol's pins are numbered the same way as the sockets' pins. `IO1` to `IO4`, on pins 1 to 4, carry bits 0 to 3 of the socket's bus, and `IO5` to `IO8`, on pins 7 to 10, carry bits 4 to 7. Pins 5 and 11 are GND, and pins 6 and 12 are the demo board's 3.3 V.

2. Connect the pins SimProc uses with net labels (Place > Place Net Labels, L). Label J1 pin 1 (`ui[0]`) as `UART_RX`, J3 pin 1 (`uo[0]`) as `UART_TX`, and J2's eight IO pins as `UIO0` to `UIO7`. Put a `+3V3` power symbol on pins 6 and 12 and a `GND` symbol on pins 5 and 11 of all three connectors. Give each of the 14 IO pins left over a no-connect flag (Place > Place No Connect Flags, Q), which tells ERC that you left it open on purpose.

3. Add the header for the dongle, `Connector_Generic:Conn_01x03`, with the footprint `Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical`. (Double-click a symbol to set its Footprint field.) The check expects the header's pin 1 on GND, pin 2 wired to the dongle's TXD and pin 3 to the dongle's RXD, so pin 2 goes on `UART_RX` and pin 3 on `UART_TX`. Leave the dongle's VCC wire unconnected, because the board gets its power from the demo board.

   Why does SimProc's receive line go to the dongle's *transmit* pin?

   Answer: A1, in Answers at the end.

4. Add the switches and their pull-downs. Use `Switch:SW_DIP_x08`, with the footprint `Button_Switch_THT:SW_DIP_SPSTx08_Slide_9.78x22.5mm_W7.62mm_P2.54mm`. Switch k sits between pin k and pin $17 - k$, so connect pins 1 to 8 to `UIO0` to `UIO7` and pins 9 to 16 to `+3V3`. Then give every `UIO` line a 10 kΩ resistor to GND. You can use eight `Device:R` with the hand-soldering footprint `Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder`, or one `Device:R_Network08` (footprint `Resistor_THT:R_Array_SIP9`), a bussed network in which all eight resistors share pin 1. That shared pin goes to GND. The check accepts either.

5. Before you add anything else, predict what ERC will say about the power pins. Then run it (Inspect > Electrical Rules Checker, or the check).

   You should see two errors of the same kind. The check prints them like this for our schematic, and the `#PWR` numbers in yours may differ:

```
  ERC errors (2):
    power_pin_not_driven      Input Power pin not driven by any Output Power pins
[Symbol #PWR001 Pin 1 [Power input, Line]]
    power_pin_not_driven      Input Power pin not driven by any Output Power pins
[Symbol #PWR003 Pin 1 [Power input, Line]]
```

Why does ERC think nothing powers the board?

Answer: A2, in Answers at the end.

6. Add one `power:PWR_FLAG` on `+3V3` and one on `GND`, and run ERC again. Commit with `git commit -am "Schematic"`.

In the check's output you should see 0 in all three columns of the ERC row. DRC still reports the `invalid_outline` error from Part 1, because you draw the outline in Part 3. Parity reports a warning for every footprint that isn't on the board yet, six in our version (`missing_footprint J1` to `missing_footprint SW1`). The Pmod check now finds the connectors and asks for them on the board: `FAIL connectors J1, J2, J3 not on the board yet (Update PCB from Schematic, F8)`.

Part 3: Place the connectors

Where exactly do the three connectors have to be?

1. In the PCB Editor, run Tools > Update PCB from Schematic (F8) and click Update PCB. The footprints arrive together, attached to the cursor; click to drop them beside where the board will go.
2. Select J1 and set its Orientation to 90° in the Properties panel, so that its pins point up, off the top of the board. Put it wherever you like; ours is at X 102, Y 56.35.
3. Select J2 and press Shift+P, which opens Position Relative To with Use polar coordinates ticked, so it asks for a Distance and an Angle. Click Select Item… and then J1, and enter 22.86 mm and 0°. That puts J2 22.86 mm to the right of J1, along +X. Give J2 the same 90° orientation. Then do the same for J3, relative to J2.

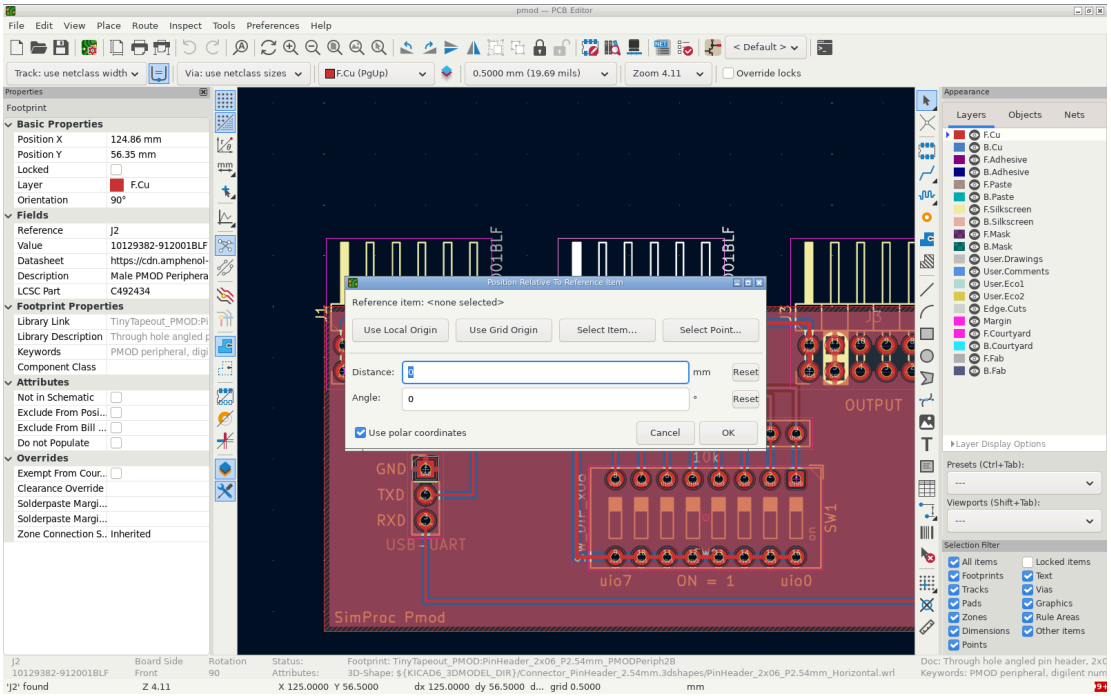


Figure 14.2: The Position Relative To Reference Item dialog over J2

Position Relative To in KiCad 10.0.5, opened with Shift+P on J2 of our finished board. The Properties panel on the left shows J2's position, its 90° orientation and the Locked box you'll tick in step 5.

You should see J2's Position X in the Properties panel exactly 22.86 mm more than J1's, and its Position Y the same. Why 22.86 mm?

Answer: A3, in Answers at the end.

4. Draw the board outline on the Edge.Cuts layer with the rectangle tool. Its top edge has to cross the connectors' plastic bodies, so that the pins stick out past the edge and into the sockets. Ours is 64 × 32 mm, from (100, 50) to (164, 82), which puts the top edge 6.35 mm above the lower row of each connector's pads.
5. Lock the three connectors by ticking Locked in the Properties panel for each one, so that a stray drag can't move them off the spacing.
6. Run the check.

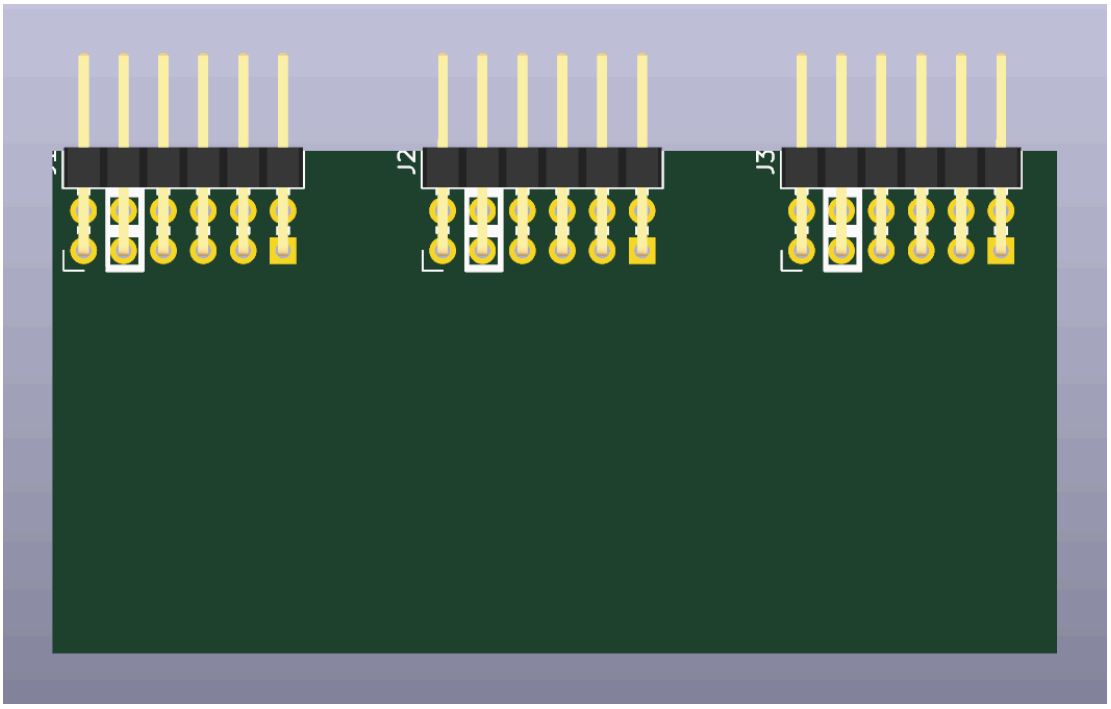


Figure 14.3: The three Pmod connectors on a 64 x 32 mm outline

Three connectors 22.86 mm apart on our outline, rendered by `kicad-cli` 10.0.5 with the parts' 3D models. They're angled pin headers, so their pins stick out past the board's top edge.

You should see the connectors pass, with your own positions in place of ours:

```
  ok     connectors    J1 (102.00, 56.35), J2 (124.86, 56.35), J3 (147.72, 56.35):
22.86 mm apart, in a row, pins past the edge, locked
  ok     sockets       J1 input (ui), J2 bidir (uio), J3 output (uo), left to right
with the pins pointing away from you
```

The board check still fails on unconnected items, and it will until you route the board in Part 4. Ours had 38 unconnected items with every footprint placed and nothing routed.

Part 4: Place the rest and route

Can every connection be made on two layers, with the bottom layer kept as a ground plane?

1. Place the switches, the pull-downs and the header. Put the switches under J2, because all eight `uio` lines come from it, and put the header where a dongle's wires can reach it.
2. Look at J2's pads and at the row of eight switches before you route. Can the eight `uio` lines reach switches 1 to 8 in order without crossing, on one layer?

Answer: A4, in Answers at the end.

3. Route every connection. Use the default 0.25 mm for signals, and 0.4 mm, which is in the track-width list, for `+3V3`.
4. Add a GND zone on B.Cu over the whole board (Place > Draw Filled Zones, net GND), and another on F.Cu if you like, then press B to fill them.
5. Run DRC (Inspect > Design Rules Checker) with Test for parity between PCB and schematic ticked, and fix everything it reports as an error, including every unconnected item. Commit, and push.
6. Run the check. You should see the board check pass:

```
               errors  warnings  excluded
ERC                 0         0         0
DRC                 0        30         0
unconnected         0         0         0
parity              0         0         0

DRC warnings by type: silk_edge_clearance 24, silk_over_copper 6
```

   Where do those 30 warnings come from?

   Answer: A5, in Answers at the end.

Look at your board in KiCad's 3D viewer (View > 3D Viewer, Alt+3). If your KiCad has its 3D models (the Linux Lite AppImage doesn't), you'll see the parts, as in our render below.

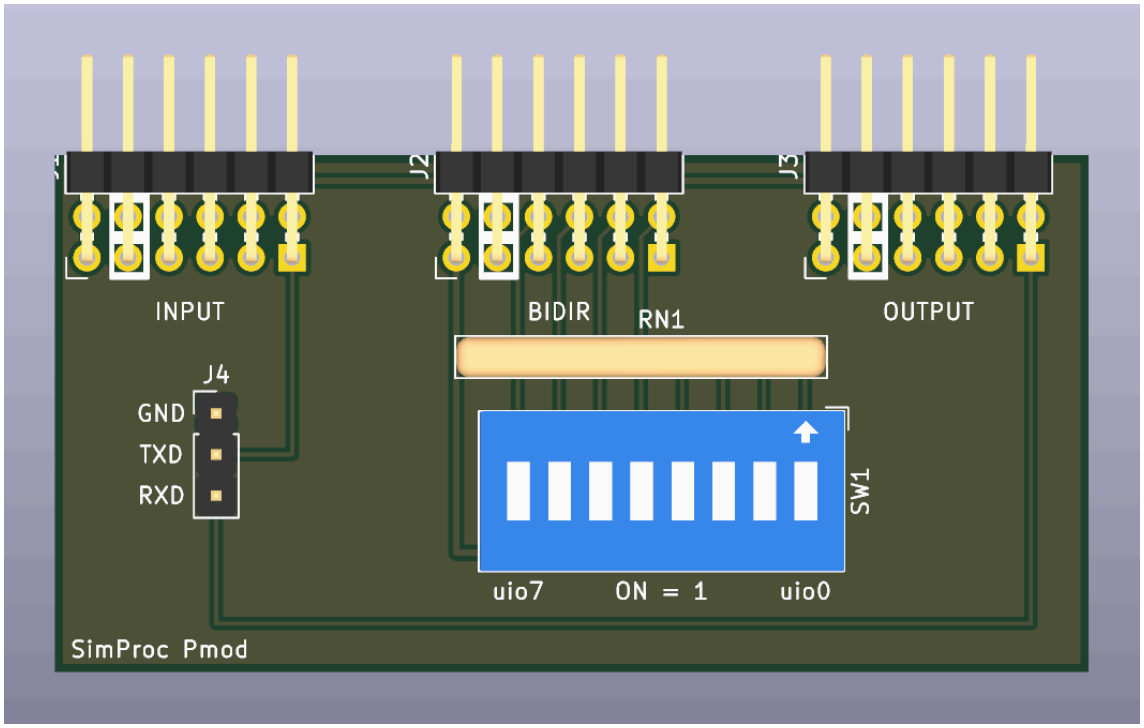


Figure 14.4: The Lab 03 reference Pmod for SimProc, from the top

Our Lab 03 board, rendered by `kicad-cli` 10.0.5 with the parts' 3D models; it's one of many layouts that would work. The `uio` lines go from J2 to the pull-down network RN1 and the switches SW1. Those from J2's upper row run straight down on the top layer, and those from its lower row cross to the right on the bottom layer.

Part 5: Check the connections against SimProc's pin list

Does your board connect SimProc's pins the way its pin list says?

1. Open `simproc.json` and compare it with your schematic, pin by pin.
2. Suppose you'd put `UART_TX` on the header's pin 2 and `UART_RX` on pin 3, and routed the board to match. Write down which parts of the check would catch that: ERC, DRC, parity, or the Pmod check.

   Answer: A6, in Answers at the end.
3. Run the check on your board. You should see every Pmod row `ok`, with your own parts named, and the last line `PASS  pmod.kicad_pro: board check PASS; 14 of 14 Pmod checks ok`.

Finishing

You're done when `~/pcb101/labs/lab-03-a-pmod-for-one-projects-pins/check.sh`, run in your own repository, ends with `PASS  pmod.kicad_pro: board check PASS; 14 of 14 Pmod checks ok`, and the commit it checked is pushed to GitHub.

Questions:

1. The demo board already shows `uo[1]` (halt) and `uo[2]` (done) on its 7-segment display. Why doesn't your Pmod have LEDs for them?

   Answer: A7, in Answers at the end.

Submit: nothing for this lab. Lab 04 continues in the same repository.

Going further (optional)

- Draw a Pmod for another team project. Copy `simproc.json`, write that project's pins into it, and run the check with `--pins your-file.json` after the folder argument.
- Put the "Designed for Tiny Tapeout" logo on your board. The logo is in Tiny Tapeout's full Pmod library; the template copies only the Pmod symbol and footprint from it.

Next on the Course path: read Protecting the chip's I/O, Supplies, back-powering and ESD, Measuring current and Bringing up a chip, then do Lab 04: Protect and plan.

Answers

A1. Part 2: Draw the schematic

Each end names TX and RX from its own point of view. The dongle transmits on its TXD pin and SimProc receives on `ui[0]`, so the dongle's TXD drives `ui[0]`. In the other direction, SimProc transmits on `uo[0]`, which drives the dongle's RXD. Swapping the two is the classic UART mistake, and Part 5 shows which check catches it.

A2. Part 2: Draw the schematic

The Pmod symbol's power pins and the power symbols are all power inputs. Nothing on the schematic is a power output, because the 3.3 V comes in through the connectors from the demo board's regulator, which isn't on your schematic. ERC reports one error per net, here `+3V3` and `GND`. A `PWR_FLAG` on a net tells ERC that power reaches it from outside.

A3. Part 3: Place the connectors

22.86 mm is 0.9 in, which is the spacing of the demo board's three I/O sockets. The README of Tiny Tapeout's Pmod library gives that spacing for a board with more than one Pmod connector, with the input connector on the left, the bidirectional one in the middle and the output one on the right, looking at the board from the top. The check allows 0.02 mm either way, and a part you drag by hand won't reliably land that close.

A4. Part 4: Place the rest and route

They can't, unless they detour around the end of the connector. J2's signal pins are in two rows of four, with bits 0 to 3 in one row and bits 4 to 7 in the other, so bit 4 sits opposite bit 0, bit 5 opposite bit 1, and so on along the connector. Tracks that leave the connector side by side on one layer can't change their order without crossing. To bring the lines out to a single row in order, put one row's four tracks on the top layer and the other row's on the bottom, where they can cross the top-layer tracks. Keep the bottom-layer tracks short, because each one cuts a slot in the ground plane there.

A5. Part 4: Place the rest and route

On our board, all of them come from Tiny Tapeout's Pmod footprint. The footprint's silkscreen draws the six pins that stick out of each connector, so when the board's edge crosses the connector, seven

silkscreen segments and the connector's reference text lie past the edge. That's 24 `silk_edge_clearance` warnings for the three connectors. The silkscreen also outlines pads 5 and 11, the GND pads, and touches their copper, which gives the 6 `silk_over_copper` warnings. KiCad reports both kinds as warnings, and the check lists warnings without failing on them. Your own count can differ if your edge crosses the connectors somewhere else.

A6. Part 5: Check the connections against SimProc's pin list

Only the Pmod check catches it. We made that board, and its board check passed with 0 errors and 0 unconnected items. ERC looks only at the schematic, DRC only at the board, and parity checks that the board matches the schematic, so the swap passes all three. The Pmod check compares the schematic with SimProc's pin list and prints:

```
  FAIL  ui[0]         J1 pin 1 reaches J4 pin 3 (the dongle's RXD), not pin 2 (the
dongle's TXD): are TX and RX swapped?
  FAIL  uo[0]         J3 pin 1 reaches J4 pin 2 (the dongle's TXD), not pin 3 (the
dongle's RXD): are TX and RX swapped?
```

A7. Finishing

Each `uo_out` line already drives a segment of the display through a 510 Ω resistor. An LED on your board would be a second load on the same pin, so the LED and the segment would share the pad's rating of 4 mA, and the LED would only repeat what the segment shows. Pins, Pmods and planning a pinout has the details.

IV 4. Protection and bring-up

15 Protecting the chip's I/O 101
15 What the pins are rated for 101
15 Beyond the rails 101
15 Sizing a series resistor 102
15 Outputs 103
15 Buffers and translators 103
15 Inputs that nobody drives 105
15 Check yourself 105
15 Answers 106

16 Supplies, back-powering and ESD . 107
16 The chip's three supplies 107
16 A regulator and its capacitors 107
16 The order the supplies come up in 108
16 Back-powering 108
16 Contention 109
16 ESD 110
16 Check yourself 110
16 Answers 111

17 Measuring current 113
17 A meter in series, or a sense resistor 113
17 Choosing the resistor 114
17 What you can measure on a Tiny Tapeout kit . 114
17 Reading the voltage 115
17 Check yourself 116
17 Answers 116

18 Bringing up a chip 117
18 Before any power 118
18 The kit alone, with a current limit 118
18 The order of the supplies 119
18 A known firmware state 120
18 Select, clock, reset, test 120
18 Things with their own power 120
18 When a reading is wrong 121
18 Check yourself 121
18 Answers 121

19 Lab 04: Protect and plan 123
19 Before you start 123
19 Part 1: Every line that leaves the board 123
19 Part 2: A buffer on each UART line 124
19 Part 3: ESD on the dongle header 126
19 Part 4: A current-sense point 127
19 Part 5: The bring-up plan 127
19 Finishing 128
19 Going further (optional) 128
19 Answers 128

15. Protecting the chip's I/O

This page answers: *what can the board around a Tiny Tapeout chip do to the chip's digital pins, and what can you put between a pin and the outside world to protect it?*

The chip on a Tiny Tapeout breakout is the only one in the kit, so a mistake on your Pmod that damages a pin is expensive. This page gives the pins' ratings and says what happens when a pin is driven past them. SkyWater's guide for the pads allows 100 µA into a pin that is driven past its supply. The page works out the series resistor that this would take on the UART lines between SimProc and a USB-UART dongle. In the worst case that the datasheets allow, no value of resistor both holds the current to 100 µA and lets the UART work. Then it covers buffers and translators, among them the buffer that Lab 04′s board uses on those lines. Last come pull resistors, which hold an input at a defined level when nothing drives it. Supplies, back-powering and ESD covers what happens when one board is powered and the other isn't, and what static discharge does to a pin. In Lab 04 you'll put a buffer and a resistor on each UART line of your Lab 03 board.

What the pins are rated for

Tiny Tapeout's GPIO page says the chip uses the `sky130_ef_io_gpiov2_pad` macro for its I/O pads, and lists these limits for them:

Parameter	Limit
I/O supply voltage	1.71 V to 5.5 V; the demo board supplies 3.3 V
Drive strength, source or sink	4 mA
Maximum output frequency	33 MHz
Maximum input frequency	66 MHz

Below the table it says, "The input pins are not 5V tolerant." On the demo board the pads run from 3.3 V, so keep every input between 0 V and 3.3 V.

SkyWater's user guide for its I/O cells describes the circuit behind that limit. The guide says its GPIO cells "use the main Vddio/Vssio connections for their ESD current steering diodes", which means that each pad has a diode up to the I/O supply and another down to ground. Its feature list marks two cells, `sky130_fd_io__sio` and `sky130_fd_io__gpio_ovtv2`, as "Hot swap & over-voltage tolerant", and doesn't mark `sky130_fd_io__gpiov2`. The one current limit for a pin that we found in it is among its integration requirements: "It is permissible to externally connect the I/O pin to a voltage above vddio or below vssd_io, provided that the current flowing into/out of the pin is externally limited to 100uA." We found no other rating in it for the current through the diodes, and none for the current that damages a pin or starts latch-up.

This course designs to that 100 µA. Whenever a pin can be taken above the chip's I/O supply or below ground, the current into or out of the pin must stay at or below 100 µA. While the chip is unpowered, any high on a pin takes it above the supply. For larger currents, the next section uses Horowitz and Hill's typical figures.

Beyond the rails

Between ground and its supply voltage, a CMOS input draws almost no current. Horowitz and Hill (*The Art of Electronics*, 3rd ed., p. 795) put the leakage at about 10^{-5} µA. Outside that range the protection diodes conduct. An input more than a diode drop above the supply pushes current through the upper diode into the supply rail, and an input more than a diode drop below ground pulls current out through the lower one. Horowitz and Hill warn that "Momentary currents greater than 20–50 mA through these diodes will damage or destroy the part, in some cases causing what's known as 'SCR latchup'". In latch-up, the current through the input diodes turns on a pair of parasitic transistors, which short the supply to ground, and the chip stays shorted and gets hot until you turn the power off (p. 759). In a footnote on

the same page they add that manufacturers often guarantee no latch-up as long as the input current stays below a maximum "input clamp current", 20 mA for example, which means that "you can use a series input resistor to prevent latchup even with input-voltage overdrive" (p. 759).

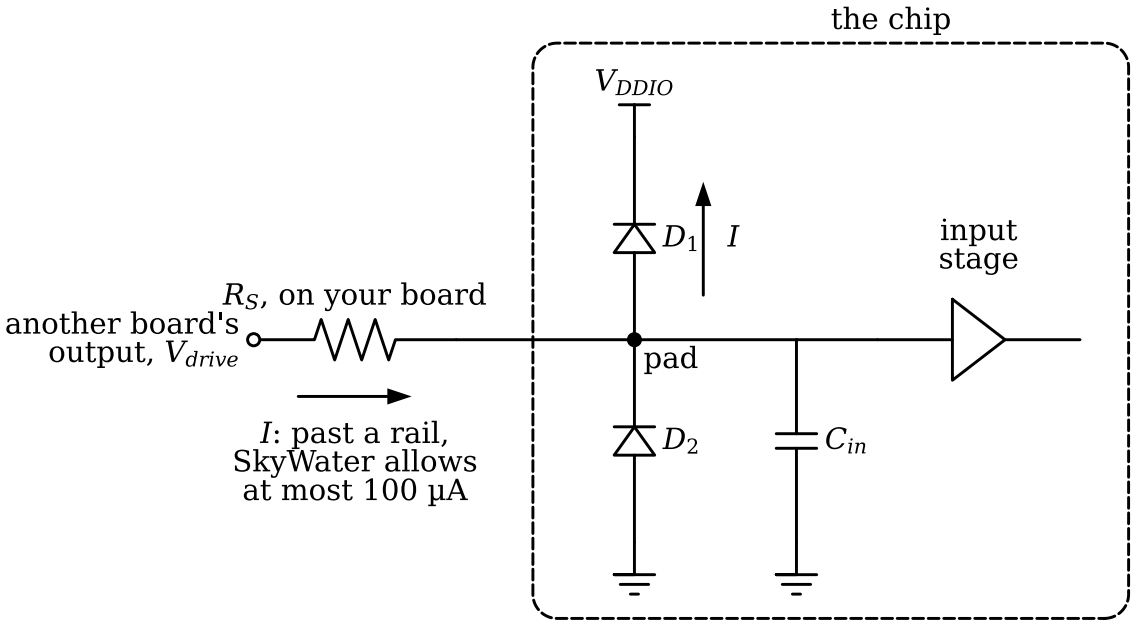


Figure 15.1: A series resistor in front of a chip input with its two clamp diodes

A chip input, as it looks electrically from outside the chip. D_1 and D_2 are the pad's protection diodes, and C_{in} stands for the capacitance of the pin, its trace and the connectors together. When V_{drive} rises more than a diode drop above V_{DDIO}, D_1 conducts, and R_S sets the current I into the pin.

Sizing a series resistor

Suppose a series resistor R_S on your board is all that stands between another board's output, at V_{drive}, and the pad. With the pad past the rail, the current through the resistor and the upper diode is

$$I = \frac{V_{drive} - V_{DDIO} - V_D}{R_S}, \tag{15.1}$$

where V_D is the diode's forward drop. We take it as 0.6 V, the level at which Horowitz and Hill say an unpowered 74HC input clamps (p. 795). For a 3.3 V source such as a USB-UART dongle, the worst case is a chip that isn't powered, with V_{DDIO} near 0 V, because the whole 3.3 V then appears across the resistor and the diode. In the table below, "kit off" means that the demo board, and the chip on it, have no power while the dongle, powered from the laptop, is still connected. The second column gives the current then.

The resistor also has to let the UART work while the kit is on. On the SimProc Pmod, the dongle's TX drives `ui_in[0]`, which SimProc's UART receives on, and SimProc transmits on `uo_out[0]` to the dongle's RX. The demo board's RP2350 is on `ui_in[0]` too, through its GPIO17. Of the three modes that the demo board firmware's example `config.ini` lists, two leave the `ui_in` pins as inputs, `SAFE` and `ASIC_MANUAL_INPUTS`, and SimProc's section of `config.ini` has to select the second (Supplies, back-powering and ESD explains the modes). In both of them the firmware turns on the RP2350's pull-down on every `ui_in` pin, GPIO17 included. The RP2350 datasheet gives that pull-down as 36 to 113 kΩ at 3.3 V. While the dongle's TX is high, R_S and the pull-down form a divider, and with the strongest pull-down the high that reaches the pad is $3.3\ \text{V} \times 36\ \text{k}\Omega/(36\ \text{k}\Omega + R_S)$. Tiny Tapeout sets the `ui_in` pads as CMOS inputs, and SkyWater's guide says that a CMOS input needs $0.7 \times V_{DDIO}$ for a high, which is 2.31 V at 3.3 V. The third column gives the high.

R_S	Kit off: $I = (3.3\ \text{V} - 0.6\ \text{V})/R_S$	Kit on: high at `ui_in[0]` against 36 kΩ
100 Ω	27 mA	3.29 V
330 Ω	8.2 mA	3.27 V
1 kΩ	2.7 mA	3.21 V
4.7 kΩ	0.57 mA	2.92 V
15.4 kΩ	0.18 mA	2.31 V
27 kΩ	0.10 mA	1.89 V

Every row but the last is over SkyWater's 100 µA limit, and the first, 27 mA, is also past Horowitz and Hill's 20 mA example. Holding the current to 100 µA takes at least 2.7 V / 100 µA = 27 kΩ, and at 27 kΩ the high is 1.89 V, below the 2.31 V that the pad needs. A valid high needs R_S of at most 15.4 kΩ, so against the strongest pull-down no value of R_S meets both. The next page follows this current further while the kit is off, through the chip's I/O supply and onto the demo board's whole 3.3 V net.

On the other line, `uo_out[0]`, a series resistor can't meet both conditions either. There the current into the unpowered pad comes from the dongle's RX input, which has a pull-up of its own. The datasheet for the CH340, one of the chips that USB-UART dongles use, gives its pull-up current as up to 200 µA at 3.3 V. It doesn't say at what pin voltage that 200 µA flows, so assume 0 V, which makes the pull-up 3.3 V / 200 µA = 16.5 kΩ. Into an unpowered `uo_out[0]`, with nothing in series, that's (3.3 − 0.6) V / 16.5 kΩ = 164 µA. Holding it to 100 µA would take 10.5 kΩ in series. With the kit on and the chip's output low, though, the RX pin sits at 3.3 V × R_S / (R_S + 16.5 kΩ), and that stays at or below the CH340's 0.8 V input-low limit only for R_S up to 5.28 kΩ, less than the 10.5 kΩ that the current needs. The board you finish in Lab 04 therefore has a buffer on each UART line, with a resistor after it, as Buffers and translators describes.

A series resistor also slows the signal's edges. That applies to the 1 kΩ after each buffer on Lab 04's board as well. Together with the capacitance at the pin it drives, the resistor forms an RC low-pass filter, whose 10–90 % rise time is $2.2RC$ (Hall and Heck, *Advanced Signal Integrity for High-Speed Digital Designs*, eq. 8-8, give 2.195τ). We have no figure for the pad's own capacitance, so suppose the pin, its trace and the connectors add up to 10 pF. Then 1 kΩ gives τ = 10 ns and an edge of 22 ns. For a UART at 115 200 baud, whose bits last 8.7 µs, an edge that long doesn't matter. For an input near the pads' 66 MHz rating, whose half period is 7.6 ns, the edge would last longer than the signal stays high, so a fast line can only have a small series resistor, of tens of ohms. A resistor of that size at the driving end is also the series termination that Placement and routing describes for a logic output driving a cable.

Outputs

Another board's driver or pull-up can push an output pin past its rails through the pin's own protection diodes, as a dongle's RX pull-up does to an unpowered `uo_out[0]`. A load on your board can also need more current than the pin's 4 mA. On the demo board, every `uo_out` line already drives one segment of the 7-segment display through a solder jumper and a 510 Ω resistor. An LED that your Pmod adds on the same line is a second load, and the LED and the segment then share the pin's 4 mA. A `uo_out` pin can also drive the input of something powered separately, such as the RX pin of a USB-UART dongle. When the chip is powered and the dongle isn't, a high output pushes current through the dongle's clamp diode, and that current loads the chip's output and feeds the dongle's supply. On Lab 04's board the chip's output drives the buffer's input instead of the dongle, and the buffer drives the dongle through a resistor that limits that current.

Buffers and translators

A buffer drives its output from its own supply, so it gives the far side that supply's logic levels, which a series resistor can't do. A buffer that is specified for partial power down also passes almost no current while its supply is off. The course's reference board for Lab 04 has a buffer on each UART line for that second reason, although the dongle and the chip both run at 3.3 V. The part is U1, a Texas Instruments SN74LVC2G17, which has two Schmitt-trigger buffers, one for each line, in a SOT-23-6 package. The datasheet's row nearest the kit's 3.3 V is for a 3 V supply, where its inputs switch high somewhere between

1.3 and 2.0 V and low between 0.8 and 1.3 V, so a 3.3 V signal from the dongle or from the chip crosses both thresholds.

Partial power down

TI's datasheet for the part says, "This device is fully specified for partial-power-down applications using Ioff. The Ioff circuitry disables the outputs, preventing damaging current backflow through the device when it is powered down." I_{off} is the most current a pin passes while the part's supply is at 0 V. For the SN74LVC2G17 it's ±10 µA at most, with 5.5 V on an input or an output, and its inputs accept up to 5.5 V. Horowitz and Hill describe the same behaviour for the 74LVC family as a whole. Its inputs accept up to 5.5 V "regardless of its own supply voltage (including when it is unpowered)" (p. 718), and its output stages are built not to load a shared line when the part is unpowered (p. 794).

With the kit off and the dongle on, the dongle's TX drives U1′s unpowered input, and U1′s output passes at most 10 µA on to `ui_in[0]`, a tenth of SkyWater's limit. TI gives that figure with U1′s supply at 0 V. U1 runs from the kit's +3V3, and current from the dongle's TX flows through R4, the 100 kΩ pull-up on U1′s input (Inputs that nobody drives), into that rail and lifts it above 0 V. Even so, the most the dongle's TX can push onto the board is about 43 µA, up to 10 µA into U1′s input and up to 33 µA through R4, so less than 100 µA can reach the chip. On the `uo_out[0]` line, the dongle's RX pull-up pushes current into U1′s unpowered output, again at most 10 µA, and none of it reaches `uo_out[0]`, which is on U1′s input side.

U1′s supply

I_{off} applies only while U1′s own supply is at 0 V, so U1 has to lose its power whenever the chip does. The reference board runs it from the Pmod's +3V3, which is the kit's 3.3 V, and the chip's I/O supply comes from the same net. If U1 ran from the dongle's 3.3 V instead, it would stay on with the kit off and drive the unpowered pad through the resistor after it, with about (3.3 − 0.6) V / 1 kΩ = 2.7 mA, the same current that a 1 kΩ resistor with no buffer would pass. Powered from the kit, U1 also can't drive the chip's input above the chip's own I/O supply while the kit is on.

The resistor after each buffer

With U1 holding the current into an unpowered chip to 10 µA, the resistor after each buffer limits two other currents. On the `ui_in[0]` line, it's the current when another driver on the same pin fights U1. Without the resistor, only the two drivers' output stages would limit that current, and each of U1′s outputs can drive 24 mA at 3.3 V. On the `uo_out[0]` line, it's the current into an unpowered dongle. The Pmod check that you run in Lab 04 accepts a window of values for each resistor, and the reference board uses 1 kΩ for both.

R1 sits between U1′s first output and the chip's `ui_in[0]` pin. The firmware's example `config.ini` starts in `ASIC_RP_CONTROL`, a mode in which the RP2350 drives the `ui_in` pins itself, so until SimProc's section of `config.ini` selects `ASIC_MANUAL_INPUTS`, the RP2350 can drive that pin against U1. The demo board's DIP switch 1 can too, because when it's on, it connects the line to +3V3 through 1 kΩ. With R1 at 1 kΩ, a fight with the RP2350 passes at most 3.3 V / 1 kΩ = 3.3 mA. A fight with the switch passes 3.3 V / 2 kΩ = 1.65 mA, through the switch's own 1 kΩ and R1 in series. The switch still has to stay off, because against R1 it holds a low on the line at 3.3 V × 1 kΩ / 2 kΩ = 1.65 V, and the pad reads a low only up to 0.3 × V_{DDIO} = 0.99 V.

The check's window for R1 is 330 Ω to 8.2 kΩ. At 330 Ω a fight passes 3.3 V / 330 Ω = 10 mA. The RP2350′s absolute maximum ratings give no current for a pin, and its pads' drive strength can be set to 2, 4, 8 or 12 mA, so that 10 mA is a limit we chose rather than a rating. The top of the window comes from an erratum. On the RP2350′s A2 stepping, erratum E9 says that a GPIO input leaks about 120 µA and holds its pad near 2.2 V, and that only a source of 8.2 kΩ or less is sure to pull it low. The firmware leaves GPIO17′s input on, so on an A2 kit R1 can't be bigger than that. The marking on the RP2350′s package includes its stepping, A2, A3 or A4. On an A3 or A4, where E9 is fixed, R1 could go up to 15.4 kΩ against the 36 kΩ pull-down. With R1 at 1 kΩ, the high against that pull-down is 3.21 V.

R2 sits between U1′s second output and the dongle's RX pin, on the `uo_out[0]` line. With the kit on and the dongle unpowered, U1 drives a high into the dongle's clamp diode, and R2 limits that current to (3.3 − 0.6) V / 1 kΩ = 2.7 mA. None of the datasheets we read for five dongle chips, the CH340 among them, gives a limit for the current into a pin, so the 1 kΩ at the low end of the check's window, and the 2.7 mA it allows, are a judgement of ours. The top end is 5.1 kΩ, the E24 value just below the 5.28 kΩ that

the CH340′s pull-up allows if its 200 µA flows at 0 V (worked out above), so that the window ends on a value you can buy.

Changing a signal's level

When the other side runs from a supply other than the chip's 3.3 V, you need a part in between that changes the signal's level. Horowitz and Hill give a set of recommended connections between logic families (pp. 798–800), and these four are the ones you'd use between another board and the chip:

- **A buffer with 5 V-tolerant inputs**, such as U1, powered from your Pmod's 3.3 V. It turns a 5 V signal into a 3.3 V one for the chip's input, because its inputs accept up to 5.5 V.
- **A dual-supply translator**, such as the 74LVC1T45, runs each side from its own supply, anywhere from 1.8 V to 5 V, and has a direction pin that sets which way the signal goes. Use one when the other side is lower than 3.3 V, or when the line goes both ways.
- **An open-drain output with a pull-up to the receiving side's supply.** It's simple, and it can translate a signal up or down across a wide range of voltages. The pull-up resistor is a compromise, because a smaller one gives faster rising edges but draws more current while the line is low.
- **An auto-direction translator**, such as the TXB0101, senses transitions on either side, so it needs no direction pin, but the supply on its B side must be at least the supply on its A side.

The chip's analog pins carry an analog voltage rather than a logic level, so these parts don't apply to them. Tiny Tapeout's analog page gives the path from each analog pad to the project as under 500 Ω and under 5 pF, through an analog switch, with 4 mA at most. Brief A of the capstone, a source board for one analog pin, asks you to work with those numbers.

Inputs that nobody drives

A CMOS input that nothing drives is left floating. Horowitz and Hill warn that an open input can float up to the logic threshold, which puts the gate's output at half the supply "with both MOS output transistors conducting", so the gate draws current and can oscillate (p. 806). So every input your board might leave undriven needs a resistor that holds it at a defined level. The usual case is a switch, which leaves its input undriven while it's open. Horowitz and Hill's circuit for a switch driving a logic input (p. 802) uses a 10 kΩ pull-up or pull-down, and with CMOS either one works. SimProc's `uio[0..7]` pins are all inputs (`uio_oe` is 0 in its RTL), and together they set its UART's bit period. On the Pmod you draw for it in Lab 03, each switch connects its `uio` line to +3V3, so each line needs a pull-down to ground. A closed switch then reads as 1, and an open one reads as 0 rather than as whatever the line picks up.

Lab 04′s board has another input of this kind, U1′s input on the `ui_in[0]` line, which nothing drives while no dongle is plugged in. A floating input on a powered buffer can switch back and forth, and U1 would pass that on to SimProc as garbage. R4, 100 kΩ, pulls that input up to +3V3. It pulls up rather than down because a UART line idles high, and a receiver reads a line held low as a "break". With the kit off and the dongle on, R4 passes up to 3.3 V / 100 kΩ = 33 µA from the dongle's TX into the kit's +3V3. That current goes into a supply rail rather than into a pin, and it's 1/80 of the 2.7 mA that a bare 1 kΩ would put into `ui_in[0]`. With the kit on, a dongle that's plugged in but unpowered may pull U1′s input down to about 0.6 V through the clamp diode on its TX pin, and so hold `ui_in[0]` low, because R4, at 100 kΩ, is too weak to stop it. That's our inference, not a measurement, and Lab 04′s reference bring-up plan says to keep the dongle powered whenever it's plugged in.

Check yourself

1. A teammate wants to connect a 5 V Arduino output straight to `ui_in[2]`. What do you tell them, and what would you put in between?

   Answer: A1, in Answers at the end.

2. With the kit switched off, a dongle's TX at 3.3 V drives `ui_in[0]` through a 1 kΩ resistor and nothing else. Roughly how much current flows into the chip? How much reaches it on Lab 04′s board, where the TX goes to U1′s input and U1′s output goes to `ui_in[0]` through R1?

   Answer: A2, in Answers at the end.

3. Why can each UART line on Lab 04′s board take a 1 kΩ resistor after its buffer when a 50 MHz clock line can't?

Answer: A3, in Answers at the end.

4. Why does U1 run from the kit's 3.3 V, which it takes from the Pmod sockets, and not from the dongle's supply?

   Answer: A4, in Answers at the end.

5. Why does each of SimProc's eight baud-rate switches need a pull-down?

   Answer: A5, in Answers at the end.

Next: Supplies, back-powering and ESD.

Answers

A1. Check yourself

The pads aren't 5 V tolerant, and on the demo board they run at 3.3 V, so a 5 V high would push current through the pad's upper diode into the 3.3 V supply, with nothing to hold it to SkyWater's 100 µA. Put a 74LVC buffer that is specified for partial power down, such as the SN74LVC2G17, in between, powered from the kit's 3.3 V and with a resistor after it, as on Lab 04′s UART lines. Its inputs take up to 5.5 V, and with the kit off its I_{off} keeps the current into the chip to 10 µA or less. A series resistor alone runs into the same conflict as on `ui_in[0]`, because in the modes that leave the `ui_in` pins as inputs the firmware pulls every one of them down. Holding the current into an unpowered chip to 100 µA from 5 V takes (5 − 0.6) V / 100 µA = 44 kΩ, and against a 36 kΩ pull-down that leaves a high of 5 V × 36 kΩ / 80 kΩ = 2.25 V, below the pad's 2.31 V.

A2. Check yourself

Through the resistor alone, with V_{DDIO} near 0 V and about 0.6 V across the diode, $(3.3 - 0.6)\ \mathrm{V}/1\ \mathrm{k\Omega}$ = 2.7 mA, 27 times SkyWater's 100 µA. On Lab 04′s board U1 is unpowered too, so its output passes at most its I_{off}, 10 µA. R4 also passes up to 33 µA from the TX into the kit's +3V3, but that goes into the supply rail, not into a pin.

A3. Check yourself

The resistor and the capacitance at the pin it drives set the edge time, $2.2RC$. With the 10 pF supposed above, 1 kΩ gives edges of about 22 ns. That's a small fraction of a UART bit, 8.7 µs at 115 200 baud, but it's longer than the 10 ns that a 50 MHz clock stays high.

A4. Check yourself

U1′s I_{off} applies only while its own supply is at 0 V, so U1 has to be off whenever the chip is off. Powered from the dongle, it would stay on with the kit off and drive the unpowered `ui_in[0]` through R1, with about 2.7 mA, as if the buffer weren't there.

A5. Check yourself

An open switch leaves its `uio` input undriven, and a floating CMOS input can drift to the logic threshold, where it can read as either value and the input stage draws current. The pull-down makes an open switch read as 0.

16. Supplies, back-powering and ESD

This page answers: *where does each of the chip's supplies come from, what happens to the chip when one board is powered and the board connected to it isn't, and where does static get in?*

Protecting the chip's I/O dealt with each pin on its own. This page starts with the supplies behind the pins. It shows how the breakout makes them, and what SkyWater says about the order in which they come up. Then it follows the current that a powered board can push backwards into an unpowered one, and the current that flows when two boards drive the same line to different levels. Last, it covers static discharge. The demo board has TVS diodes on some of its lines but not on the pins your Pmod reaches, so you put the protection for those pins on your own board.

The chip's three supplies

The SKY130 chip on Tiny Tapeout's breakout has three supply rails: `VCCD`, the 1.8 V core supply; `VDD_IO`, the supply of the I/O pads; and `VDDA`, the analog supply. The breakout has footprints for two regulators, and as it's built only one of them is fitted. Tiny Tapeout's boards, part by part walks through the whole board, and the figure below shows where each of the three supplies comes from.

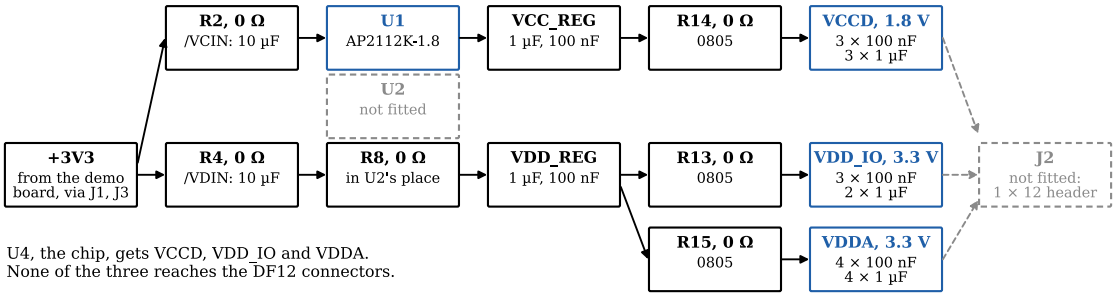


Figure 16.1: Where the SKY130 breakout's three supplies come from

The supplies on Tiny Tapeout's SKY130 QFN breakout, rev 3.2, as fitted, from the board file at 91a2bf1 (drawn by `scripts/figures/sky_breakout_rails.py`). U1, an AP2112K-1.8 regulator, makes `VCCD`. `VDD_IO` and `VDDA` come from the demo board's 3.3 V through four 0 Ω links, with no regulator in between. Both rails go through R4 and then R8, which bypasses the empty footprint of the second regulator, U2. After that, R13 goes to `VDD_IO` and R15 to `VDDA`. The dashed parts are marked DNP, for "do not populate", so they aren't fitted.

So a board plugged into the demo board can't set the chip's supplies one at a time. The only supplies on the breakout's connectors are +3V3, +5V and ground, so changing +3V3 changes `VDD_IO` and `VDDA` together, while `VCCD` stays at 1.8 V as long as U1 has enough input. `VDD_IO` is also joined to the demo board's +3V3 by just three 0 Ω links (R4, R8 and R13), so anything that pushes current into the chip's I/O supply pushes it into the demo board's whole 3.3 V net as well.

A regulator and its capacitors

A regulator's datasheet says which capacitors it needs on its input and output, so U1's capacitors are part of the regulator's design. Diodes Inc.'s AP2112 datasheet describes a 600 mA CMOS low-dropout regulator that is "Stable with 1.0µF Flexible Cap: Ceramic, Tantalum and Aluminum Electrolytic", and it specifies the regulator's electrical characteristics with 1.0 µF ceramic capacitors on the input and the output. The breakout gives U1 10 µF on its input (C6), and 1 µF and 100 nF on its output (C18 and C21). The six capacitors on `VCCD` itself come after those, on the far side of R14.

Other regulators are stricter. Horowitz and Hill's example (*The Art of Electronics*, 3rd ed., p. 611) is the TPS775xx, whose output capacitor "must be at least 10 µF (with an ESR no less than 50 mΩ and no greater

than 1.5 Ω)", and whose datasheet plots the regions of stability and instability. So when you put a regulator on a board, take the value of its output capacitor from the datasheet, and take its type and series resistance from there too whenever the datasheet sets them, as the TPS775xx's does.

The order the supplies come up in

We found no required power-up order between `VDD_IO`, `VCCD` and `VDDA` in SkyWater's user guide for its I/O cells, and no power-sequencing statement from Tiny Tapeout for the chip itself. The guide's rules are about a supply that is off, and about what the pads do while the supplies come up:

- "no single power supply will be grounded while another supply is active. This will help alleviate the grounded NWELL issues that induce latch-up. It is OK to float a supply while another is active though."
- Before `vccd` is taken to 0 or left floating, the cell's hold mode must be asserted.
- During power-up a control signal, `enable_h`, is held low, and "the output driver is forced to a known state (Hi-Z)".

These rules are written for the people who design the chip's I/O ring, and the guide says that the chip team is responsible for the levels on those supplies. On the breakout as it's fitted, `VDD_IO` and `VDDA` rise with the demo board's +3V3, and `VCCD` follows once U1 has started, so the board fixes the order and nothing on a Pmod can change it.

Back-powering

Tiny Tapeout's ETR guide uses "back-powered" for feeding the boards 3.3 V on purpose, through a header pin. None of the breakout boards so far needs the 5 V that comes in from USB, it says, so "the boards can be back-powered by feeding 3v3 on a header pin marked as such". On this page the word means an accident instead, in which a chip is powered through its own input pins.

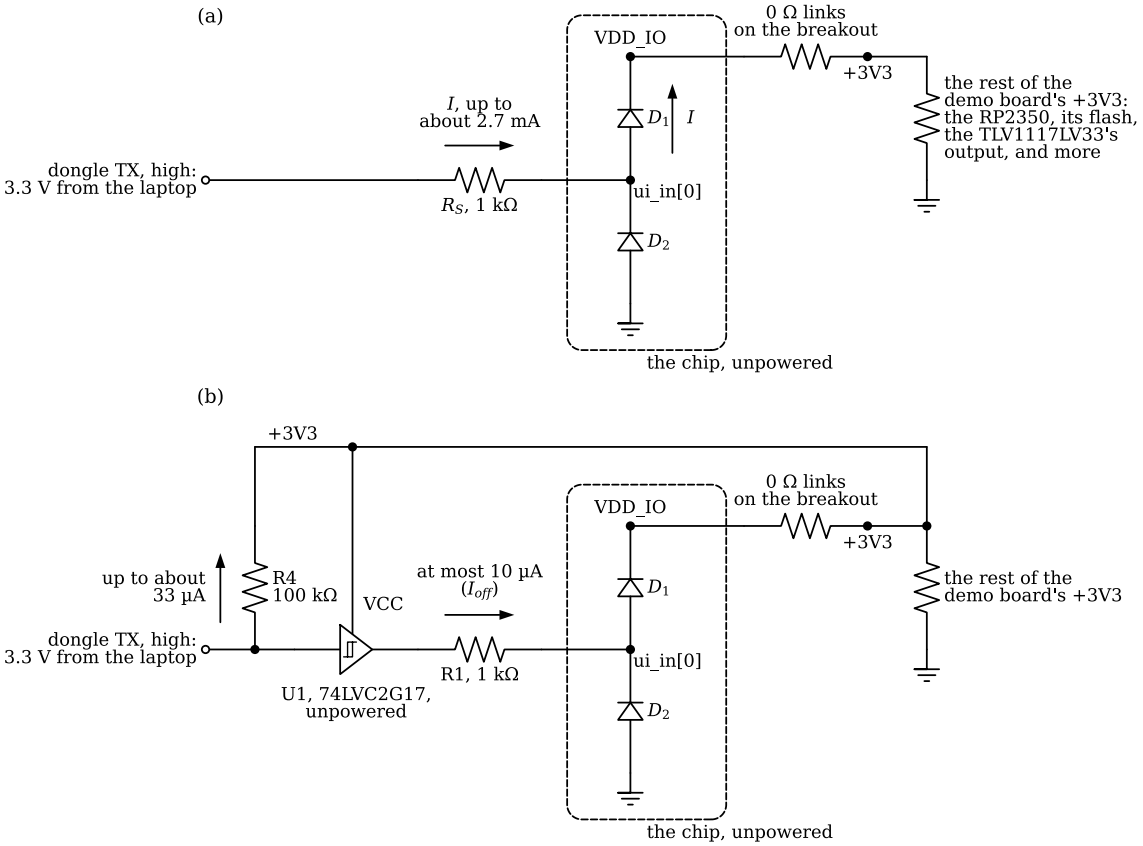


Figure 16.2: A powered dongle driving an input of an unpowered kit, through a series resistor in (a) and through a buffer and a resistor in (b)

The kit is switched off, but the USB-UART dongle on the SimProc Pmod takes its power from the laptop, and the dongle's TX line sits high between characters. (a) With only a series resistor R_S between the dongle and `ui_in[0]`, current flows through R_S and the pad's upper protection diode into `VDD_IO`, and from there through the breakout's 0 Ω links onto the demo board's +3V3 net. With 1 kΩ, that's about 2.7 mA. (b) On Lab 04's board, the dongle's TX goes through U1, a 74LVC2G17 buffer powered from the kit's +3V3, and then through R1 to `ui_in[0]`. U1 is unpowered too, so at most 10 µA, U1's I_{off}, reaches `ui_in[0]`. The pull-up R4 puts up to about 33 µA onto +3V3. Each panel draws the three links as one resistor, and everything else on +3V3 as a second resistor. In the demo board's file, that net includes its RP2350, the RP2350's flash memory and the output of its TLV1117LV33 regulator.

Horowitz and Hill say that a HIGH input on an unpowered chip without tolerant inputs will "partially power the supply rail" through the clamp (p. 795). The "craziest" case they give is a CMOS chip whose supply pin was never wired up, and which works anyway because it is "powered by one of its logic inputs (via the chip's internal input-protection diodes from input to VDD)", until the moment all of its inputs are low at once (p. 760). Current through those input diodes can also start latch-up (p. 759). Their advice for low-power CMOS boards includes the same warning: "be careful whenever mixing supply voltages. Otherwise you may have current flowing through input-protection diodes" (p. 754).

SkyWater's rules cover a supply that is grounded and a supply that is floating, and in this case the kit's +3V3 is neither, because the dongle's current holds it up at a voltage that depends on that current and on the rail's load. You can keep that current small, or stop it, in three ways:

- **A series resistor** on the line limits the current to $(3.3\ \text{V} - 0.6\ \text{V})/R_S$, and keeping it under the 100 µA that SkyWater allows into a pin driven above its supply takes at least 27 kΩ. Protecting the chip's I/O shows why a resistor that big doesn't work on `ui_in[0]`. The demo board's firmware turns on the RP2350's pull-down on that line, and against the strongest pull-down a resistor over 15.4 kΩ leaves the high below the 2.31 V that the chip needs.
- **A buffer whose inputs accept a voltage even while it's unpowered** sits between the dongle and the chip, powered from the 3.3 V that your Pmod takes from the kit, so when the kit is off, the buffer is off too. In Lab 04 you add a 74LVC2G17, which holds two such buffers, one for each UART line. After each buffer you put a 1 kΩ resistor, R1 on the way into `ui_in[0]` and R2 on the way out to the dongle. With its supply at 0 V, TI specifies each of the part's pins for at most 10 µA (Ioff), so the buffer's input can take the dongle's 3.3 V, and its output passes at most 10 µA into `ui_in[0]`, a tenth of SkyWater's limit. R4, 100 kΩ, holds the buffer's input high while no dongle is plugged in. With the kit off, it passes up to 3.3 V / 100 kΩ = 33 µA from the dongle into the kit's +3V3. On the line out of the chip, the dongle's RX pull-up pushes current into the second buffer's unpowered output, which passes at most 10 µA. `uo_out[0]` is on that buffer's input side, so none of the current reaches it.
- **A written order** for powering and unplugging the boards keeps the dongle from feeding an unpowered kit at all. The buffer already keeps the chip within SkyWater's limit, so on Lab 04's board the order is a second protection. It keeps the dongle's TX off the buffer's unpowered input and stops the 33 µA through R4. In Lab 04 you'll write it in `BRINGUP.md`, which says which board is powered first and which is unplugged first, and why.

It can also happen the other way round. If the kit is on and the dongle is unplugged from the laptop, the buffer drives SimProc's TX into the dongle's unpowered RX input, and current flows through the dongle's own clamp diode into its supply. R2 limits that current to (3.3 V − 0.6 V) / 1 kΩ = 2.7 mA. None of the dongle datasheets we read gives a limit for it, so allowing 2.7 mA is a judgement of ours. An unpowered dongle may also pull the buffer's input low through a clamp diode on its own TX pin, and the buffer then holds `ui_in[0]` low (Protecting the chip's I/O describes this), so keep the dongle powered whenever it's plugged into your board.

Contention

Contention means that two outputs are driving one line to different levels at the same time. Only the two drivers' output resistances limit the current, and it flows for as long as they disagree. A Pmod for a Tiny Tapeout project is never the only thing connected to its pins (Pins, Pmods and planning a pinout has the list), so for each pin you need to know what the other drivers on it are doing:

- **The RP2350.** The demo board's firmware has three modes, which the ETR guide's example `config.ini` lists: `SAFE`, with "all RP2 I/O connected pins inputs"; `ASIC_RP_CONTROL`, in which the "TT inputs,nrst and clock" are driven by the RP2350; and `ASIC_MANUAL_INPUTS`, "basically same as safe, but intent is clear". The example sets `ASIC_RP_CONTROL` as the default, and `config.ini` can set a different mode for each project. A SimProc Pmod drives `ui_in[0]` from the dongle, through its buffer, so SimProc's section of

`config.ini` has to select `ASIC_MANUAL_INPUTS`. On the `uio` pins, a separate option called `uio_oe_pico` decides which of the RP2350's pins are outputs, and for SimProc it has to leave all eight as inputs, because the switches on your Pmod drive those lines.

- **The DIP switches.** A switch that's on connects its `ui_in` line to +3V3 through 1 kΩ. If your board drives the same line low directly, 3.3 V / 1 kΩ = 3.3 mA flows into your driver, and the line's low level depends on how stiff that driver is. If your board drives it low through 1 kΩ, as Lab 04's buffer does through R1, then 3.3 V / 2 kΩ = 1.65 mA flows through the two resistors, and they leave the line at 1.65 V. That's between the chip's input thresholds of 0.99 V and 2.31 V (0.3 and 0.7 × `VDD_IO`), so the chip can read it either way. For SimProc, switch 1, the one on `ui_in[0]`, has to be off.
- **The chip.** On a `uio` pin, the project's own `uio_oe` decides whether the chip drives it. SimProc sets `uio_oe` to 0, so its eight `uio` pins are inputs and your switches never fight it.

R1 limits a fight on `ui_in[0]` with a stiff driver, such as the RP2350 in `ASIC_RP_CONTROL`, so the worst-case contention current is about 3.3 V / 1 kΩ = 3.3 mA. The buffer is a strong driver, rated for 24 mA of output current at 3.3 V, so without R1 only the two output stages would limit the fight.

ESD

Static discharge is another way for current to get into a pin from outside. The human body model, which chip makers test with, treats a charged person as 100 pF in series with 1.5 kΩ, and Horowitz and Hill (p. 804) give its usual charging voltages as 1 to 2.5 kV, with some ICs rated up to 15 kV. At 2.5 kV that's a peak current of 1.67 A for a pulse with a time constant of 150 ns. Their estimate is that such a discharge delivers 15 µJ to an unprotected input, and that a pair of external clamp diodes, one to the supply and one to ground, with a series resistor on each side of them, cuts that to about 0.1 µJ (p. 805). Ordinary diodes like these have more junction area than the clamps on a chip, so they clamp better. The resistor before them limits the current through the diodes, and the resistor after them limits the current into the chip.

Static gets in wherever something from outside touches the board, such as a connector, a cable or a finger. The demo board's file has TVS diodes, which are parts made to clamp such pulses, in two places. D1 and D2, valued "TVS", sit on the USB data lines. Twenty-two more, valued "5VTVS" and in SOD-882 packages (D6 to D27), sit on the lines `/A` to `/X`, which run from the breakout's connector to an unfitted 2 × 16 header. We matched the demo board's connector to the SKY130 breakout's pad by pad, and the sixteen digital project lines on that connector all line up. On the breakout, those 22 lines are the analog pads `an0` to `an11`, two `res_an` lines and eight grounds. The digital project pins have no TVS diodes, so the `ui_in`, `uo_out` and `uio` lines that your Pmod reaches have only the chip's own protection, and a cable from a dongle gives static a direct path to them. In Lab 04 you'll put ESD protection on the connector that the dongle's cable plugs into, from each of its signal pins to ground, so that a discharge arriving on the cable reaches the ESD part before anything else on your board.

Static can also reach the chip while you're handling the breakout, and each kit has only one chip, so work on an ESD mat, wear a wrist strap, and hold the breakout by its edges.

Check yourself

1. The kit is off, and the SimProc dongle is plugged into both the laptop and your Pmod. Where does current flow, and what limits it?

   Answer: A1, in Answers at the end.

2. Why does SimProc's section of `config.ini` need `ASIC_MANUAL_INPUTS` when the Pmod is plugged in?

   Answer: A2, in Answers at the end.

3. Can a board plugged into the demo board run the chip's I/O at 1.8 V while `VDDA` stays at 3.3 V?

   Answer: A3, in Answers at the end.

4. Which lines on the demo board have TVS diodes, and what does that mean for your Pmod's dongle header?

   Answer: A4, in Answers at the end.

Next: Measuring current.

Answers

A1. Check yourself

On the board as Lab 03 leaves it, nothing sits between the dongle and the chip. Current flows from the dongle's TX, which idles high, into `ui_in[0]`, through the pad's upper protection diode into `VDD_IO`, and through the breakout's 0 Ω links onto the demo board's +3V3, where it partly powers everything on that net. Only the dongle's output stage and the diode's drop limit it. The dongle's RX pull-up also pushes current into the unpowered `uo_out[0]`, up to 164 µA for a CH340, one of the chips that dongles use, as Protecting the chip's I/O works out. On the board as Lab 04 leaves it, the TX drives the unpowered buffer's input, and at most 10 µA, the buffer's Ioff, reaches `ui_in[0]`. R4, the pull-up on that input, passes up to 33 µA from the TX into the kit's +3V3. The dongle's RX pull-up pushes at most 10 µA into the buffer's other output, and none of it reaches `uo_out[0]`.

A2. Check yourself

In `ASIC_RP_CONTROL`, the default in the example configuration, the RP2350 drives the `ui_in` pins. The dongle drives `ui_in[0]` too, through your buffer, so the two would fight. `ASIC_MANUAL_INPUTS` leaves the RP2350's pins as inputs.

A3. Check yourself

No. The only supplies on the breakout's connectors are +3V3, +5V and ground, and on the fitted breakout `VDD_IO` and `VDDA` both come from +3V3 through 0 Ω links. To separate them you'd have to remove R13, the link that feeds `VDD_IO`, and feed `VDD_IO` from outside through the unfitted header J2.

A4. Check yourself

The USB data lines, and the lines `/A` to `/X`, which connect the breakout's analog pads, and some of its ground pins, to the unfitted analog header. The digital project pins have none, so the dongle header needs its own ESD part, from each of its two signal pins (the dongle's TXD and RXD) to ground.

17. Measuring current

This page answers: *how do you measure the current a board draws, and how much does the measurement change what you're measuring?*

Current is the first thing you read when you power a new board. A board that draws far more than you expected may have a short, a latched-up chip or a damaged part, and one that draws nothing may not be powered at all. Bringing up a chip puts these readings into a bring-up plan, with the value you expect at each step. In Lab 04 you'll add a current-sense point to your Pmod and write down what it should read.

A meter in series, or a sense resistor

There are two ordinary ways to measure a current on the bench. Both put a resistor in the current's path and read the voltage across it. On a multimeter's current range the resistor is inside the meter, and the other way is to build one into your board as a sense resistor.

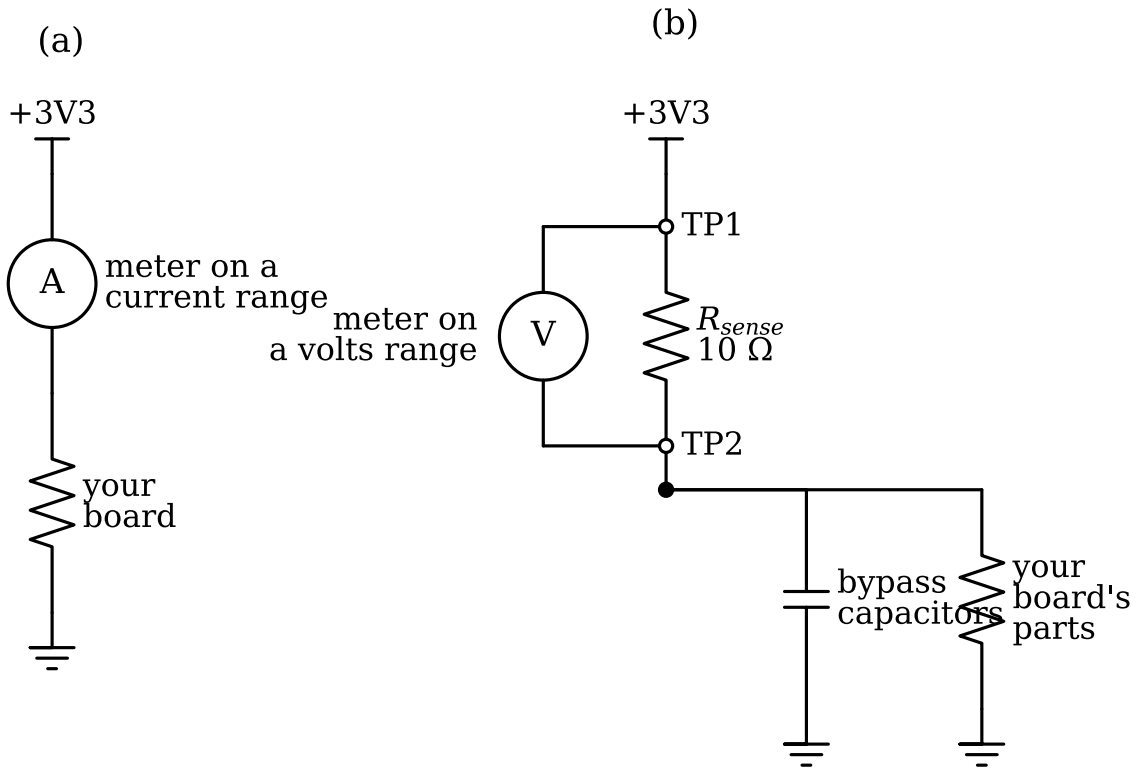


Figure 17.1: A meter in series, and a sense resistor with two test points

(a) A multimeter on a current range, in series with the supply. (b) A sense resistor in the supply, with a test point at each end and a multimeter on a volts range across them. In (b) the bypass capacitors are on the board's side of the resistor, as in Horowitz and Hill's Figure 10.93.

A meter in series. Selecting a current range on a multimeter "puts a small resistor across the meter's input terminals", Horowitz and Hill write (*The Art of Electronics*, 3rd ed., p. 10). The resistor is typically chosen to drop 0.1 V to 0.25 V at the range's full-scale current, and the meter displays that voltage as a current. They note that the drop is sometimes called the "voltage burden". Because the meter sits in series with your board, the drop comes off the supply voltage the board sees. A meter in series is convenient because nothing on the board has to be built for it, but the board does need a place where you can break the supply and put the meter in.

A sense resistor on the board. You build the resistor into the board's supply, put a test point at each end, and read the voltage across it with a meter on a volts range. Horowitz and Hill recommend one on CMOS boards (p. 754). A CMOS chip damaged by static can start drawing excess quiescent current, and "a 10 Ω resistor in series with VDD on each board" makes it easy to see when that happens. The caption of their Figure 10.93 calls the resistor a "current spy". Elsewhere they suggest 1 Ω in series with each chip's supply lead "(with downstream bypassing)" to find a faulty chip, and 0.1 Ω "for power drivers or chips driving many outputs" (p. 760). The bypass capacitors go on the board's side of the resistor, so that the fast pulses of current at each logic edge come from the capacitors and the resistor carries mostly the average current, which is the one you want to read.

Choosing the resistor

The voltage across the resistor is $V = IR$. You want it large enough for your meter to read, and small enough that the board is still left with the supply voltage it needs.

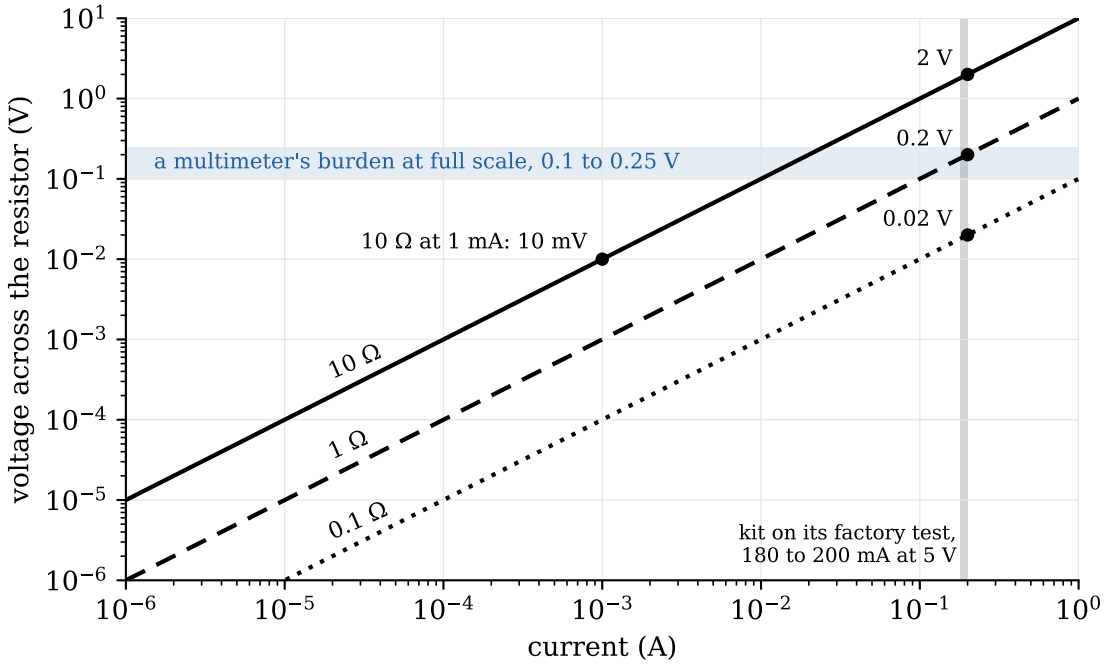


Figure 17.2: Voltage across a sense resistor against current

$V = IR$ for 0.1 Ω, 1 Ω and 10 Ω, drawn by `scripts/figures/shunt_drop.py`. The blue band is the 0.1 V to 0.25 V that a multimeter drops at full scale, and the grey band is the 180 mA to 200 mA that the whole kit draws from USB on its factory test.

The figure gives these numbers for three cases:

- A Pmod that draws 1 mA through a 10 Ω spy drops 10 mV across it. A multimeter reads 10 mV easily, and 10 mV is too small a part of the board's 3.3 V to matter.
- If the whole kit's draw, about 200 mA, went through the same 10 Ω, the resistor would drop 2 V, and a 3.3 V rail would be left with 1.3 V. At that current you'd use 1 Ω (0.2 V) or 0.1 Ω (20 mV), the values Horowitz and Hill suggest for single chips and for heavy drivers.
- A board that idles at 10 µA and runs at 10 mA covers three decades of current. A resistor that drops 10 mV at 10 mA drops only 10 µV at 10 µA, and you'd need a meter that resolves microvolts to read that. So either measure the two cases with different resistors, as a meter does when you change its range, or choose the resistor for one case and accept that it won't give a useful reading of the other.

What you can measure on a Tiny Tapeout kit

The whole kit. Tiny Tapeout's ETR demo board guide says that with the factory test project running, "you should see a draw of around 180-200 milliAmps, on the 5V supply and not much more". That's the

current into the USB connector, for the demo board and the breakout together, and it's the only reading you can take without a board of your own or rework on the breakout. You can read it with a USB power meter, or with any other current meter in series with the USB cable. If it reads well above that range, switch off.

The breakout's total. The breakout takes all of its power through its two connectors. They carry +3V3 and +5V, but on the fitted breakout +5V goes nowhere, because R1 and R3 aren't fitted, and those are the links that would take it to the regulators. +3V3 feeds `VDD_IO` and `VDDA` through 0 Ω links, and `VCCD` through the regulator U1. So a motherboard, a board of your own in the demo board's place, can measure the breakout's total current on +3V3. That total includes U1's own quiescent current, which the AP2112 datasheet gives as 55 µA typical.

One of the chip's rails. You can measure a single rail only after rework on the breakout. Its schematic says each supply pin "may be monitored or fed through these headers by popping off the relevant resistor (0805s for easier manip)". The resistors are the 0 Ω links R13 (`VDD_IO`), R14 (`VCCD`) and R15 (`VDDA`), and the header is J2, a 1 × 12 header that the board marks as not fitted. To measure `VCCD`, for example, you'd remove R14, fit J2, and connect a meter or a sense resistor between the J2 pins on the regulator's side and on the chip's side.

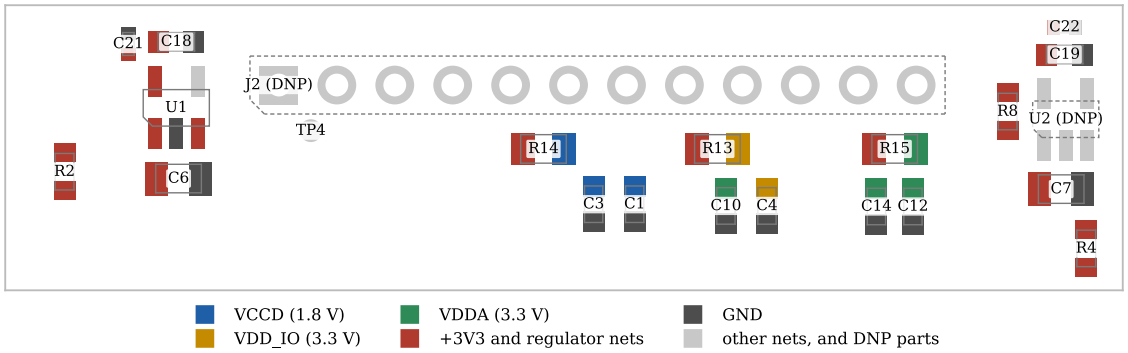


Figure 17.3: The top edge of the SKY130 breakout, with the 0 Ω links and J2

A 12.5 mm strip along the top edge of Tiny Tapeout's SKY130 breakout, drawn to scale from the board file by `scripts/figures/sky_breakout_map.py`, with pads coloured by supply. R14, R13 and R15 each join a regulator net (red) to one of the chip's rails. J2's twelve pads, and U2's, are grey because those parts aren't fitted.

That rework is on the board that carries the kit's only chip, which is why the course doesn't ask you to do it. The current a project draws lowers the supply voltage that reaches it inside the chip. Tiny Tapeout's analog page says that "A draw of around 20mA from the power supply will result in a 0.1V drop through the PDN", where the PDN is the chip's internal power network. If the team needs per-rail readings, practise the rework first on a breakout made from Tiny Tapeout's open files, with its passives and no chip.

Your Pmod. The current-sense point you'll add in Lab 04 sits in your Pmod's own supply, the +3V3 it takes from pins 6 and 12 of the sockets. It measures the current drawn by the parts on the far side of its resistor. The buffer on the UART lines isn't one of them, because it takes its 3.3 V from the sockets' side of the resistor. The chip's rails don't reach the Pmod, so it can't tell you what the chip draws. You can still work out what the sense point should read, and a reading far from that number means there's a mistake on your board. In Lab 03 you gave each of the eight baud-rate switches a 10 kΩ pull-down, which is also the value in Horowitz and Hill's switch circuit (p. 802). Every switch that's on draws 3.3 V / 10 kΩ = 0.33 mA through its pull-down, so all eight together draw 2.64 mA.

Reading the voltage

To read a sense resistor, put a multimeter on its millivolt range, with its leads on the two test points. A bench oscilloscope can show how the current changes over time, but its probe's ground clip is connected to mains earth, so clipping it to either end of a resistor in the supply shorts that point to ground. Bringing up a chip covers probing safely.

Check yourself

1. You put a 10 Ω current spy in the +3V3 that feeds the whole kit, which draws about 200 mA. What happens?

   Answer: A1, in Answers at the end.

2. Your multimeter's 200 mA range drops 0.25 V at full scale. About how much does it drop while it measures your Pmod's 2.6 mA?

   Answer: A2, in Answers at the end.

3. Can you measure the chip's `VCCD` current on a kit as it comes, without rework?

   Answer: A3, in Answers at the end.

4. Why do the bypass capacitors go on the board's side of a sense resistor rather than on the supply's side?

   Answer: A4, in Answers at the end.

Next: Bringing up a chip.

Answers

A1. Check yourself

The resistor drops 200 mA × 10 Ω = 2 V, which leaves about 1.3 V for the kit. A 10 Ω spy is for a board that draws a few milliamps. For the whole kit you'd use 0.1 Ω or 1 Ω, or measure at the USB input.

A2. Check yourself

The drop is proportional to the current through the meter's internal resistor, so it's 0.25 V × 2.6 / 200 ≈ 3.3 mV, far too little to affect the Pmod.

A3. Check yourself

No. `VCCD` doesn't reach the two connectors that join the breakout to the demo board. The most you can measure without rework is the whole kit at USB, or the breakout's total on +3V3 from a board of your own. Measuring `VCCD` alone means removing R14 and fitting J2.

A4. Check yourself

So that the fast current pulses at each logic edge come from the capacitors, and the resistor carries mostly the average current you want to read. With the capacitors on the supply's side, the resistor would sit between the board and the nearest stored charge, and every pulse would have to come through it.

18. Bringing up a chip

This page answers: *the first time you power a board with one of the team's chips on it, what do you do, and in what order, so that a mistake costs you a resistor and not the chip?*

Bring-up is the first power-up and first test of new hardware. For a board of your own it's mostly a matter of patience, but one of the team's Tiny Tapeout chips needs more care. Tiny Tapeout's FAQ says "You only get 1 chip", and that if you want more you have to order more devkits. It also says "We don't ship bare dies either", so you can't buy the chip by itself. Because of that, bring-up with a team chip goes in small steps, each small enough that if something is wrong, a reading shows it before the chip is damaged.

In Lab 04 you write a `BRINGUP.md` for your Pmod. It opens with a table of every line that leaves your board, which says what drives each line, what it drives, and what happens when one side is powered and the other isn't. Then come your settings (the current limit, the firmware mode, the switch positions and the clock), the checks you make before anything is powered, and the steps in order, each with the reading you expect and its unit. It ends with what to do when a reading is wrong. This page explains what goes in each part after the table, and why.

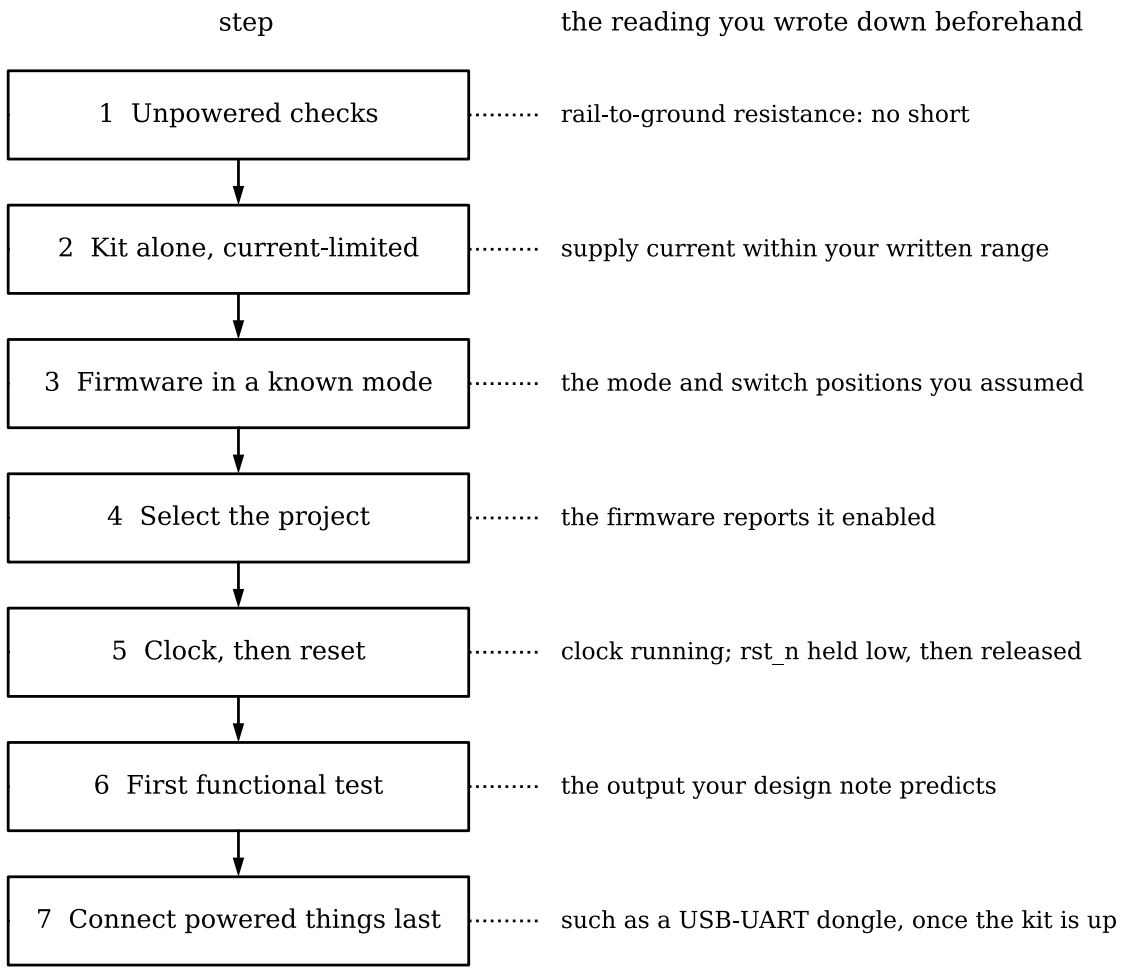


Figure 18.1: A flow chart of seven bring-up steps, each with the reading written down beforehand

The order of a first power-up. Each step has a reading that you write down before you do it. If a step doesn't give that reading, you turn the power off and find out why before you go on. Made by `scripts/figures/bringup_order.py`.

Before any power

The review

Nobody powers a kit from your board until a second member of the team has read its `BRINGUP.md`. The team doesn't yet have anyone who has reviewed boards before, so until it does, that reading and the checklist in Reviewing and ordering a board are the review.

Handling

Static can damage a chip before it has ever been powered. Chip makers model a charged person touching a part as 100 pF discharging through 1.5 kΩ, which is called the human body model. They usually test with it charged to 1 to 2.5 kV, and some ICs are rated up to 15 kV (Horowitz and Hill, *The Art of Electronics*, 3rd ed., §12.1.5). Charged to 1 kV, the model puts a peak of 0.67 A into whatever pin you touch first. At team sessions, work on an ESD mat with a wrist strap, and hold the breakout by its edges.

Unpowered checks

Before you measure anything, work out from the schematic what each reading should be and write it down, so that you have something to compare each measurement with. Then, with nothing connected, use a multimeter to measure the resistance from each supply rail of your board to ground. A capacitor on the rail, such as C1, the 100 nF capacitor that you put beside the UART lines' buffer in Lab 04, makes the reading start low and climb for a moment while the meter charges it, so read the value once it settles. On Lab 04's board, with every switch off, it settles above 1 MΩ. A rail that stays close to 0 Ω is shorted, and you stop there, because a short that you find with a meter costs you nothing, while the same short found by powering the board may cost you a trace or a part.

Orientation

Check that every connector goes in the right way round. Tiny Tapeout's breakouts are keyed "such that direction of insertion onto hosts is made clear", in the words of their README. Your Pmod's connectors and any header you add may not be keyed, so mark pin 1 on your board's silkscreen when you draw it, and look at the mark before you plug anything in.

The kit alone, with a current limit

Power the kit first without your board, from a supply whose current you can see and limit. That tells you whether the kit works at all, and how much current it draws. Tiny Tapeout's demo board guide says that with the factory test project running, "you should see a draw of around 180-200 milliAmps, on the 5V supply", so you have a number to compare with. Run that project before your own, so that you know the chip, the breakout and the demo board all work before anything of yours is involved.

Your Pmod can't measure the chip's current, because the chip's current never flows through your board. The chip has three supply rails, and all three stay on the breakout: `VCCD`, at 1.8 V from the breakout's regulator, and `VDD_IO` and `VDDA`, both at 3.3 V. The only power pins on the two DF12 connectors that join the breakout to the demo board are +3V3, +5V and GND, and a Pmod gets only what the demo board passes on to its sockets. The chip's current is therefore part of the kit's total, and you read it at the USB input.

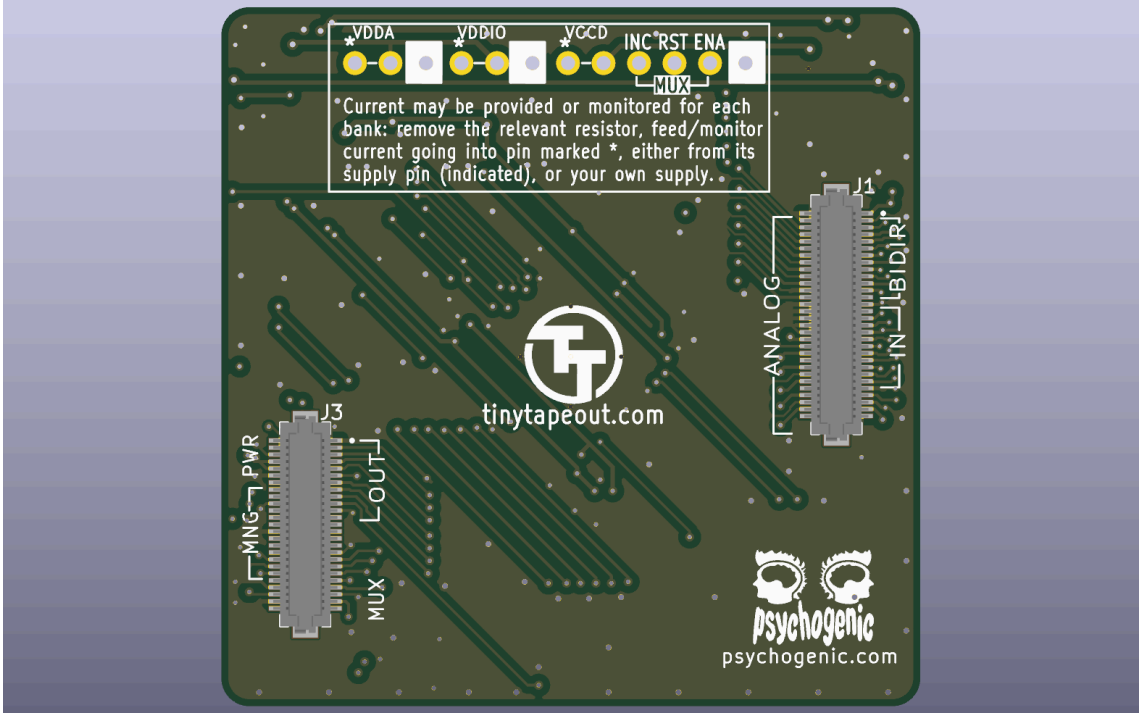


Figure 18.2: The bottom of Tiny Tapeout's SKY130 breakout: the DF12 connectors J1 and J3, and the unfitted supply header with a note about measuring each supply

*The bottom of Tiny Tapeout's SKY130 QFN breakout, rev 3.2 (`breakout-pcb` at 91a2bf1, Apache-2.0), rendered by `kicad-cli` with the parts' 3D models. J1 and J3 are the DF12 connectors to the demo board. The row of pads at the top is J2, a header that isn't fitted on the boards Tiny Tapeout ships, so the render leaves out its part and shows only the pads. To measure one supply, the note on the silkscreen under it says to "remove the relevant resistor, feed/monitor current going into pin marked *". Made by `scripts/figures/board_renders.sh`.*

Measuring one rail on its own means taking an 0805 link off the breakout and fitting J2. That's rework on the board that holds the kit's only chip, so the course doesn't ask you to do it (Measuring current describes the links and J2). If the team ever needs per-rail readings, practise the rework first on a breakout without a chip.

Your Pmod can measure its own draw. A 10 Ω resistor in your board's supply turns your board's current into a voltage that you can read with a meter. Horowitz and Hill call this a "current spy" (§10.7.1), and they recommend one on every board, because a chip damaged by static can draw too much current, and the spy makes that easy to find. Work out from the part values what the spy should read, and write it down. It sees only the current of the parts on the far side of the resistor from the sockets. On Lab 04's board those are the switches and their pull-downs. The buffer U1 that you add in Lab 04, C1 and the pull-up R4 take their 3.3 V from the sockets' side of the resistor, so their current doesn't pass through it.

If you power your board on its own from a bench supply, before it goes into the kit, first hold at a fixed level each buffer input that the chip would drive. A floating input on a powered buffer can switch back and forth, and TI's datasheet for U1, a 74LVC2G17, says that "All unused inputs of the device must be held at VCC or GND". On Lab 04's board that's the input on pin 1 of your Pmod's J3, its connector for the chip's outputs. A 10 kΩ resistor from that pin to GND holds it low during the bench test, and comes off before the board goes into the kit.

The order of the supplies

SkyWater's user guide for its I/O cells, the cells around the edge of the chip, gives no required order in which `VDDIO`, `VCCD` and `VDDA` have to come up, and we found no statement about sequencing for the chip itself from Tiny Tapeout or ChipFoundry. The guide does say three things about the supplies:

- "no single power supply will be grounded while another supply is active", and "It is OK to float a supply while another is active though";
- before `VCCD` is taken to 0 or left floating, "the hold mode must be asserted";

- during power-up, `enable_h` is low and "the output driver is forced to a known state (Hi-Z)", so the pads' outputs don't drive anything while the supplies come up.

On the breakout as it's built, every rail comes from the demo board's +3V3. `VDD_IO` and `VDDA` are connected to it directly through 0 Ω links, and `VCCD` comes through the 1.8 V regulator. By our reading of the board, that means `VCCD` comes up a little after the other two, by however long the regulator takes to start. Tiny Tapeout ships the breakout that way, and a board of yours that powers the kit through the demo board gets the same order.

A known firmware state

The demo board's RP2350 runs Tiny Tapeout's MicroPython firmware, and which pins it drives depends on the firmware's mode (firmware README):

Mode	What the RP2350 does
`RPMode.SAFE`	"all RP2 pins are inputs"; a comment in the README's example calls it "the default"
`RPMode.ASIC_RP_CONTROL`	drives the project's inputs ("ui_in pins are OUTPUTS for the RP2")
`RPMode.ASIC_MANUAL_INPUTS`	drives only the management pins, so that you can "twiddle inputs and clock using on-board DIP switches and buttons"

The mode a kit starts in comes from the DEFAULT section of its `config.ini`, and the README's example configuration has `mode: ASIC_RP_CONTROL` there, so a kit set up that way drives every `ui_in` line from the RP2350. The DIP switches can drive the same lines, because each switch, when it's closed, ties its `ui_in` line to +3V3 through a 1 kΩ resistor. A Pmod that also drives a `ui_in` line fights whichever of those is active. Your `BRINGUP.md` says which mode and which switch positions your board assumes, and step 3 in the figure is where you check that the kit is set up that way, before your board is connected.

Select, clock, reset, test

The demo board's README (`tt-demo-pcb`) says, "The RP2350 is responsible for selecting projects, by controlling the project MUX that's on the ASIC and, under most circumstances, providing the clock for the projects". In the firmware, enabling a project "does all the control signal twiddling needed to select and enable the project". `reset_project(True)` holds `rst_n` LOW, and "The button on the demoboards does the same job". The firmware can step the project's clock one cycle at a time or run it continuously. So in steps 4 and 5 you select the project, start its clock, hold the project in reset, and then release it.

Step 6 is the first functional test, which should be the smallest test that shows the design is alive. Write down its expected result, and if you can, choose a test whose result you can see without your board. SimProc's halt and done flags are on `uo_out[1]` and `uo_out[2]`, which drive segments B and C of the demo board's 7-segment display. After a reset, `uo_out` should read 0x03, so segments A (TX, which idles high) and B (`halt`) are lit and C (done) is dark. Lab 04 takes that value from SimProc's documentation.

Things with their own power

Connect your board to the kit while nothing is powered. Connect anything with a supply of its own, such as a USB-UART dongle powered from your laptop, only once the kit is up. Disconnect in the reverse order.

This order keeps a powered dongle from feeding current into the kit while the kit is off. On the board as Lab 03 leaves it, nothing sits between the dongle and the chip, so that current goes into the chip's pins. A CMOS input has clamp diodes to its supply, and if something drives the input HIGH while the chip is unpowered, current flows through the clamp into the chip's supply rail, enough to power it partly (Horowitz and Hill, §12.1.2). Enough of that current can also trigger latch-up, and because latch-up depends on the input current rather than the voltage, a series resistor on the line prevents it (§10.8.3). SkyWater allows a pin to be driven above its supply only if the current into it is limited to 100 µA, though. On `ui_in[0]`, the line the dongle drives, that takes a resistor of at least (3.3 V − 0.6 V) / 100 µA = 27 kΩ, and a resistor that big can leave the high too low for the chip to read, because the demo board's firmware turns on the RP2350′s pull-down on the same line. So after Lab 04 your board has a buffer, U1, between the

dongle and `ui_in[0]`. With the kit off, U1 is unpowered as well, and TI specifies at most 10 µA through each of its pins in that state (its Ioff), so the current into the chip stays under SkyWater's 100 µA even if you connect the dongle before the kit is up. On Lab 04′s board the order is a second protection. It keeps the dongle's TX off U1′s unpowered input, and stops R4, the pull-up on that input, from passing up to 33 µA into the kit's +3V3. Supplies, back-powering and ESD goes through the cases.

When a reading is wrong

Turn the power off first, then work out why. It's tempting to raise the current limit and watch what happens, but then a fault that was drawing a few milliamps can draw enough to damage the chip. Write down the reading you got next to the one you expected, because those two numbers are the first thing anyone helping you will ask for.

Two things to watch when you measure with an oscilloscope:

- A bench oscilloscope's probe ground is connected to mains earth through the scope's 3-wire power cord, so clipping the ground lead to a node that isn't at ground shorts that node to ground (Horowitz and Hill, Appendix O, §O.1.6). Clip it only to GND.
- The 6-inch ground lead of the standard 10× probe makes fast edges ring on the screen even though they don't ring in your circuit. A short ground tip that touches ground right beside the signal gives a much cleaner picture (§12.2).

Check yourself

1. You power the kit alone, running the factory test project, and it draws 450 mA from 5 V. What do you do next?

   Answer: A1, in Answers at the end.

2. Your Pmod has a 10 Ω current spy in its supply. Why can't it tell you how much current the chip draws?

   Answer: A2, in Answers at the end.

3. A friend plugs a laptop-powered USB-UART dongle into your Pmod while the kit is still unplugged. What can happen, and what on your board limits it?

   Answer: A3, in Answers at the end.

4. You want to see the ripple on your board's 3.3 V rail, so you clip the scope probe's tip to ground and its ground lead to 3.3 V. What happens?

   Answer: A4, in Answers at the end.

Next: Lab 04: Protect and plan.

Answers

A1. Check yourself

Turn it off. Tiny Tapeout's guide expects about 180 to 200 mA for that project, and 450 mA is more than twice that. None of your board is connected yet, so the fault is in the kit. Write the reading down and find out why before you go any further.

A2. Check yourself

The chip's rails, `VCCD`, `VDD_IO` and `VDDA`, stay on the breakout, and the only power pins on the DF12 connectors between the breakout and the demo board are +3V3, +5V and GND. Your Pmod's supply feeds only your Pmod. The chip's current is part of the kit's total at its USB input, and you can measure a single rail only after rework on the breakout.

A3. Check yourself

On the board as Lab 03 leaves it, nothing sits between the dongle and the chip, so the dongle's TX drives a `ui_in` pin of an unpowered chip. Current flows through the input's clamp diode into the chip's supply and partly powers it, and enough of that current can cause latch-up. On your board after Lab 04, the TX drives U1′s input instead. U1 is unpowered too, and passes at most 10 µA (its Ioff) on to `ui_in[0]`, a tenth of SkyWater's 100 µA limit. R4, the pull-up on U1′s input, passes up to 33 µA from the TX into the kit's +3V3. Connecting the dongle only after the kit is powered keeps both from flowing.

A4. Check yourself

The ground lead is connected to mains earth through the scope, so you short the 3.3 V rail to ground. The tip goes on the rail and the ground lead on GND.

19. Lab 04: Protect and plan

In this lab you make your Lab 03 board safe to plug into a Tiny Tapeout kit, and you write a plan for bringing it up with the kit. A kit has only one chip, so a mistake on your board that damages one of the chip's pins is expensive. On the board, you'll put a buffer and a resistor on each UART line, an ESD array on the dongle header and a current-sense point in the supply to your switches. The plan is a file in your repository, `BRINGUP.md`. It says what drives each line that leaves the board and what happens when one side is powered and the other isn't, it lists the checks you make before anything is powered, and it gives the reading you expect at every step.

The lab's files are in `labs/lab-04-protect-and-plan/` in the course repo.

Before you start

- Finish Lab 03, so that its check passes on your own repository.
- Read the pages on row 7 of the course path: Protecting the chip's I/O, Supplies, back-powering and ESD, Measuring current and Bringing up a chip. The numbers that this lab asks you to work out come from those pages.

The lab takes about 2 hours, roughly 20 minutes for Part 1, 40 for Part 2, 15 for each of Parts 3 and 4, and 30 for the bring-up plan. Those times are our estimates. We did time the check itself, which took 18 seconds in our run with the container limited to one CPU.

Copy the bring-up plan's template into your repository and run the Lab 04 check on the board as Lab 03 left it:

```
cd ~/simproc-pmod
cp ~/pcb101/labs/lab-04-protect-and-plan/BRINGUP.md .
git add BRINGUP.md
~/pcb101/labs/lab-04-protect-and-plan/check.sh
```

The check runs everything that Lab 03's check runs, then the rows that Lab 04 adds, then the check of the bring-up plan. On our Lab 03 board the board check still passes, and five Pmod rows fail:

```
   FAIL  ui[0]         J1 pin 1 reaches J4 pin 2 (the dongle's TXD) directly, with
no buffer
                    a resistor big enough for SkyWater's 100 µA (27 kΩ) is more than
the 15.4 kΩ that still gives a valid high against the RP2350's 36 kΩ pull-down
   FAIL  uo[0]         J3 pin 1 reaches J4 pin 3 (the dongle's RXD) directly, with
no buffer
                   a resistor big enough for SkyWater's 100 µA from a CH340's 16.5 kΩ
RX pull-up (10.5 kΩ) is more than the 5.28 kΩ that still gives a valid low at that RX
...
  FAIL  ESD J4.2     /UART_RX (the dongle's TXD) has no ESD part to GND on it
  FAIL  ESD J4.3     /UART_TX (the dongle's RXD) has no ESD part to GND on it
  FAIL  sense       no current-sense resistor between +3V3 and a net of the board's own
...
FAIL  pmod.kicad_pro: board check PASS; 12 of 17 Pmod checks ok; bring-up plan FAIL
```

The six rows that check the bring-up plan fail as well, although the excerpt above leaves them out. The template still says TODO in every section, and its Before power and Steps sections have one item each, where the check wants at least two and at least four. In Parts 2 to 4 you fix the five Pmod rows, and in Parts 1 and 5 you fill in the plan.

Part 1: Every line that leaves the board

For each line between your board and something else, what drives it, what does it drive, and what happens when one side is powered and the other isn't?

1. Fill in the table under "Lines that leave the board" in `BRINGUP.md`. It has a row each for `ui[0]`, `uo[0]`, `uio[0..7]` and the 3.3 V that your board takes from the sockets. Pins, Pmods and planning a pinout lists what else the demo board connects to each pin. Part 2 changes what's on the two UART lines, and its last step has you rewrite their rows.
2. Start with the worst case, which is the `ui[0]` row with the kit switched off. The dongle is plugged into a laptop, and its TXD idles high at 3.3 V into `ui[0]`. On the board as Lab 03 left it, what limits the current that flows into the chip?

   Answer: A1, in Answers at the end.
3. Your board drives `ui[0]`, and two things on the demo board can drive it too. Under Settings, write which firmware mode and which DIP-switch positions your board needs, and why.

   Answer: A2, in Answers at the end.

You should see the check's `Lines that leave the board` row go to `ok` once no cell is empty or says TODO.

Part 2: A buffer on each UART line

What keeps the current into an unpowered chip under SkyWater's 100 µA, and still lets the UART work?

1. SkyWater's I/O guide allows a pin to be driven above its supply only if the current into it is limited to 100 µA (Protecting the chip's I/O quotes it). Work out the smallest series resistor that keeps the current from a 3.3 V dongle under 100 µA when the chip's `VDD_IO` is near 0 V, $R \geq (3.3\ \text{V} - 0.6\ \text{V})/100\ \mu\text{A}$, and write it down.
2. The same resistor has to pass the UART's high. In `ASIC_MANUAL_INPUTS`, the demo board's firmware turns on the RP2350's pull-down on GPIO17, which is `ui_in[0]`, and the RP2350's datasheet gives that pull-down as 36 to 113 kΩ at 3.3 V. The chip's `ui_in` pads are CMOS inputs, and SkyWater guarantees a high only at 0.7 × 3.3 V = 2.31 V or more. Write down the high you expect at `ui_in[0]` through your resistor from step 1 against the strongest pull-down, 36 kΩ. Then work out the largest resistor that still gives 2.31 V.

   Answer: A3, in Answers at the end.
3. The `uo[0]` row of that run gives a different pair, 10.5 kΩ and 5.28 kΩ. Where do they come from?

   Answer: A4, in Answers at the end.
4. Each line needs a buffer that passes almost no current while the kit is off and drives the line while it's on. Ours is TI's SN74LVC2G17 (U1), a dual Schmitt-trigger buffer, powered from the Pmod's +3V3 so that it's off whenever the kit is (U1's supply). TI specifies it for partial power down, and with its supply at 0 V a pin passes at most 10 µA, which TI calls Ioff. To add it, press A and filter for `74LVC2G17`. KiCad's symbol is `74xGxx:74LVC2G17`, described as "Dual Buffer, Schmitt Triggered, Low-Voltage CMOS". It doesn't name a footprint, so give it `Package_TO_SOT_SMD:SOT-23-6`, which is TI's DBV package.

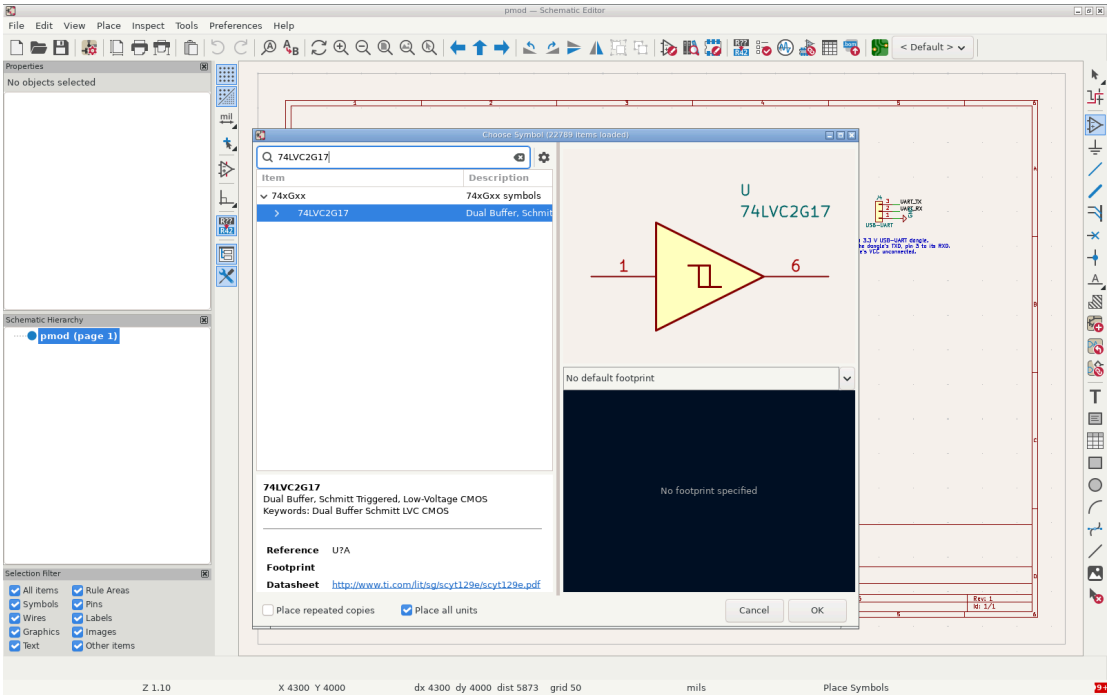


Figure 19.1: The Choose Symbol dialog filtered to 74LVC2G17

Choose Symbol in KiCad 10.0.5, filtered to `74LVC2G17`. The preview shows unit A, the buffer from pin 1 to pin 6, and the box under it says No default footprint. Unit B is the buffer from pin 3 to pin 4, and unit C holds GND (pin 2) and VCC (pin 5).

Leave Place all units ticked, and place all three. The pins have numbers but no names, so the wiring below goes by number, with the names from TI's pin table in brackets.

5. On your Lab 03 board, J4 pin 2 and J1 pin 1 are both on `UART_RX`, and J4 pin 3 and J3 pin 1 are both on `UART_TX`. Split each net, keep its name on the Pmod side (J1 or J3), give the J4 side a net of its own (ours are `DONGLE_TXD` and `DONGLE_RXD`), and wire each line through the buffer, U1, in the direction its signal runs:

   - `ui[0]`, into the chip: J4 pin 2 to U1 pin 1 (1A), and U1 pin 6 (1Y, our net `RX_BUF`) through R1 (1 kΩ) to J1 pin 1.
   - `uo[0]`, out of the chip: J3 pin 1 to U1 pin 3 (2A), and U1 pin 4 (2Y, our net `TX_BUF`) through R2 (1 kΩ) to J4 pin 3.
   - U1 pin 5 (VCC) to +3V3 and pin 2 (GND) to GND, with C1 (100 nF) from +3V3 to GND.
   - R4 (100 kΩ) from `DONGLE_TXD` to +3V3.

   R1, R2 and R4 (our references; R3 is the sense resistor in Part 4) are `Device:R` with the footprint `Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder`, and C1 is `Device:C` with `Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder`. Why does U1′s first input need R4, and why does R4 pull up rather than down?

   Answer: A5, in Answers at the end.

   With U1 driving both lines, what are R1 and R2 for?

   Answer: A6, in Answers at the end.

6. Update the board (F8), place the new parts near J4, and route. Put one of C1′s pads within 3 mm of U1′s pin 5, measured centre to centre, or both UART rows fail; ours is 2.24 mm away. On our board one line had to cross another near U1, so `DONGLE_RXD` goes from R2 through a via and along the bottom layer to J4 pin 3. If DRC reports `starved_thermal` on a GND pad, look for a piece of ground fill that your new tracks have cut off, and join it to the plane on the bottom layer with a GND via. Ours had two. Write down the high you expect at `ui[0]` through U1 and R1 against the 36 kΩ pull-down, then run the check. You should see both UART rows pass. Ours read:

```
  ok    ui[0]       J4 pin 2 (the dongle's TXD) drives J1 pin 1 through U1 (74LVC2G17,
1A pin 1 to 1Y pin 6) and R1 (1k)
                      U1: Ioff-rated, VCC (pin 5) on +3V3, GND (pin 2) on GND
                       R1 = 1 kΩ (330 Ω to 8.2 kΩ): 3.3 V / 1 kΩ = 3.3 mA against a
pin driving the other way; 3.3 V × 36 kΩ / (36 kΩ + 1 kΩ) = 3.21 V high against a
36 kΩ pull-down
                       decoupling: C1 (100n) from +3V3 to GND, a pad 2.24 mm from U1
pin 5
                      pull: R4 (100k) from U1's input (/DONGLE_TXD) up to +3V3
  ok    uo[0]       J3 pin 1 drives J4 pin 3 (the dongle's RXD) through U1 (74LVC2G17,
2A pin 3 to 2Y pin 4) and R2 (1k)
                      U1: Ioff-rated, VCC (pin 5) on +3V3, GND (pin 2) on GND
                     R2 = 1 kΩ (1 kΩ to 5.1 kΩ): (3.3 V - 0.6 V) / 1 kΩ = 2.7 mA into
an unpowered dongle; 3.3 V × 1 kΩ / (1 kΩ + 16.5 kΩ) = 0.19 V low against a 16.5 kΩ
pull-up
                       decoupling: C1 (100n) from +3V3 to GND, a pad 2.24 mm from U1
pin 5
```

7. Rewrite the `ui[0]` and `uo[0]` rows of your plan's table for the new circuit. With the kit off and the dongle powered, how much current can reach the chip now?

   Answer: A7, in Answers at the end.

Part 3: ESD on the dongle header

Where does static get onto your board, and where does the protection have to sit?

1. Add a two-line ESD array to the schematic. Ours is `Power_Protection:SP0502BAHT`, which KiCad's library describes as "TVS Diode Array, 5.5V Standoff, 2 Channels, SOT-23 package". The symbol already names its footprint, `Package_TO_SOT_SMD:SOT-23`, whose leads come out beside the body, where you can reach them with an iron.

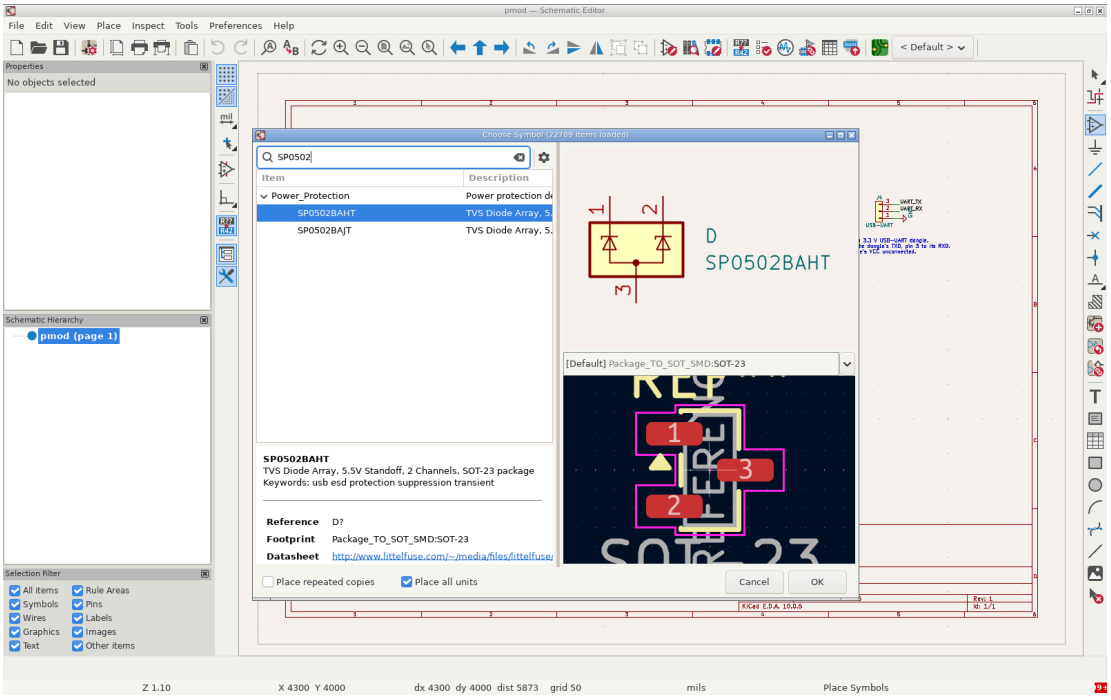


Figure 19.2: The Choose Symbol dialog filtered to SP0502

Choose Symbol in KiCad 10.0.5, filtered to `SP0502`. Pins 1 and 2 are the two protected lines and pin 3 goes to GND.

2. Connect its pins 1 and 2 to the header's pins 2 and 3 (`DONGLE_TXD` and `DONGLE_RXD`, on the header side of U1 and R2), and pin 3 to GND. Why there, rather than at the Pmod connectors?

   Answer: A8, in Answers at the end.

3. Place the array next to the header with a via from pin 3 to the ground plane, route it, and run the check. You should see:

```
ok    ESD J4.2     /DONGLE_TXD: D1 (SP0502BAHT, pin 1) to GND
ok    ESD J4.3     /DONGLE_RXD: D1 (SP0502BAHT, pin 2) to GND
```

The check counts a part as ESD protection if it has one pin on the line and another on GND, and either its reference starts with D or it comes from KiCad's `Power_Protection` library.

Part 4: A current-sense point

What should your board draw from the demo board's 3.3 V, and how will you read it?

1. Give your switches a supply net of their own. Put a 10 Ω resistor, with the same symbol and footprint as R1, from the sockets' `+3V3` to a new net (ours is `+3V3_PMOD`), and move the common side of the switches onto that net. Leave U1 and C1 on `+3V3`, the net of the sockets' pins 6 and 12. The check looks for U1′s VCC there, and the 10 Ω then carries only the switches' current. Put a test point on each side of the resistor, using the symbol `Connector:TestPoint` with the footprint `TestPoint:TestPoint_Pad_D1.5mm`.
2. Each switch that's on draws 3.3 V / 10 kΩ through its pull-down. Work out what your switches draw, set the way you'll use them, and the voltage that this current puts across the 10 Ω, and write both numbers in your plan.

   Answer: A9, in Answers at the end.

3. Can this resistor tell you what the chip draws?

   Answer: A10, in Answers at the end.

4. Update the board, place the resistor and the test points, route, and run the check. You should see:

```
ok    sense        R3 (10) from +3V3 to /+3V3_PMOD, test points TP1 and TP2
```

Part 5: The bring-up plan

What do you check before power, and what should you read at each step?

1. Under "Before power", list the checks you'd make with a multimeter on your board on its own, with nothing powered:

   - the resistance from +3V3 to GND, with every switch off and again with every switch on;
   - the resistance across R1 and across R2;
   - the resistance of R4, from J4 pin 2 to the test point on +3V3 (our TP1);
   - the resistance from each of the header's two signal pins to GND.

   Give each check an `Expect:` with its value. With every switch on, ours is 10 Ω + 10 kΩ / 8 = 1.26 kΩ. With every switch off, the reading climbs for a moment while the meter charges C1, and then settles above 1 MΩ. Why do you measure R1 and R2 at their own pads, rather than from one header to another?

   Answer: A11, in Answers at the end.

2. Fill in the rest of Settings: the current limit for your supplies (with its unit), your board's switches and the clock. The firmware mode and the DIP switches are there already from Part 1. SimProc's bit period is `clk_per_bit` = `uio` × 4 clocks, so with a 10 MHz clock and the switches set to 22, the UART runs at 10 MHz / 88 = 113,636 baud.
3. Under "Steps", write the bring-up in order: your board alone on a bench supply, then the kit's power, the clock, reset and project select, and the first functional test. Give each step an `Expect:` with a value and its unit, or with a byte in hex. On the bench nothing drives U1′s second input, and TI's datasheet says that "All unused inputs of the device must be held at VCC or GND", so put 10 kΩ from J3 pin 1 to

J3 pin 5 (GND) for the bench steps, and take it off before the board goes into the kit. With that resistor in place you can test both buffers on the bench: expect 3.3 V at J1 pin 1, because R4 holds U1′s first input high, and 0 V at J4 pin 3. With the kit on, J1 pin 1 should read 3.21 V to 3.3 V, where 3.21 V is the high you worked out in Part 2 against the strongest pull-down.

You can take two more values from DD104′s SimProc in one page. After reset `uo_out` reads 0x03, because TX idles high and `halt` is high, so the demo board lights segments A and B. When you send SimProc `PING`, the three bytes F0 00 00, it replies 0xAA.

4. Under “When a reading is wrong”, say what you do first and what each wrong reading points to. Also write down the order for unplugging, in which the dongle comes off J4 before the kit is switched off.
5. Run the check. You should see the bring-up plan pass:

```
ok    TODO                          none left
ok    Lines that leave the board    4 rows, every cell filled in
ok    Settings                      5 items; a current limit with its unit
ok    Before power                 5 items, each with an expected value and its unit
ok    Steps                        8 items, each with an expected value and its unit
ok    When a reading is wrong       7 lines
```

Your counts may differ from ours. The check can’t tell whether your expected values are right, so someone else should read your plan before any kit is powered with your board plugged in.

Finishing

You’re done when `~/pcb101/labs/lab-04-protect-and-plan/check.sh`, run in your own repository, ends with `PASS  pmod.kicad_pro: board check PASS; 17 of 17 Pmod checks ok; bring-up plan PASS`, and the files it checked are committed and pushed to GitHub.

Questions:

1. You power your board alone from a bench supply, with three switches on and the 10 kΩ on J3. What current does the supply show, and what voltage do you read between your two test points?

   Answer: A12, in Answers at the end.

2. Why does the dongle come off J4 before the kit is switched off?

   Answer: A13, in Answers at the end.

Submit: nothing for this lab. Keep `BRINGUP.md` in your repository, because the capstone asks for a bring-up plan, and for Brief D you start from this one.

Going further (optional)

- Give U1′s second input a pull-down of its own, from `UART_TX` to GND, so that the bench steps no longer need the 10 kΩ on J3. The check doesn’t ask for a pull on a `uo` line. Work out what a 100 kΩ pull-down takes from SimProc’s TX while the line is high.
- Change R1 to 100 Ω, or R2 to 10 kΩ, run the check, and read the reason it gives. Then put the 1 kΩ back.

Next on the Course path: read Fabrication outputs, Outputs in CI and KiCad files in git, then do Lab 05: Fab outputs in CI.

Answers

A1. Part 1: Every line that leaves the board

Nothing on your board does. The current flows from the dongle’s TXD through the pad’s upper protection diode into `VDD_IO`, and from there onto the demo board’s +3V3, and it’s limited only by the dongle’s own output stage and the diode’s drop. Horowitz and Hill warn that momentary currents of more than 20 to 50 mA through those diodes can damage the part, and in some cases latch it up. Supplies, back-powering and ESD draws this path in a figure.

A2. Part 1: Every line that leaves the board

In the firmware’s `ASIC_RP_CONTROL` mode, which the example configuration sets as the default, the RP2350 drives the `ui_in` pins, so SimProc’s section of `config.ini` has to select `ASIC_MANUAL_INPUTS`. The

demo board's DIP switch 1 connects `ui_in[0]` to +3V3 through 1 kΩ, so it has to be off. After Part 2 your board drives the line through R1, also 1 kΩ, so with the switch on a low would sit at 1.65 V, and the chip reads a low only at 0.99 V or less (Contention).

A3. Part 2: A buffer on each UART line

3.3 V × 36 kΩ/(36 kΩ + 27 kΩ) = 1.89 V, below 2.31 V. The high reaches 2.31 V only while $R \leq 15.4$ kΩ, less than the 27 kΩ that the current needs, so on this kit no series resistor does both. The `ui[0]` row of your first run of the check gives the same two numbers.

A4. Part 2: A buffer on each UART line

From the dongle's RX pull-up. Of the five dongle chips we looked at (FT232R, FT231X, CP2102, CP2102N and CH340), the CH340′s is the strongest. Its datasheet gives up to 200 µA at 3.3 V, which is 16.5 kΩ if that current flows at 0 V. Into an unpowered `uo_out[0]`, it takes 10.5 kΩ in series, for 27 kΩ in all, to keep that pull-up's current under 100 µA. With the chip powered, SimProc's low reaches the RX through the resistor and stays under the CH340′s input-low limit of 0.8 V only while $R \leq 5.28$ kΩ. Sizing a series resistor works through both lines.

A5. Part 2: A buffer on each UART line

With no dongle on J4, nothing drives U1′s pin 1, and a floating input on a powered buffer can toggle and send SimProc garbage. R4 pulls up because a UART line idles high, and a receiver reads a line held low as a break (Inputs that nobody drives).

A6. Part 2: A buffer on each UART line

R1 limits the current to 3.3 mA if the RP2350 fights U1 on `ui_in[0]`, and to 1.65 mA if DIP switch 1 does, through the switch's own 1 kΩ and R1 in series (Part 1, step 3). With the kit on and the dongle unpowered, U1′s high pushes current through the clamp diode on the dongle's RX, and R2 limits it to 2.7 mA. Each also slows the edges at the pin it drives, to $2.2RC$ = 22 ns with 1 kΩ and an assumed 10 pF. That's short next to the 8.7 µs of a bit at 115,200 baud.

The check accepts R1 from 330 Ω to 8.2 kΩ. At 330 Ω a fight passes 10 mA. The top of the window is below step 2′s 15.4 kΩ because of erratum E9 on the RP2350′s A2 stepping (RP2350 datasheet). It accepts R2 from 1 kΩ to 5.1 kΩ, the E24 value just under step 3′s 5.28 kΩ. Neither the RP2350′s absolute maximum ratings nor any of the dongle datasheets we read gives a current limit for a pin, so the 10 mA and the 2.7 mA are limits we chose. The resistor after each buffer explains both windows.

A7. Part 2: A buffer on each UART line

At most 10 µA into `ui_in[0]`, U1′s Ioff, because U1 is off with the kit. On `uo[0]` the dongle's RX pull-up pushes into U1′s output (2Y), and `uo_out[0]` is on U1′s input (2A), so none of that current reaches the chip. R4 adds up to 3.3 V / 100 kΩ = 33 µA from the dongle's TXD into the kit's +3V3, a supply rail rather than a pin.

The "Kit on, dongle unpowered" column changes too. An unpowered dongle may pull `DONGLE_TXD` down to about 0.6 V through the clamp diode on its TXD. R4 is too weak to stop it, so U1 holds `ui[0]` low. That's our inference, not a measurement, and it's why your plan should say to keep the dongle powered whenever it's plugged into J4.

A8. Part 3: ESD on the dongle header

Static arrives on the dongle's cable, so the clamp should be the first thing it meets, close to the header and with a short path to ground. U1 sits between the clamp and the chip on both lines, so whatever gets past the clamp reaches U1 first. The demo board's TVS diodes are on the USB data lines and the analog lines, none of which your board uses, so the array on J4 is the only TVS protection your board's lines get, and it covers just the two UART lines. ESD gives Horowitz and Hill's estimate of the energy that a discharge delivers to an input with and without external clamp diodes and series resistors.

A9. Part 4: A current-sense point

Each switch that's on draws 0.33 mA. With three on, the switches draw 0.99 mA, which drops 9.9 mV across 10 Ω, and with all eight on they draw 2.64 mA and drop 26.4 mV. A multimeter on its millivolt range can read either.

A10. Part 4: A current-sense point

No. The chip's supplies stay on the breakout, so the current through this resistor is only what your switches and their pull-downs draw. With your board on the demo board and no rework on the breakout, you can read the chip's current only as part of the whole kit's, at the USB input, and Tiny Tapeout's ETR guide puts the kit's current at 180 to 200 mA with the factory test project running. Measuring current explains why, and describes a board in the demo board's place that can measure the breakout's total current on +3V3.

A11. Part 5: The bring-up plan

From header to header, both lines go through U1, so the reading tells you nothing about the resistor. Measure from J1 pin 1 to R1′s other pad and from J4 pin 3 to R2′s other pad, and expect 1.00 kΩ each.

A12. Finishing

The supply shows 0.99 mA, plus at most 10 µA of supply current for U1 (TI's ICC), and the meter across the test points reads 9.9 mV, as you worked out in Part 4, step 2. U1 takes its supply from +3V3 ahead of R3, so its current doesn't flow through R3.

A13. Finishing

So that the dongle never feeds the unpowered kit. If the kit goes off first, the dongle's TXD keeps driving U1′s unpowered input and R4 (Part 2, step 7), and R4′s 33 µA lifts the kit's +3V3 by an amount set by that rail's load. U1 keeps the current into the chip's pins under SkyWater's 100 µA in either order.



5. Outputs and CI

20 Fabrication outputs 133
20 One Gerber per layer 133
20 Inside a Gerber file 135
20 The board's plot settings 135
20 The drill file 136
20 Files for assembly: BOM and position 138
20 One-file formats: IPC-2581, ODB++ and STEP . 138
20 Dates in the files 139
20 Check yourself 139
20 Answers 139

21 Outputs in CI 141
21 Checks that fail the run 142
21 Pinning the image 142
21 The user and the library tables 143
21 Fonts and 3D models 143
21 Outputs that repeat from run to run 144
21 Jobsets 144
21 KiBot 145
21 Check yourself 145
21 Answers 145

22 KiCad files in git 147
22 What's in a project folder 147
22 A `.gitignore` for a board 148
22 Ignoring a file that git already tracks 149
22 A save with no edits 149
22 Reviewing a change to a board 149
22 Libraries in the repository 150
22 Check yourself 150
22 Answers 151

23 Lab 05: Fab outputs in CI 153
23 Before you start 153
23 Part 1: Add the workflow 153
23 Part 2: Keep KiCad's own files out of git 154
23 Part 3: Finish the fabrication files and review images 155
23 Part 4: Look at what the fab will get 156
23 Part 5: Add KiBot's outputs 157
23 Part 6: Break it on purpose 160
23 Finishing 161
23 Going further (optional) 161
23 Answers 162

20. Fabrication outputs

This page answers: *what does a fabricator actually need from you to make a board, and what is in each of those files?*

To have a board made, you normally send the board house a set of plain files that any CAM (computer-aided manufacturing) program can read, rather than your `.kicad_pcb`. The set has one Gerber file for each layer the board house has to make and a drill file that says where the holes go, and if the board house also assembles the board, it needs a bill of materials (BOM) and a position file as well. KiCad writes all of them, either from its editors' menus or from `kicad-cli`, and in Lab 05 you set up a CI workflow that writes them for you on every push. This page goes through each of them on a real 2-layer board, so that when you open your own outputs you know what you're looking at.

The board is Tiny Tapeout's DIP dock (`TinyTapeout/breakout-pcb` at 91a2bf1, Apache-2.0). It's the 2-layer board that Tiny Tapeout's DIP carriers plug into, and of their boards it's the closest to the Pmod you draw in Lab 03. According to its Gerber job file, the dock is 85.05 × 85.05 mm and 1.561 mm thick, with 2 copper layers, and the PCB Editor counts 370 pads, 111 vias and 535 track segments on it. Every file name, size and line quoted below comes from exports we ran with `kicad-cli` 10.0.5 in the course's pinned image, on the evening of November 16, 2026, in Toronto. The files record the time in UTC, and in UTC it was already November 17.

One Gerber per layer

A Gerber file describes one layer as flat 2D shapes (lines with a width, pads and filled regions), placed in millimetres (or inches) from an origin. There are no parts or footprints in a Gerber file, although the newer version of the format can label each shape with the net it belongs to, and each pad with the part and pin it belongs to. The fabricator lines the layers up on top of each other, so every file has to use the same origin and the same orientation.

For the DIP dock, the board's own Gerber settings produce ten files:

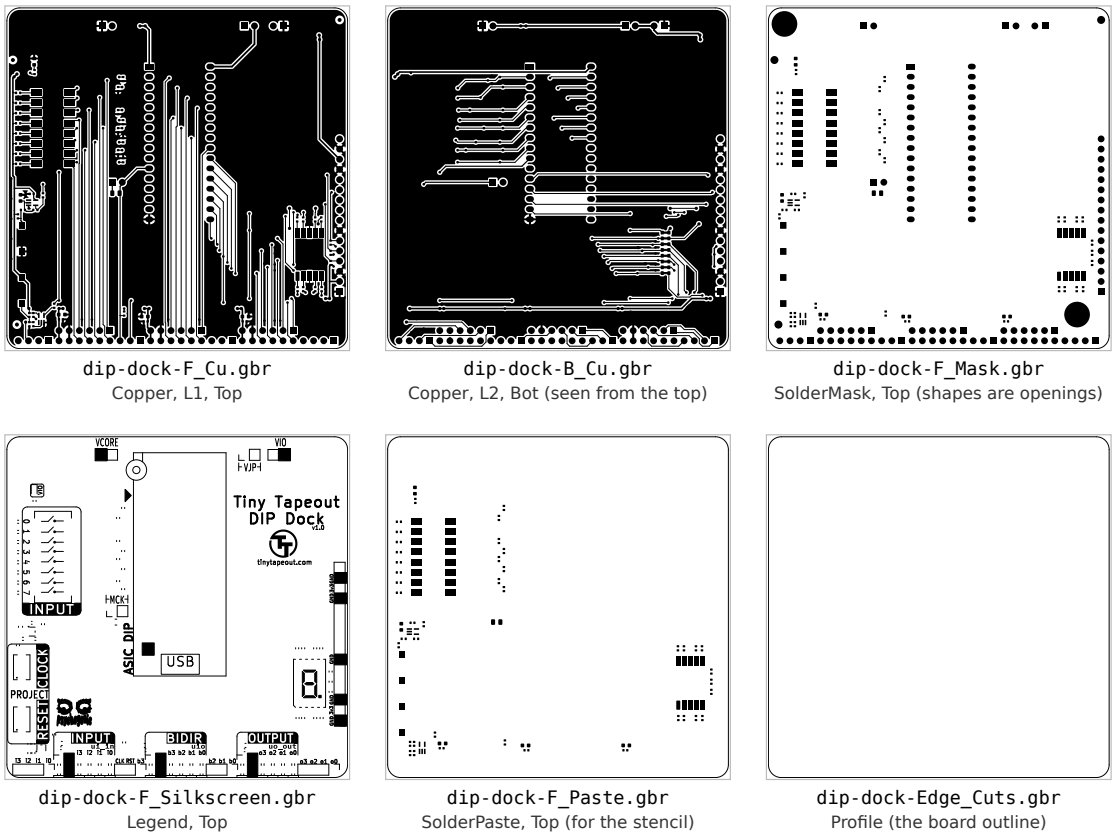


Figure 20.1: KiCad's SVG plots of six of the DIP dock's layers, each labelled with the Gerber file that holds it

Six of the DIP dock's ten Gerber layers, plotted by `kicad-cli pcb export svg` from the same board, in black on white. Under each panel are the name of the Gerber file and the file function that KiCad records for it. Holes aren't drawn, because they go in the drill file and none of the Gerbers holds them. Made by `scripts/figures/dip_dock_layers.py`.

File	What the fabricator does with it
`dip-dock-F_Cu.gbr`, `dip-dock-B_Cu.gbr`	Etches the copper on the top and bottom: tracks, pads and the ground pour.
`dip-dock-F_Mask.gbr`, `dip-dock-B_Mask.gbr`	Makes the solder mask, the coloured coating. These files are *negative*: each shape is an opening in the mask, where the copper stays bare so that it can be soldered.
`dip-dock-F_Silkscreen.gbr`, `dip-dock-B_Silkscreen.gbr`	Prints the legend: reference designators, outlines, labels and logos.
`dip-dock-F_Paste.gbr`, `dip-dock-B_Paste.gbr`	Cuts the stencil that puts solder paste on the surface-mount pads. Only an assembled order uses these.
`dip-dock-Edge_Cuts.gbr`	Routes the board outline, which here is a square with rounded corners.
`dip-dock-F_Fab.gbr`	Doesn't need it to make the bare board. It holds the assembly drawing from the front fabrication layer.

It's easy to read the mask layers backwards. In the F.Mask panel above, the black shapes are where there *won't* be any mask. So if a pad is missing from the mask layer, the pad ends up covered and can't be soldered, and if there's a mask opening with no pad under it, the opening exposes whatever copper is there, which is usually the ground pour.

Next to the Gerbers, KiCad writes a job file, `dip-dock-job.gbrjob`. It's a JSON file that lists every Gerber with its function and polarity, along with the board's size, number of layers and thickness.

Inside a Gerber file

A Gerber file is plain text, and its first few lines describe the file itself. These are the first ten lines of `dip-dock-F_Cu.gbr`:

```
G04 #@! TF.GenerationSoftware,KiCad,Pcbnew,10.0.5*
G04 #@! TF.CreationDate,2026-11-17T01:02:51+00:00*
G04 #@! TF.ProjectId,dip-dock,6469702d-646f-4636-9b2e-6b696361645f,1.0*
G04 #@! TF.SameCoordinates,PX3fe56c0PY5cfbb60*
G04 #@! TF.FileFunction,Copper,L1,Top*
G04 #@! TF.FilePolarity,Positive*
%FSLAX46Y46*%
G04 Gerber Fmt 4.6, Leading zero omitted, Abs format (unit mm)*
G04 Created by KiCad (PCBNEW 10.0.5) date 2026-11-17 01:02:51*
%MOMM*%
```

The `TF` lines are **file attributes**, which come from the extended version of the format, called Gerber X2. `TF.FileFunction,Copper,L1,Top` says that this is copper layer 1, the top layer, so a fabricator's software doesn't have to guess the layer from the file name. `TF.SameCoordinates` marks all the files of one export as sharing a coordinate system. `%MOMM*%` sets the unit to millimetres, and `%FSLAX46Y46*%` sets the number format, with four integer digits and six decimal digits.

The DIP dock's plot settings have "Use extended X2 format" turned off, so KiCad writes the attributes inside `G04` comments, which software that doesn't know X2 skips like any other comment. When we exported without the board's settings, `kicad-cli`'s defaults wrote the same attributes as X2 commands instead, for example `%TF.FileFunction,Copper,L1,Top*%`, so the information is the same either way.

The board's plot settings

The board file stores which layers become Gerbers, and with what options. In the PCB Editor you set them in the Plot dialog, under File → Plot:

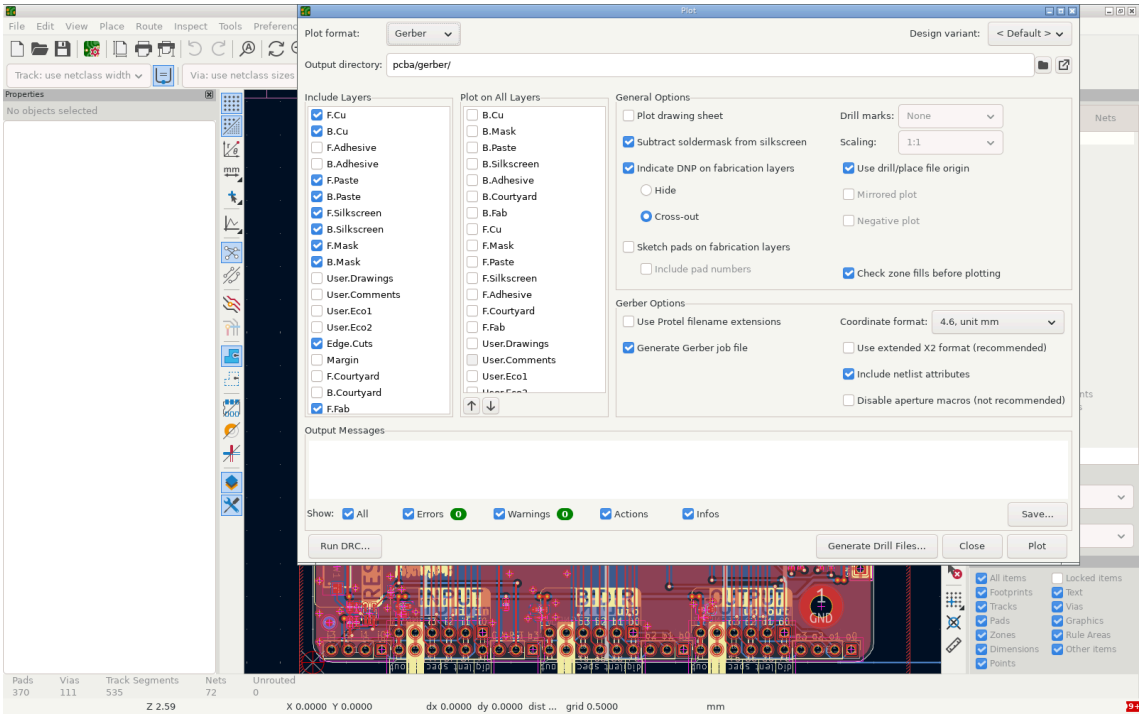


Figure 20.2: The Plot dialog in KiCad 10.0.5's PCB Editor, with the DIP dock open

The PCB Editor's Plot dialog (File → Plot) with the DIP dock open, in KiCad 10.0.5. Under "Include Layers" the designer ticked F.Cu and B.Cu, the paste, silkscreen and mask layers on both sides, Edge.Cuts and F.Fab. On the right, "Use drill/place file origin" is ticked and "Use extended X2 format (recommended)" isn't. "Generate Drill Files..." at the bottom opens a second dialog for the drill file.

`kicad-cli pcb export gerbers` uses those settings only when you pass `--board-plot-params`, and otherwise uses its own defaults. On the DIP dock, with `--board-plot-params`, it wrote the ten `.gbr` files above. Without it, it wrote a job file and 26 Gerbers, one for every layer the board has, including the courtyards, the margin and ten user layers such as `dip-dock-User_Eco1.gbr`. It also gave the main layers the old Protel extensions instead of `.gbr`, such as `dip-dock-F_Cu.gtl`, `dip-dock-B_Cu.gbl`, `dip-dock-F_Mask.gts` and `dip-dock-Edge_Cuts.gm1`. The layers the fab needs are in both sets, but with the second set a fabricator has to work out which 10 of the 26 Gerbers matter. The silkscreen is different too, because the board's settings subtract the solder mask from the silkscreen and `kicad-cli`'s defaults don't.

The simplest way to avoid both problems is to set up the Plot dialog once, save the board, and pass `--board-plot-params` to every `kicad-cli` export, whether you run it yourself or CI does. Then CI makes the same files that the Plot dialog makes in the GUI.

The drill file

The holes go in a separate drill file, which KiCad writes in the Excellon format unless you ask for Gerber (`--format gerber`). For the DIP dock, `kicad-cli pcb export drill` wrote one file, `dip-dock.drl`, and its header lists the drill sizes, which are called tools:

```
M48
; DRILL file KiCad 10.0.5 date 2026-11-17T01:03:09
; FORMAT={-:-/ absolute / metric / decimal}
; #@! TF.CreationDate,2026-11-17T01:03:09+00:00
; #@! TF.GenerationSoftware,Kicad,Pcbnew,10.0.5
; #@! TF.FileFunction,MixedPlating,1,2
FMAT,2
METRIC
; #@! TA.AperFunction,Plated,PTH,ViaDrill
T1C0.300
```

The header goes on to list six tools, all plated, in four sizes, because KiCad gives via holes and component holes of the same size a tool each: 0.3 mm (90 via holes and 4 component holes), 0.4 mm (21 via holes and 103 component holes), 1.0 mm (106 component holes) and 3.2 mm (2 component holes, the mounting holes). The 4 component holes at 0.3 mm and the 103 at 0.4 mm are stitching vias that the designer placed as footprints, which is why the file counts them as component holes. The 90 and 21 via holes add up to the 111 vias that the PCB Editor counts, which is a quick check that no via is missing from the file. `MixedPlating` means that the file is allowed to hold plated and unplated holes together; `--excellon-separate-th` writes them to separate files instead.

The drill file and the Gerbers line up only through their coordinates, so the drill file has to use the same origin as the Gerbers. In our first export of the DIP dock it used a different one. The board's plot settings tick "Use drill/place file origin", so its Gerbers are drawn from an origin that the designer placed at the board's bottom-left corner, but `kicad-cli pcb export drill` defaults to `--drill-origin absolute`, which is the page origin. When we opened the two together in GerbView, KiCad's Gerber viewer, the holes landed below the board and to the right of it:

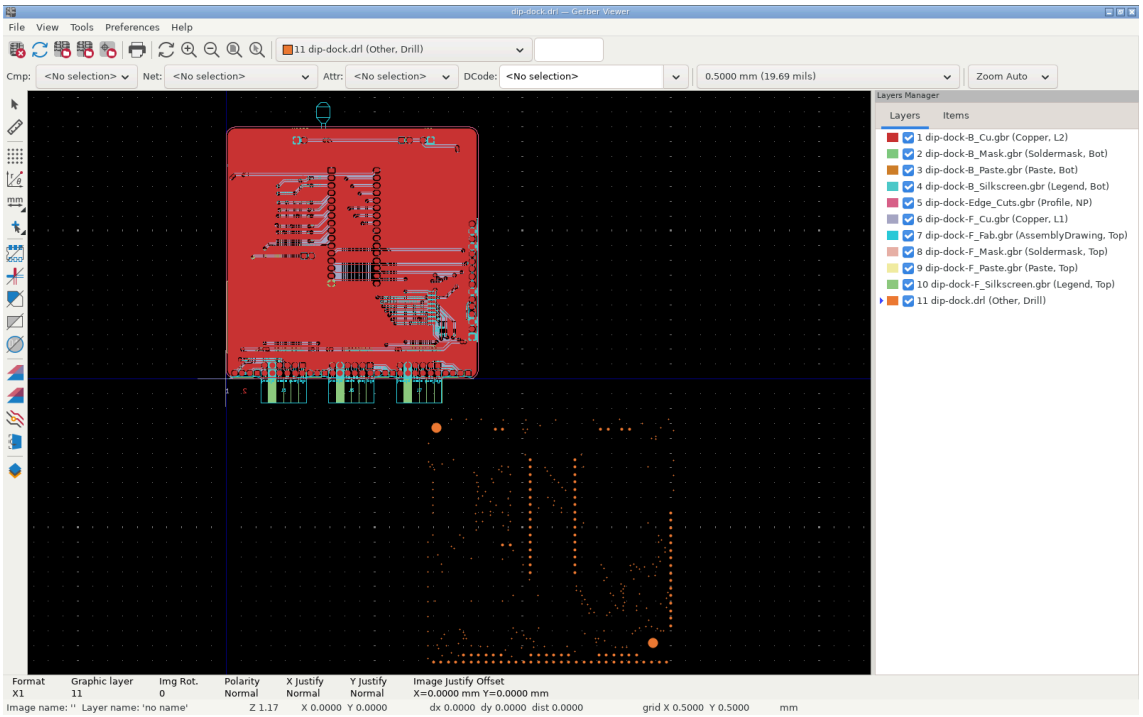


Figure 20.3: GerbView showing the DIP dock's Gerbers with the drill holes offset to the lower right of the board

The DIP dock's ten Gerbers and a drill file made with `kicad-cli`'s default drill origin, in GerbView (KiCad 10.0.5). The board sits above and to the right of the origin, and the orange holes from `dip-dock.drl` sit below and to the right of it. Both exports succeeded, and neither printed a warning about the origin.

When we exported the drill file again with `--drill-origin plot`, which uses the same origin as the plot, the holes sat in their pads:

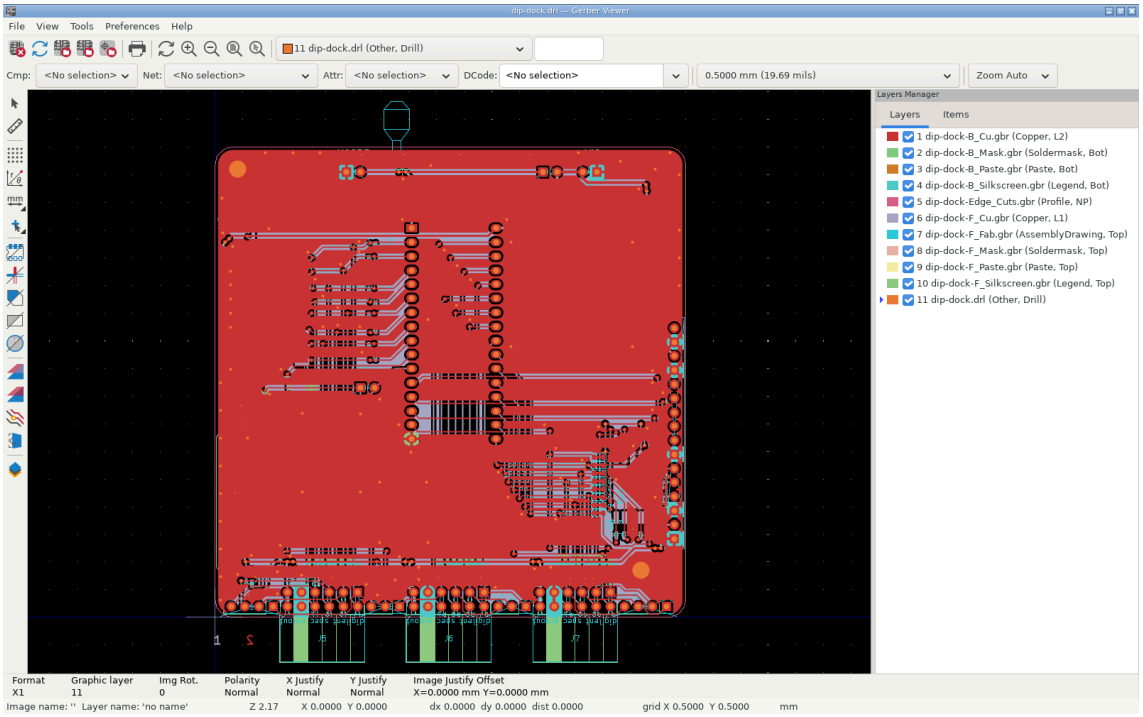


Figure 20.4: GerbView showing the same Gerbers with the drill holes centred in their pads

The same Gerbers with a drill file made with `--drill-origin plot`. The Layers Manager on the right lists all eleven files, each with the function that GerbView read from it, such as "Copper, L2", "Soldermask, Bot" and "Profile, NP". Both screenshots were made by `scripts/figures/shots/fab-outputs-dip-dock.sh`.

Opening the whole set in GerbView before you upload it takes a minute, and it shows you the files the fab will get rather than the board they were made from. Lab 05 has you do it with the files your CI built. Lab 05's check catches this offset as well, because it checks that every hole in the drill file lies inside the outline in the Edge.Cuts Gerber.

Files for assembly: BOM and position

Whoever assembles the board uses these two files, so they're for you if you solder the board yourself, and for the fab if it assembles the board for you.

The **BOM** comes from the schematic (`kicad-cli sch export bom`). With the default columns, `Refs`, `Value`, `Footprint`, `Qty` and `DNP`, the DIP dock's BOM has one row per part, 67 rows in all. One row, R15, is marked `DNP` (do not populate). R15 is a 0 Ω resistor in an 0805 footprint, and the board has room for it but doesn't fit it. If a BOM leaves out the `DNP` column, R15 looks like any other part on it, and an assembler can end up fitting a part that the designer meant to leave off. In Lab 01, Part 2 you list the DNP parts on the SKY130 breakout and compare your list with Tiny Tapeout's production BOM, which leaves them out.

The **position file** (`kicad-cli pcb export pos`), also called a pick-and-place or centroid file, gives each part's reference, value, package, X and Y position, rotation and side. The DIP dock's position file lists 80 parts, 72 on the top and 8 on the bottom (the solder jumpers JP1 to JP8). By default the command writes inches, in a fixed-width text format, so ask for `--format csv --units mm` if your assembler wants a spreadsheet in millimetres, and for `--use-drill-file-origin` if the Gerbers use that origin.

One-file formats: IPC-2581, ODB++ and STEP

Gerbers and a drill file are the usual set. IPC-2581 and ODB++ each put the whole board in one file, with its stackup, nets and parts, and `kicad-cli` writes both. For the DIP dock, `pcb export ipc2581` wrote a 5.1 MB XML file (IPC-2581 revision C, the default), and `pcb export odb` wrote ODB++ as a 292 kB zip of 113 entries. The ODB++ export also printed 88 warnings like the one below, because the board's stitching vias are footprints that share reference designators:

```
Warning: Component 'STITCH12' has an ambiguous designator after conversion; renamed to
'STITCH12_1' for ODB++ export.
```

`pcb export step` writes a 3D model of the board for mechanical work, such as checking that it fits a case. In the course image it wrote a 2.2 MB file that holds the board and one part, the DIP switch SW3, whose 3D model is in Tiny Tapeout's own library, `tinytapeout-kicad-libs`, which the course fetches next to the board. For each of 66 other parts it printed a message like this one:

```
Could not add 3D model for R29.
File not found: ${KICAD10_3DMODEL_DIR}/Resistor_SMD.3dshapes/R_0402_1005Metric.step
```

The paths in those messages point into KiCad's own library of 3D models, which the image doesn't include. A KiCad installed with its 3D models, such as the full Linux AppImage, fills those parts in.

Dates in the files

We exported the same board's Gerbers twice, 11 seconds apart. Every Gerber differed in exactly two lines, the `TF.CreationDate` attribute and the `Created by KiCad ... date` comment, and the job file differed in its `CreationDate`. The drill file has the date in its header too. Turning off X2 doesn't remove these lines, and when we exported with `--no-x2` they were still there.

So two exports of the same commit, made at different times, are never identical byte for byte. If you want CI to tell you whether a commit changed the fabrication files, compare them with the date lines taken out. Outputs in CI comes back to this.

Check yourself

1. In the F.Mask panel of the figure, the pads of the DIP socket are black. Is there solder mask on them?

   Answer: A1, in Answers at the end.

2. You open your Gerbers and drill file in GerbView, and every hole is shifted the same distance away from its pad. What happened, and what would you change?

   Answer: A2, in Answers at the end.

3. CI builds Gerbers from the same commit on Monday and on Tuesday, and their checksums differ. Did the board change?

   Answer: A3, in Answers at the end.

4. You're ordering bare boards and will solder them yourself. Which of the files on this page does the fabricator need?

   Answer: A4, in Answers at the end.

Next: Outputs in CI.

Answers

A1. Check yourself

No. The mask files are negative, so a shape in `dip-dock-F_Mask.gbr` is an opening in the mask. The black pads are where the mask is left off, so that the pads can be soldered.

A2. Check yourself

The Gerbers and the drill file were made with different origins, as in the first GerbView screenshot. If your Gerbers use the drill/place file origin (`--board-plot-params` with that box ticked), make the drill file with `--drill-origin plot`, or untick "Use drill/place file origin" so that both use the absolute origin.

A3. Check yourself

No. Both builds are of the same commit, so the board is the same. Every Gerber, the job file and the drill file carry the date of the export, and that's enough to change their checksums. Compare the files with the date lines removed before you conclude that anything else changed.

A4. Check yourself

The copper, mask, silkscreen and Edge.Cuts Gerbers and the drill file, plus the job file if their upload page reads it. The paste layers, the BOM and the position file are for assembly, and F.Fab is a drawing for people.

21. Outputs in CI

This page answers: *how can a computer check your board and build its fabrication files every time you push, and how do you know that it ran the same KiCad, set up the same way, as the runs that produced the numbers on the lab pages?*

Continuous integration (CI) means that every time you push, a fresh machine checks out your commit and runs a fixed list of commands on it. You met it in IC102, in the workflows that Tiny Tapeout's template runs on your project. For a board, the first commands are the checks you'd otherwise run by hand (ERC, DRC and schematic parity), and the rest build the fabrication outputs and some images that a reviewer can look at without opening KiCad. In Lab 05 you add a workflow like this to your board repository, then break the board on purpose and watch the run go red.

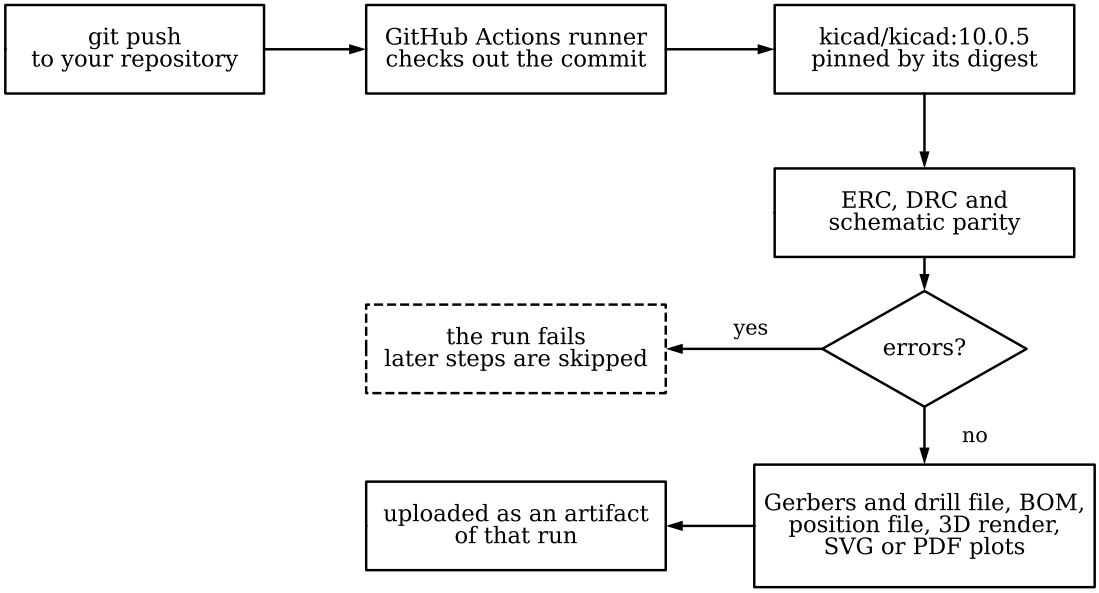


Figure 21.1: A flow chart of one CI run: a push, a runner that checks out the commit, the pinned KiCad image, the checks, and either a failed run or the outputs uploaded as an artifact

One CI run for a board. If ERC, DRC or schematic parity fails the board, that step fails, and GitHub Actions skips the steps after it that build the outputs, so a broken board never produces a set of Gerbers that looks ready to order. In Lab 05's workflow the upload step runs anyway, so a red run's artifact still holds the ERC and DRC reports. Made by `scripts/figures/ci_flow.py`.

Other KiCad projects already build their outputs in CI. wafer.space's chip-on-board repository, for example, has a GitHub Actions workflow that runs in the `kicad/kicad:10.0` container each time a release is tagged. It renders the top and bottom of each board with `kicad-cli pcb render`, plots the copper and silkscreen layers with `kicad-cli pcb export svg`, exports each schematic with `kicad-cli sch export pdf`, and commits the results back to the repository (`.github/workflows/release-docs.yml` and `.github/scripts/gen_docs.py` in `wafer-space/chip-on-board-wire-bonded-pcbs`). This course's own repository uses CI for checks instead. Its `labs.yml` workflow runs the board check on every lab's reference solution whenever something in `labs/` changes. It also runs once a week, to catch breakage that comes from outside the repository, such as a pinned Tiny Tapeout commit or KiCad image that stops being served.

Checks that fail the run

A CI step fails when its command exits with a status other than 0, so each check needs a command that exits with a nonzero status when, and only when, the board has the problem you're checking for. By default `kicad-cli` exits 0 even when it finds problems. On Tiny Tapeout's DIP dock (`breakout-pcb` at 91a2bf1), ERC found 42 violations and DRC found 156, and both exited 0. With `--exit-code-violations` it exits 5 if it reports anything, and the `--severity-...` options decide which severities it reports. Here are both checks on the DIP dock with those options, run in the course's pinned image. We've shortened the paths. We've also left out the line that says where each report was saved, and the font message that each DRC run printed, which Fonts and 3D models explains.

```
$ kicad-cli sch erc --severity-error --exit-code-violations -o erc.rpt dip-
dock.kicad_sch
Found 2 violations
exit status 5
$ kicad-cli pcb drc --severity-error --schematic-parity --exit-code-violations -o
drc.rpt dip-dock.kicad_pcb
Found 0 violations
Found 0 unconnected items
Found 0 schematic parity issues
exit status 0
```

ERC fails on two `power_pin_not_driven` errors, and DRC passes. With `--severity-all` in place of `--severity-error`, the same DRC command reports this:

```
Found 162 violations
Found 0 unconnected items
Found 3 schematic parity issues
exit status 5
```

The three parity issues were there in the first run too, but `--severity-error` left them out. In every project we checked, KiCad reported schematic parity problems as warnings, including `missing_footprint`, a footprint that is in the schematic and missing from the board. On the DIP dock all three are about the logo footprint `PSYLOGO01`, which is placed twice on the board and isn't in the schematic, so they're harmless. On your board the same filter would let a missing connector through with a green run.

So the labs' checks don't rely on `kicad-cli`'s own exit status. In every lab that checks a board, the check runs `labs/common/check_board.py` instead. It runs ERC and DRC with every severity, reads the JSON reports, and fails on any error, any unconnected item, and any parity item at all. On the DIP dock it took 28 s with the container limited to one CPU, and ended like this:

```
                errors  warnings  excluded
  ERC                2        40         0
  DRC                0       156         6
  unconnected        0         0         0
  parity             0         3         0
...
FAIL  erc.error: 2 (want 0)
FAIL  parity.warning: 3 (want 0)
```

It exits 1 on a FAIL, so a workflow step that runs it fails the run. The 156 DRC warnings don't fail it. They're `lib_footprint_issues` and `lib_footprint_mismatch` warnings, which compare the footprints on the board with the libraries installed wherever the check runs. Their count depends on which KiCad image runs the check and on which user runs it, and the next two sections take those in turn.

Pinning the image

In a Docker image name such as `kicad/kicad:10.0`, the part after the colon is a tag, and the publisher moves a tag from one image to another as new releases come out. On Docker Hub, `kicad/kicad:10.0` and `kicad/kicad:10.0.8` were both last updated on October 30, 2026, and both are 812 MB, because `10.0` currently points at 10.0.8. KiCad posted 10.0.5 on July 22, 10.0.6 on August 29, 10.0.7 on October 7 and 10.0.8 on October 30, so a workflow that names `10.0` gets a new KiCad every few weeks without anyone

committing anything. That doesn't matter much for pictures, but for checks it means a DRC count can change on a Monday morning for reasons that have nothing to do with your board.

A digest names one exact image, and unlike a tag it can't be moved to another. The course's checks name `kicad/kicad:10.0.5@sha256:182c8005cb77…` (in full in `labs/common/kicad-image.txt`), and `labs/common/kicad.sh` prints that digest before every check, so the log of each check records which KiCad produced it. The course pins 10.0.5 rather than the newer 10.0.8 so that the checks, KiBot and the course's 3D renders all run the same KiCad release (see KiBot below). We moved the pin from 10.0.8 to 10.0.5 on November 16, 2026, after running the checks in both releases. The course's own boards gave the same counts in both, but Tiny Tapeout's boards gave different warning counts, and a broken copy of one of KiCad's demo boards gave a different number of DRC errors. Getting started lists the differences. Moving the pin meant running every check again and copying the new counts onto every page, and a move to a newer release will take the same work.

The user and the library tables

Pinning the image isn't quite enough, because KiCad's warnings also depend on which user runs it. Many warnings compare the parts in your design with the libraries KiCad knows about, and KiCad keeps the list of those libraries, its library tables, in the user's settings folder. A container that runs as root, or as your own uid with an empty home folder, has no library tables at all, so every standard library looks missing. On the SKY130 breakout, the course's setup gives 97 ERC warnings and 120 DRC warnings. Run as root in the image, `kicad-cli` gives 233 and 128, and Getting started shows 236 and 130 when Tiny Tapeout's own library is missing from beside the board as well. ERC finds 10 errors and DRC finds none in every one of these setups.

So `kicad.sh` runs the image as your uid, and before it runs your command, it copies KiCad's default library tables into place. The course's `labs.yml` calls `kicad.sh` instead of running its steps in a job `container:`, so that CI sees the same tables that the counts on the lab pages were recorded with. wafer.space's workflow runs its container with `--user root`. That's fine for renders and plots, but if it ran ERC and DRC, its warning counts wouldn't match ours.

Fonts and 3D models

KiCad made the image for automation workflows, and it carries KiCad, Python and KiCad's standard symbol and footprint libraries. Two things that board outputs can use aren't in it: some of the fonts a designer might choose, and KiCad's 3D models of parts.

The DIP dock's designer set three silkscreen labels, `ui_in`, `uio` and `uo_out`, in the FreeMono font, which the image doesn't have. Every export of the board, and every DRC run, printed this line (after a timestamp, which we've left off):

```
Font 'FreeMono' not found; substituting 'DejaVu Sans Mono Bold'.
```

So in CI's outputs those three labels are drawn in DejaVu Sans Mono Bold instead of the font the designer chose. Text in KiCad's own default font doesn't depend on the fonts installed on a machine, which makes it the safe choice for a board whose outputs come from CI.

The image also has no 3D models of parts (`/usr/share/kicad/3dmodels` doesn't exist in it), so a render made in CI with this image shows the bare board, apart from any part whose model is saved inside the board file. The models are in `kicad/kicad:10.0.5-full`, which has the same `kicad-cli` as the plain image (we compared the two binaries' checksums) and adds the models. It's a 1,444 MB download, against 808 MB for the plain image. The course's 3D renders are made in that image, named by its digest in `scripts/screenshots/kicad/render-image.txt`.

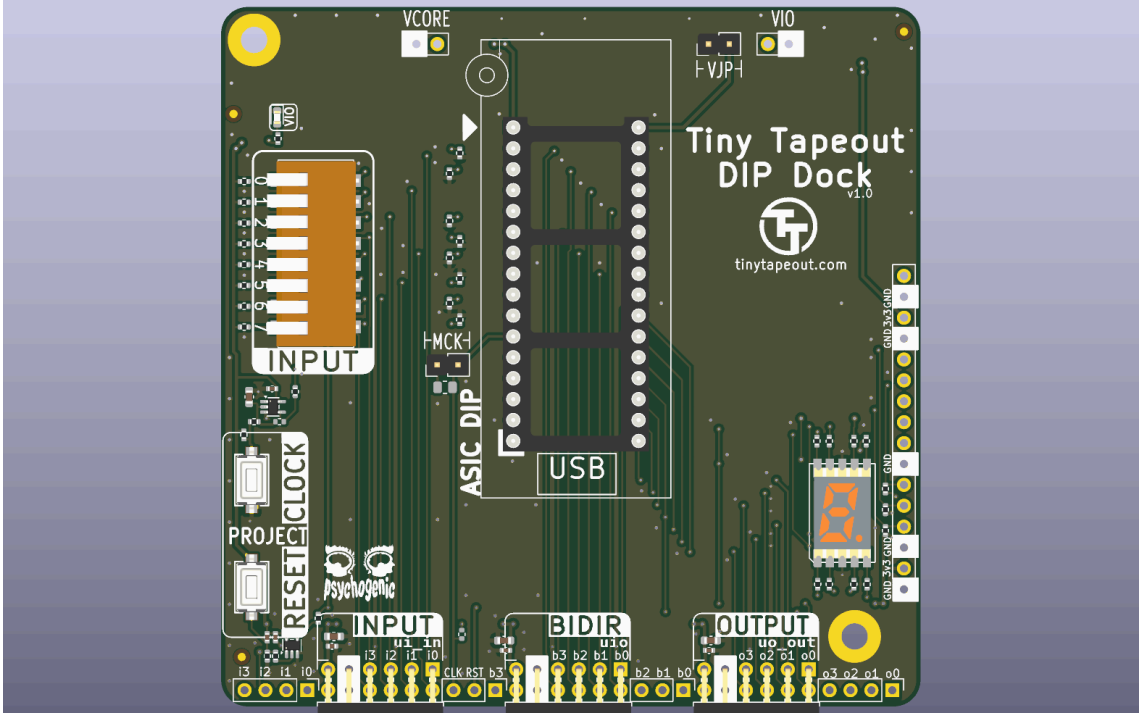


Figure 21.2: A kicad-cli render of the top of the DIP dock, with its parts

The top of Tiny Tapeout's DIP dock as `kicad-cli pcb render` draws it in `kicad/kicad:10.0.5-full`: copper under the green mask, gold pads, silkscreen and holes, and a 3D model for each part whose footprint names one that the image has, such as the DIP switch, the push buttons and the 7-segment display. Made by `scripts/figures/board_renders.sh`.

For a reviewer, a render without the parts is often better, since the pads and the silkscreen are what they're checking, and that's what the plain image gives you.

Outputs that repeat from run to run

It helps to know which outputs come out the same on every run, because then CI can tell you whether a commit changed them:

- SVG plots of layers were identical from run to run once the date that `kicad-cli` writes into each file's `<title>` was removed.
- Two top-view renders of the DIP dock were byte-identical. Two bottom views of the SKY130 breakout differed: the PNGs were 161,218 and 161,207 bytes, and 54 of their 1,542,912 pixels were different.
- Gerbers, the job file and the drill file carry the date of the export, as Fabrication outputs shows, so two exports made at different times are never identical as they stand.

A render can change by a few pixels from one run to the next, so the renders are for people to look at. When a workflow needs to decide whether the board changed, have it compare the Gerbers with their date lines removed.

`kicad-cli` also writes a `.kicad_prl` file of local settings next to the project it reads, even for a check. In CI that doesn't matter, because the runner is thrown away afterwards. On your own machine, it's one of the files that KiCad files in git tells you to ignore.

Jobsets

Instead of listing every export in the workflow, you can save the list in the project as a jobset and run it with `kicad-cli jobset run`. The list of outputs then lives in one file next to the board. We haven't tried a jobset in this course yet.

KiBot

KiBot is a tool that drives KiCad from a YAML file and produces outputs that `kicad-cli` doesn't, such as an interactive HTML BOM. It's free software, under the AGPL-3.0. In Lab 05, Part 5, your workflow runs it after the checks to make an interactive BOM and a design report. On November 16, 2026 its newest release was 1.9.1, dated 2026-07-28. Under "Render 3D", the changelog for that release says "KiCad 10: support for kicad-cli interface, adds some new options but has various limitations". Four things about KiBot affect how the course runs it:

- Its KiCad 10 images lag behind KiCad's releases. The newest KiCad version in any `ghcr.io/inti-cmnb/kicad10_auto_full` tag was 10.0.5, three releases behind KiCad's own 10.0.8. So KiBot's image has the same KiCad version as the course's checks, but it isn't the same build as `kicad/kicad:10.0.5`. In the `1.9.1-2_k10.0.5_d13.2_b4.2.4LTS` image, `kicad-cli version` prints `10.0.5`, but that KiCad is KiCad's Ubuntu package, version `10.0.5-testing-188+202607221313~667fa36054~ubuntu25.10.1`, and the image has the 10.0.4 symbol and footprint libraries and no 3D models. We ran the course's board check in it on the SKY130 breakout and on Lab 04's reference board, and it gave the same counts as `kicad/kicad:10.0.5` on both.
- KiBot's GitHub Action for KiCad 10, `v2_k10`, builds its image `FROM ghcr.io/inti-cmnb/kicad10_auto_full:latest`, and `latest` moves to each new KiBot release. On November 16, 2026, `latest`, `1.9.1` and `1.9.1-2_k10.0.5_d13.2_b4.2.4LTS` all named the same image, `sha256:96e75f885b15…`. Lab 05's workflow doesn't use the action. It names the fixed tag and that digest in `ci/kibot-image.txt`, a copy of the course's `labs/common/kibot-image.txt`, and runs the image with `docker run` as the runner's uid, the same way it runs KiCad's image. On your own machine, `labs/common/kibot.sh` runs it the way `kicad.sh` runs KiCad's.
- The image runs as root unless you give it another user, and it's a 1,347 MB download that takes 3.9 GB on disk. Lab 05's workflow pulls it in KiBot's own step, which GitHub skips when an earlier step has failed, so a run whose board check fails never downloads it.
- KiBot can run ERC and DRC itself, as preflights, but the course doesn't use them. Its DRC fails only on errors unless you tell it to fail on every warning. KiCad reports a footprint that's missing from the board as a warning, so with its default settings KiBot passed Lab 04's board with R3 deleted. The course keeps its checks in `check_board.py`, run in `kicad/kicad:10.0.5`, and uses KiBot only for outputs. Lab 05, Part 5 shows these runs.

Check yourself

1. A workflow runs `kicad-cli pcb drc --severity-error --schematic-parity --exit-code-violations` and goes green, but the board is missing a footprint that the schematic has. How?

   Answer: A1, in Answers at the end.

2. Your workflow names `kicad/kicad:10.0`. One week the DRC warning count changes, and nobody has committed anything. What's the likeliest reason?

   Answer: A2, in Answers at the end.

3. You run ERC on the SKY130 breakout in a container as root and get 233 warnings. The lab page says 97. Which number is wrong?

   Answer: A3, in Answers at the end.

4. The render in your CI artifact shows bare pads and no parts. Is your workflow broken?

   Answer: A4, in Answers at the end.

Next: KiCad files in git.

Answers

A1. Check yourself
KiCad reports schematic parity problems, including a missing footprint, as warnings, and `--severity-error` leaves warnings out, so DRC found nothing at error severity and exited 0. `check_board.py` fails on any parity item for this reason.

A2. Check yourself

The tag moved to a new 10.0 patch release, so CI ran a different KiCad. Naming the image by its digest, as `labs/common/kicad-image.txt` does, keeps it the same until you change it on purpose.

A3. Check yourself

Neither. Root in the image has no library tables, so ERC reports the standard libraries as missing, and the warnings go up. Both setups give the same 10 errors. The lab pages quote warnings only for the course's setup, which `kicad.sh` gives you.

A4. Check yourself

No. The `kicad/kicad:10.0.5` image has no 3D models of parts, so a render made in it shows the bare board. The `-full` image has them, and the course's own 3D renders are made in it.

22. KiCad files in git

This page answers: *which of the files in a KiCad project belong in your repository, and how does anyone review a change to a board when its diff runs to thousands of lines?*

KiCad saves a board, its schematic and its project settings as plain text, so git can store them, show their history and merge them the way it does source code. KiCad also writes other files next to them, such as lock files, a cache and its own local history, and none of those belong in git. And a small change to a board can rewrite thousands of lines of the board file. From Lab 03 on, your board lives in a repository of your own, and in Lab 05 you add the `.gitignore` that this page describes.

What's in a project folder

To see which files KiCad adds to a project, we copied Tiny Tapeout's DIP dock to a scratch folder, opened its board in the PCB Editor of KiCad 10.0.5, saved it without changing anything, and closed the editor, listing the folder after each step. Before we opened the board, the folder already held a `dip-dock.kicad_prl`, because every `kicad-cli` run on a project writes one next to it, even a check. Opening the board added two lock files, `~dip-dock.kicad_pcb.lck` and `~dip-dock.kicad_pro.lck`, which tell another copy of KiCad that the files are in use. Saving added a folder named `.history`, and closing the editor removed the two lock files.

A file belongs in git if you made it and it can't be made again from the other files. The table lists each kind of file in a project and whether it belongs in git. Everything in its "No" rows either records one person's session or can be made again.

File	What it is	In git?
`name.kicad_pro`	The project: design rules, net classes, ERC and DRC severities, and which checks the designer turned off. JSON.	Yes
`name.kicad_sch`	The schematic, one file per sheet.	Yes
`name.kicad_pcb`	The board.	Yes
`name.kicad_dru`	Custom design rules, if the board has any.	Yes
`fp-lib-table`, `sym-lib-table`	The project's own library tables, which say where its footprint and symbol libraries are.	Yes
your own `.kicad_sym` and `.pretty` libraries	Symbols and footprints you made for this board.	Yes
`name.kicad_prl`	Local settings: which layers are visible, and other choices about your own view of the board.	No
`~name.kicad_pcb.lck` and other `.lck` files	Lock files, present only while a file is open.	No
`.history/`	KiCad 10's local history, a folder of automatic snapshots.	No

File	What it is	In git?
`fp-info-cache`	A cache of what's in the footprint libraries, which KiCad can build again.	No
`*-backups/`, `_autosave-*`	Backup and autosave files.	No
Gerbers, drill files, BOMs, renders	Outputs, made from the files above.	No: CI makes them

The `.history` folder is new in KiCad 10, and KiCad puts a `README.txt` in it that says:

```
This directory contains automatic snapshots of your project files.
KiCad periodically saves copies of your work here, allowing you to
recover from accidental changes or data loss.
```

The folder is a git repository of its own, with a `.gitignore` that lists `fp-info-cache` and `*-backups/`, and the README says you can turn it off in Preferences → Common → Project Backup. The snapshots are useful on your own machine, but your board's repository already keeps the board's history, so leave `.history/` out of it.

Outputs stay out of git because each one is correct only for the version of the design it was made from. Tiny Tapeout's `breakout-pcb` repository, for example, tracks `dip-dock/dip-dock.xml`, an export from the Schematic Editor that records the date it was made (2026-10-29) and the full path of the schematic on the designer's own computer. As soon as anyone changes the schematic, that file is out of date until someone remembers to export it again. If you do want to keep the files that went to the fab, keep a copy for each order, the way Tiny Tapeout's separate `pcb-files` repository archives the production files it used for each run.

A `.gitignore` for a board

Tiny Tapeout's demo board repository (`TinyTapeout/tt-demo-pcb` at bbf3dd1) has a `.gitignore` written for KiCad, and it's a good place to start:

```
# For PCBs designed using KiCad: https://www.kicad.org/
# Format documentation: https://kicad.org/help/file-formats/

# Temporary files
*.000
*.bak
*.bck
*.kicad_pcb-bak
*.kicad_sch-bak
*-backups
*.kicad_prl
*.sch-bak
*.lck
*~
_autosave-*
*.tmp
*-save.pro
*-save.kicad_pcb
fp-info-cache

# Netlist files (exported from Eeschema)
*.net

# Autorouter files (exported from Pcbnew)
*.dsn
*.ses

# Exported BOM files
```

```
*.xml
*.csv
```

It covers the lock files (`*.lck`), the local settings (`*.kicad_prl`), the footprint cache and the backups. It was written before KiCad 10, so it doesn't list `.history/`, and a board repository for this course needs that line as well.

Ignoring a file that git already tracks

Line 19 of the demo board's `.gitignore` is `fp-info-cache`, but the same repository tracks the board's `fp-info-cache` anyway, and at the pinned commit the cache is a 3,924,749-byte file in git like any other. The cache was committed first, on October 14, 2023 at 16:49 (commit 974c613, 93,108 lines), and the `.gitignore` that lists it was committed four minutes later, at 16:53 (commit 0e63c95). Counting that first commit, 20 commits up to the pinned one have changed the cache, adding 306,006 lines and removing 198,254, all in a file that KiCad can build again by itself.

This is a common mistake, and git is doing what its documentation describes. The gitignore manual page says that a gitignore file "specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected." A pattern in `.gitignore` stops git from *adding* a file, and it does nothing to a file that git already has. To stop tracking one, the same page says to "use `git rm --cached` to remove the file from the index". That leaves your copy on disk and records the file's removal in your next commit, and from then on the ignore rule applies to it.

The demo board's `.kicad_prl` was in the same first commit, and the same `.gitignore` lists it as `*.kicad_prl`. It stayed in the repository until commit 3b4873f removed it in September 2025. Counting the commit that added it and the one that removed it, 31 commits changed it. So write the `.gitignore` before your first commit, and if `git status` ever lists a file you meant to ignore, check whether git is already tracking it.

A save with no edits

When we saved the DIP dock in the course's pinned image without touching anything, git saw 111 lines added and 225 removed in `dip-dock.kicad_pcb`. Every changed line was inside the `render_cache` of three silkscreen labels, `ui_in`, `uio` and `uo_out`. The designer set those three labels in the FreeMono font, and KiCad stores the outlines of labels like these in the board file. The image doesn't have FreeMono, so KiCad drew the labels again in the font it substituted and saved the new outlines. (Outputs in CI shows the warning that `kicad-cli` prints about the same missing font.)

The same thing can happen on your own computer with any font that the designer had and you don't. A KiCad 9 board that you save in KiCad 10 is also rewritten in the newer format, as Getting started explains. Neither does the board any harm, but both put lines in a commit that have nothing to do with the change you meant to make. Read `git diff --stat` before you commit, and if it lists a file you didn't mean to change, find out why before you commit it.

Reviewing a change to a board

git shows a change to a board file as a text diff, the same as for any other file, but the diff can be enormous. Commit ecb636a of the demo board, "Reworked manual clocking circuit, Iref current source for analog, TVS diodes on analog lines", changed 60,318 lines of `tinytapeout-demo.kicad_pcb` and 12,558 lines of its schematic. Later the same day, a commit that its author called "minor pcb cleanup" (293a3e2) still changed 5,770 lines of the board, far more than a reviewer can read line by line.

A reviewer can compare pictures of the board instead. Here is the "minor" commit as two renders made by `kicad-cli`, with the pixels that changed between them marked in red:

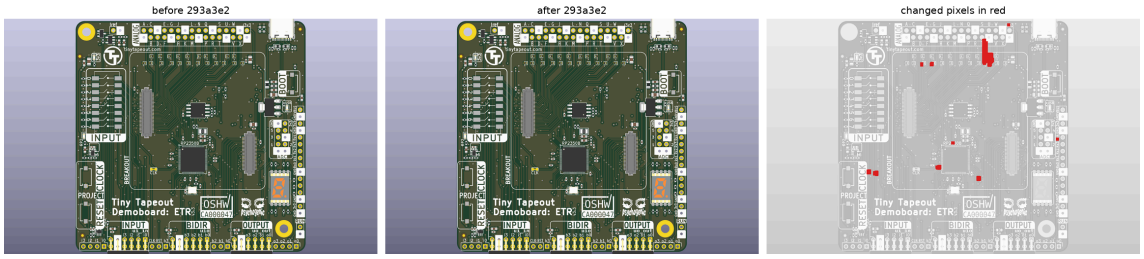


Figure 22.1: Renders of Tiny Tapeout's demo board before and after commit 293a3e2, and the changed pixels in red: a track near the analog header and a few small spots elsewhere

Tiny Tapeout's demo board (`tt-demo-pcb`, Apache-2.0) before and after commit 293a3e2, rendered from the top, with the parts' 3D models, by `kicad-cli pcb render` in `kicad/kicad:10.0.5-full`. 934 of the 1,542,912 pixels in a render changed, about 0.1 %. The red marks are thickened so that you can see them at this size. They are a track near the analog header with a spot at the header's right-hand end, a few spots along the row of small parts below the header and around the RP2350, two spots on the left edge of the board and one on the right. Made by `scripts/figures/review_diff.py`.

Lab 05's workflow makes renders and layer plots on every push so that whoever reviews a pull request can compare the images from the two commits, and open the board in KiCad only where a picture raises a question. A render shows only the outside of the board, so the top render above would miss a change on the bottom or on an inner layer. The workflow renders the bottom as well, and a change on an inner layer shows up only in that layer's plot.

Pictures are easiest to review when each commit makes one change. If a commit moves a part and also re-annotates the schematic, both the pictures and the diff show the two changes at once, and the reviewer can't tell which marks or lines belong to which. (On the demo board, commit 9c07f7f, "re-annotated entire schem", replaced 2,102 lines of the board file by itself, which git counts as 4,204 changed lines.) Write what changed, and why, in the commit message too, because the message is the only part of a commit that a reviewer reads without opening anything.

KiCanvas, a viewer for KiCad files that runs in a browser, would let a reviewer look at a board without installing KiCad. Its last commit was on April 28, 2026, though, and we haven't confirmed that it reads KiCad 10 files, so until someone does, use the images from CI and KiCad itself.

Libraries in the repository

KiCad needs a board's libraries to check the board fully, so keep them where KiCad can find them on any machine. KiCad's project library tables (`fp-lib-table` and `sym-lib-table`) can give a library's path relative to the project with `${KIPRJMOD}`, which stands for the project's own folder. A library inside the repository is then found wherever the repository is cloned.

Tiny Tapeout keeps its library in a git submodule. The SKY130 breakout's tables find it at `${KIPRJMOD}/../tinytapeout-kicad-libs/`, a folder beside the board's own, and `breakout-pcb`'s `.gitmodules` makes that folder a submodule that points at `psychogenic/tinytapeout-kicad-libs`. A plain `git clone` leaves a submodule's folder empty until you ask git to fetch it, so the course's fetch script (Getting started) fetches the library separately and puts it in that folder.

A board still opens when its libraries are missing, because the board and schematic files carry copies of every footprint and symbol they use. DRC and ERC compare those copies with the libraries, and they report each copy whose library they can't find or whose library version differs, so missing libraries show up as warnings. The SKY130 breakout's stitching vias use a footprint library named `flyingcarsfootprints`, which isn't in either of Tiny Tapeout's library repositories, and we couldn't find it published anywhere, so there's no library you can install to clear the warnings about it. Your own board shouldn't need anything that isn't in your repository or in KiCad's standard libraries.

Check yourself

1. After an evening's work, `git status` lists `board.kicad_prl`, `~board.kicad_pcb.lck` and a new `.history/` folder as untracked. Which of them do you commit?

   Answer: A1, in Answers at the end.

2. You add `fp-info-cache` to your `.gitignore`, but `git status` still shows it as modified after every session. Why, and what fixes it?

   Answer: A2, in Answers at the end.

3. You opened your board, changed nothing, saved it, and `git diff --stat` shows 300 changed lines in the `.kicad_pcb`. Name two things that could cause that.

   Answer: A3, in Answers at the end.

4. A teammate's pull request changes 5,000 lines of the board file. How do you review it?

   Answer: A4, in Answers at the end.

Next: Lab 05: Fab outputs in CI.

Answers

A1. Check yourself

None. The `.kicad_prl` holds your local view settings, the lock file exists only while the board is open, and `.history/` holds KiCad's own snapshots. All three belong in `.gitignore`.

A2. Check yourself

git is already tracking the file, and `.gitignore` doesn't affect tracked files. `git rm --cached fp-info-cache` removes it from the index without deleting your copy; after you commit that, the ignore rule applies. This is what happened to Tiny Tapeout's demo board, where the cache was committed four minutes before the `.gitignore` that lists it.

A3. Check yourself

KiCad redrawing text in a font your machine doesn't have and saving the new outlines, as it did with the DIP dock in the course image, and KiCad converting a file saved by an older major version to its own format. Look at which lines changed before you commit them.

A4. Check yourself

Not by reading the diff. Compare the renders and layer plots that CI made for the two commits, open the board in KiCad where a picture raises a question, and read the commit messages for what was meant to change. If the pull request mixes several changes, ask for it to be split.

23. Lab 05: Fab outputs in CI

In this lab you give your board repository a GitHub Actions workflow. On every push it runs ERC, DRC and the schematic parity check on your board. Then it builds the Gerbers and the drill file that a fab needs, the BOM and the position file that an assembler needs, and renders, copper plots and a PDF of the schematic for whoever reviews the board. KiBot then makes an interactive BOM and a design report, and the workflow uploads all of these files as one artifact. You'll write the `.gitignore` that keeps KiCad's local files and the build's outputs out of git, and finish the commands in the build script that the workflow runs. Then you'll open the Gerbers in GerbView to see them as the fab will, and write KiBot's configuration. Last, you'll break the board on purpose and check that the workflow fails on it.

The lab's files are in `labs/lab-05-fab-outputs-in-ci/` in the course repo.

Before you start

- Finish Lab 04: its check passes on your repository, and the commit it checked is pushed to GitHub.
- Read the pages on row 9 of the course path: Fabrication outputs, Outputs in CI and KiCad files in git.

The check is a bash script, as in Labs 03 and 04, so on Windows you run it in WSL. It runs both KiCad and git inside the course image, and we have run it only on Linux. It also runs KiBot, in KiBot's own image, which is a 1,347 MB download the first time you run the check.

The lab takes about 1 hour 40 minutes: roughly 10 minutes for Part 1, 10 for Part 2, 25 for Part 3, 10 for Part 4, 25 for Part 5 and 20 for Part 6. Those times are our estimates. We did time the check itself. With one CPU for each container, it took 26 s on our reference repository, and 42 s with `--planted`, because that also runs the board check on a second commit.

Part 1: Add the workflow

1. Copy the starter into your board repository and see what arrived:

   ```
   cd ~/simproc-pmod
   cp -r ~/pcb101/labs/lab-05-fab-outputs-in-ci/starter/. .
   git status
   ```

   You should see `git status` list two new folders, `.github/` and `ci/`, and also `pmod.kicad_prl`, which is still untracked because Lab 03 had you leave it out of your commits. `.github/workflows/board.yml` is the workflow, and `ci/build.sh` is the script it runs. `ci/check_board.py` is a copy of the course's board check. `ci/kicad-image.txt` names the pinned KiCad image by its digest, and `ci/kibot-image.txt` does the same for KiBot's image. `ci/board.kibot.yaml` is KiBot's configuration, which you'll finish in Part 5.

2. Read `board.yml`. After checking out your commit, it pulls the image named in `ci/kicad-image.txt`, then runs `bash ci/build.sh check`, `fab` and `images` as three steps, each in a fresh container. A fourth step pulls the KiBot image named in `ci/kibot-image.txt` and runs `bash ci/build.sh kibot` in it, and the last step uploads the folder `out/` as an artifact called `board-outputs`. Then read `ci/build.sh`, which has the commands that those four steps run.

   Why does the workflow start the containers with `docker run` as the runner's own uid, rather than running the whole job in the image with GitHub's `container:` setting?

   Answer: A1, in Answers at the end.

3. Commit the starter, but don't push it yet. Name its two folders in `git add`, so that `pmod.kicad_prl` stays out of the commit:

```
git add .github ci
git commit -m "Lab 05 starter"
```

If you pushed it now, would the run on GitHub be green or red?

Answer: A2, in Answers at the end.

4. `build.sh` has five TODOs, which you'll do in Part 3, and `ci/board.kibot.yaml` has three, which you'll do in Part 5. There's no `.gitignore` yet either. Before you run anything, write down which rows of the Lab 05 check you expect to fail. Then run it:

```
~/pcb101/labs/lab-05-fab-outputs-in-ci/check.sh
```

It first runs `bash ci/build.sh all` in the pinned KiCad image and prints that command's output, apart from the renders' progress lines. If the build passes, it runs `bash ci/build.sh kibot` in the KiBot image, as your workflow would. Either way, it then prints one row per check. On Lab 04's reference board with the starter copied in and committed, the rows were:

```
  ok    workflow   .github/workflows/board.yml runs check, fab and images in kicad/
kicad:10.0.5 and kibot in kicad10_auto_full:1.9.1-2_k10.0.5_d13.2_b4.2.4LTS, both
by digest, and uploads out/
  ok    check      the board check passes
  ok    gerbers    9 Gerbers and pmod-job.gbrjob in out/fab/
  FAIL  drill      no drill file (.drl) in out/fab/
  FAIL  bom        no BOM in out/assembly/ (a .csv with 'bom' in its name)
  FAIL  position   no position file in out/assembly/ (a .csv with 'pos' in its name)
   FAIL  images      out/review/ is missing top render, bottom render, F.Cu plot,
B.Cu plot
   FAIL  ibom        no interactive HTML BOM in out/kibot/ (the .html that an ibom
output writes)
  FAIL  report     no design report in out/kibot/ (a .txt from a report output, with
a 'Board size:' line)
   FAIL  gitignore   no .gitignore; after the build git sees 15 untracked files:
out/fab/pmod-B_Cu.gbr, out/fab/pmod-B_Mask.gbr, out/fab/pmod-B_Paste.gbr, out/fab/
pmod-B_Silkscreen.gbr ...

FAIL  pmod: 3 of 10 Lab 05 checks ok; add --planted REV once you've done Part 6
```

Part 2: Keep KiCad's own files out of git

Which files in your repository's folder belong in git, and which should git ignore?

1. Run `git status` again. It lists `out/`, which the build just wrote, and `pmod.kicad_prl`, which `kicad-cli` writes next to any project it reads. If you've saved the board in KiCad, it may also list a `.history/` folder or a `pmod-backups/` folder, and while the board is open in KiCad, it lists lock files ending in `.lck`. KiCad files in git says what each one is.
2. Make a `.gitignore` in the repository's root folder. Start from Tiny Tapeout's, which KiCad files in git prints in full, and add the two lines it doesn't have: `.history/` for KiCad 10's local history, and `out/` for everything `build.sh` writes. CI makes `out/` again on every run, so it never needs to be in git.
3. Run `git status`. You should see `.gitignore` as the only new file, apart from any files of your own that you haven't committed yet. If a file you meant to ignore still shows as modified, git is already tracking it. Run `git rm --cached` on it and commit, and git stops tracking it.
4. Tiny Tapeout's demo board repository lists `fp-info-cache` in its `.gitignore`, and git still tracks the file. How?

   Answer: A3, in Answers at the end.

5. Commit the `.gitignore` and run the check. You should see `ok  gitignore  nothing untracked after the build, and none of KiCad's local files tracked`.

Part 3: Finish the fabrication files and review images

What do a fab, an assembler and a reviewer each need from your CI?

1. Write the five commands that the TODOs in `ci/build.sh` ask for. Each `kicad-cli` command prints its own options with `--help`; for example, run

   ```
   ~/pcb101/labs/common/kicad.sh kicad-cli pcb export drill --help
   ```

   and the same for `sch export bom`, `pcb export pos`, `pcb render` and `pcb export svg`. Keep the file names that the comments give, because the check finds each file by its name. Things to decide as you go:

   - The drill file has to use the same origin as the Gerbers. To see which origin the Gerbers use, open File > Plot in the PCB Editor. If "Use drill/place file origin" isn't ticked, the Gerbers use the absolute origin, and so should the drill file (`--drill-origin`). On our board the drill/place origin was never moved from (0, 0), so `absolute` and `plot` wrote the same file. On a board where someone has placed that origin, such as the DIP dock, the two options put the holes in different places.
   - An assembler wants the position file as CSV, in millimetres (`--format`, `--units`), but by default the command writes a text table in inches.
   - A render takes `--side top` or `--side bottom`.
   - For the copper plots, `--layers F.Cu,B.Cu` with `--mode-multi` writes one SVG per layer, `--common-layers Edge.Cuts` adds the outline to each, and `--page-size-mode 2` with `--exclude-drawing-sheet` crops them to the board.

2. Before you run the check, count the holes that your drill file should have, one for every pad with a hole and one for every via, and write the number down. On our board it's 70: 36 pins in the three Pmod connectors, 3 in the dongle header, 16 in the switch, 9 in the resistor network and 6 vias.

3. Run the check. You should see every row `ok` except `ibom` and `report`, which fail until you've done Part 5. Our rows were:

   ```
     ok    workflow   .github/workflows/board.yml runs check, fab and images in kicad/
   kicad:10.0.5 and kibot in kicad10_auto_full:1.9.1-2_k10.0.5_d13.2_b4.2.4LTS, both
   by digest, and uploads out/
     ok    check      the board check passes
     ok    gerbers    9 Gerbers and pmod-job.gbrjob in out/fab/
     ok    drill      pmod.drl: 70 holes, all inside the outline
     ok    bom        pmod-bom.csv: 15 rows, every footprint listed
     ok    position   pmod-pos.csv: 13 parts, in mm, where the board has them
     ok    images     5 files in out/review/: renders, copper plots and the schematic
    FAIL   ibom        no interactive HTML BOM in out/kibot/ (the .html that an ibom
   output writes)
    FAIL   report     no design report in out/kibot/ (a .txt from a report output, with
   a 'Board size:' line)
     ok    gitignore  nothing untracked after the build, and none of KiCad's local
   files tracked

   FAIL  pmod: 8 of 10 Lab 05 checks ok; add --planted REV once you've done Part 6
   ```

   The drill row checks that every hole lies inside the outline in the Edge.Cuts Gerber, so a drill file made in a different origin from the Gerbers fails it. Fabrication outputs shows that mistake on the DIP dock.

4. Our BOM has 15 rows and our position file only 13 parts. Open `out/assembly/` and find the two parts that the position file leaves out. Why would KiCad leave them out?

   Answer: A4, in Answers at the end.

5. Commit and push. On your repository's page on GitHub, the Actions tab shows a run of the workflow, which is named Board, for your commit. The run's page lists the job's steps, and once the run has finished, it also has the `board-outputs` artifact, which downloads as a zip of `out/`. You should see a green run, even though KiBot has nothing to make yet. We couldn't push from the machine we wrote the course on, so we ran the workflow's steps locally with `src/run_workflow.py`, with each container

limited to one CPU. All seven steps succeeded in 35 s, and the artifact held 21 files. On GitHub the run also has to pull the two images (808 MB and 1,347 MB), which adds to that time, and we haven't timed a run there.

Part 4: Look at what the fab will get

Do the files CI built describe your board?

1. Unzip the artifact, open GerbView (the Gerber Viewer, from KiCad's project manager), and use File > Open Gerber Plot File(s)... for every `.gbr` in `fab/`, then File > Open Excellon Drill File(s)... for the `.drl`. The `out/fab/` folder that the check wrote on your own machine has the same files.

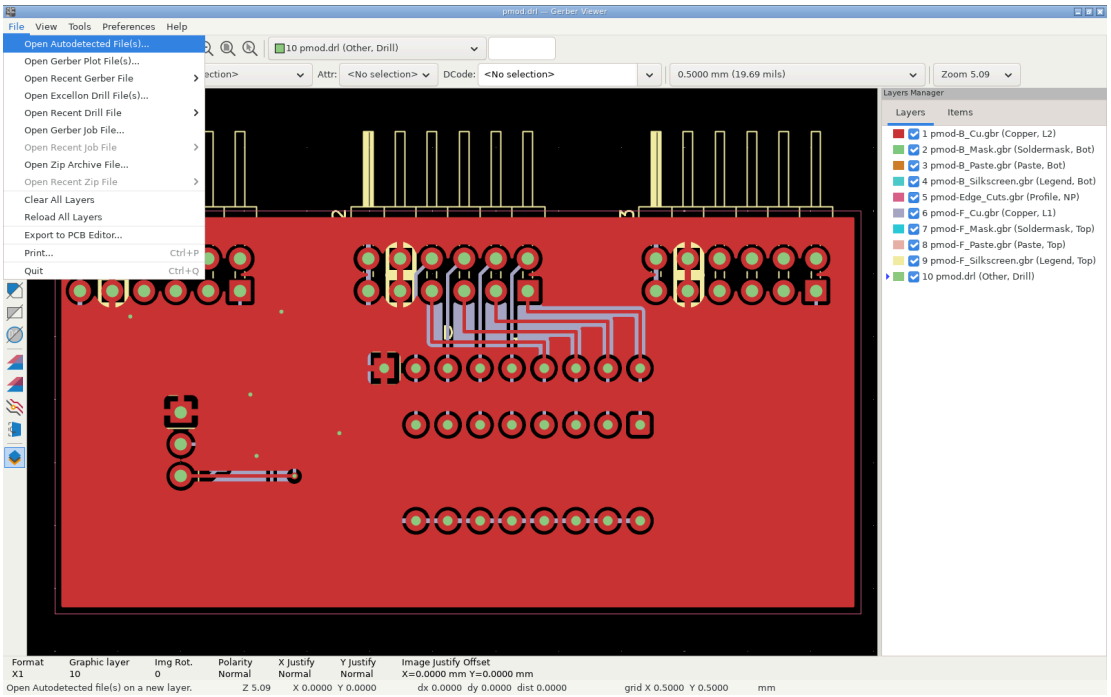


Figure 23.1: GerbView with its File menu open over the SimProc Pmod's Gerbers

GerbView's File menu in KiCad 10.0.5, over the nine Gerbers and the drill file that `ci/build.sh fab` wrote for our Lab 04 board. We opened these from the command line (`gerbview out/fab/*.gbr out/fab/*.drl`), which also works if KiCad's programs are on your path.

2. Before you look closely, write down how many layers the Layers Manager should list and what function GerbView should give each.
3. Compare the view with your board in the PCB Editor. You should see every drill hit (the small dots of the drill layer) in the middle of its pad, and the Layers Manager listing every file with the function that GerbView read from it, such as "Copper, L1" for the top copper and "Profile, NP" for the outline:

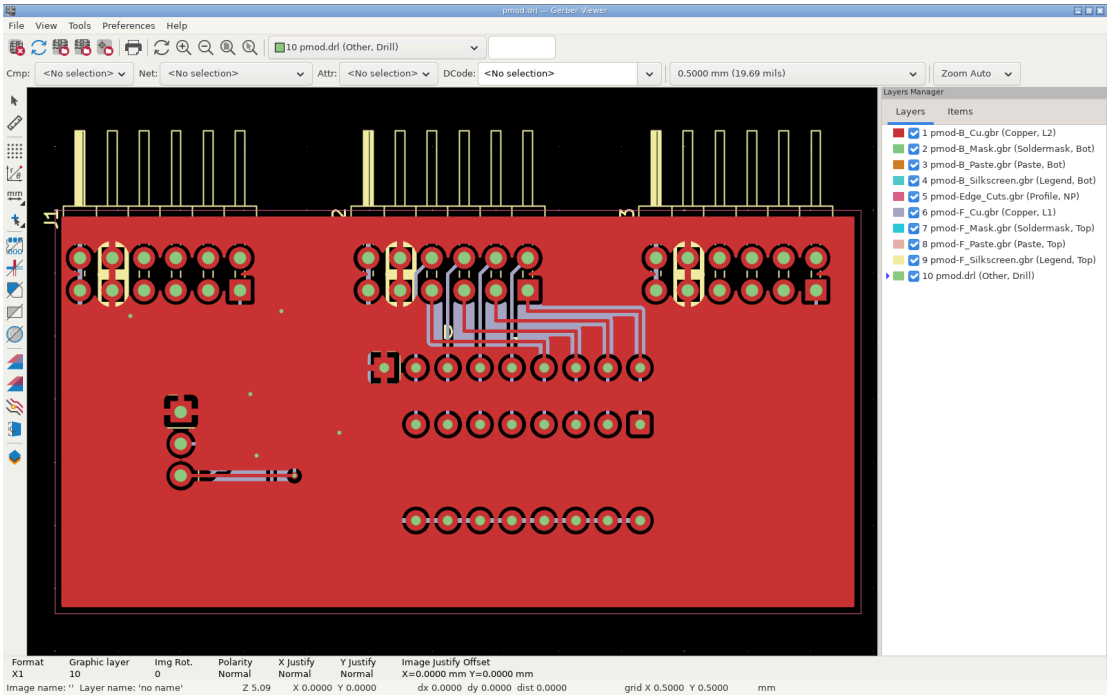


Figure 23.2: GerbView with the SimProc Pmod's nine Gerbers and drill file

Our board's nine Gerbers and its drill file in GerbView. The red is the bottom copper, mostly its ground pour, and the lighter tracks are on the top copper; untick a layer in the Layers Manager to see the others more clearly.

If every hole sat the same distance away from its pad, what would you change?

Answer: A5, in Answers at the end.

Part 5: Add KiBot's outputs

How do you add outputs that `kicad-cli` can't make, such as an interactive BOM, and keep the whole run on one version of KiCad?

1. Look at what the starter put in place for KiBot. KiBot is a program that drives KiCad from a YAML file, and it can make outputs that `kicad-cli` can't. It's free software, under the AGPL-3.0. It needs a good deal of software besides KiCad, so it runs in an image of its own, `ghcr.io/inti-cmnb/kicad10_auto_full`, which `ci/kibot-image.txt` names by its digest. That image carries KiBot 1.9.1 and KiCad 10.0.5, the same KiCad version that the course's checks use. The workflow's "KiBot outputs" step pulls the image and runs `bash ci/build.sh kibot` in it with `docker run`, as the steps before it run theirs. KiBot also has a GitHub Action of its own, but the action builds its image from a tag that moves with every KiBot release, so the workflow doesn't use it (Outputs in CI has the details). The `kibot` stage at the end of `build.sh` runs KiBot on `ci/board.kibot.yaml` and has it write what it makes into `out/kibot/`. You'll finish that configuration file in step 4.

   KiBot can also run ERC and DRC itself; it calls them preflights. Why does the workflow keep its checks in `check_board.py`, in KiCad's own image, and use KiBot only for outputs?

   Answer: A6, in Answers at the end.

2. See what KiBot can make by running its help in KiBot's image, through the course's `kibot.sh`:

```
~/pcb101/labs/common/kibot.sh kibot --help-list-outputs
~/pcb101/labs/common/kibot.sh kibot --help-output ibom
```

   The first lists 56 types of output. Many of them are files that `kicad-cli` makes too, such as Gerbers, drill files and position files. Two that `kicad-cli` doesn't make are `ibom`, an interactive HTML BOM, and

`report`, a design report written for whoever orders the board. `kicad-cli pcb export stats` writes a board statistics report that overlaps with KiBot's `report`. It gives each hole at its finished size, though, and it leaves out what KiBot adds for the fab, such as the drill sizes, a Eurocircuits board class and an estimate of the solder paste. The second command prints every option of the `ibom` output, and `kibot --help-output report` and `kibot --help-filters` do the same for the report and for KiBot's filters.

3. Before you change the configuration, write down which of your board's parts the interactive BOM should list, and how many rows it will have if parts with the same value and footprint share a row. Then decide what it should do with a part marked DNP (Tiny Tapeout's boards, part by part explains the mark): should a part you won't fit be on the list you solder from?

4. Finish the three TODOs in `ci/board.kibot.yaml`. Things to decide as you go:

   - A filter goes in a `filters:` list, with a `name` of your choosing and a `type`. The `generic` type has an option that picks out the parts KiCad marks "Do not populate"; `kibot --help-filters` lists it.
   - The `ibom` output names that filter in its `dnf_filter` option (DNF is "do not fit"), and every part the filter picks out is then left off the BOM. Without a filter, the ibom lists a DNP part along with all the others.
   - The `report` output needs only its `name` and `type`. Keep its default template, because the check reads the board size and the drill tools from it.
   - Leave the file names as KiBot makes them, which for our board were `pmod-ibom.html` and `pmod-report.txt`.

5. Run the KiBot stage in the pinned KiBot image:

   ```
   ~/pcb101/labs/common/kibot.sh bash ci/build.sh kibot
   ```

   You should see a line for each output, and the two files in `out/kibot/`. Ours took 2.3 s with one CPU:

   ```
   pcb101: KiBot image ghcr.io/inti-cmnb/kicad10_auto_full@sha256:96e75f885b15617c6b38aef11db7856a93f4367c
   == KiBot outputs
   CI/CD environment detected
   WARNING:(W182) No instance for `/00000000-0000-0000-0000-000000000000` using the
   first (kibot - v6_sch.py:1458)
   - 'Interactive HTML BOM' (ibom) [ibom]
   WARNING:(W013) U1 not linked to an existing schematic component, but U1 found in
   schematic, assuming this is the same (kibot - kiplot.py:488)
   - 'Design report' (report) [report]
   Found 2 unique warning/s (47 total, 3 filtered)
   ```

   The text in quotes on each output's line is that output's `comment`. Our configuration gives each output one (`Interactive HTML BOM` and `Design report`). An output without a comment makes the same file, and its line reads `- '' (ibom) [ibom]`.

   The first warning, W182, comes from the way KiBot reads a schematic. KiBot looks for each symbol's instance under a sheet path of zeros, while our schematic records each one under the schematic's own UUID, so KiBot falls back to the symbol's first instance, as the message says. Our schematic is one sheet, and each of its 43 symbols (the power symbols and U1's three units among them) has only one instance, so the first instance is the right one. KiBot prints W182 once and counts it for each symbol.

   The second warning, W013, is about U1, the 74LVC2G17 dual buffer. The schematic draws it as three units, one for each buffer and one for the GND and VCC pins. Each unit is a symbol of its own in the schematic, with its own UUID. KiBot folds the three into one U1 and keeps the UUID of the first unit it reads, unit 2, which is one of the buffers. Our board's U1 footprint records the UUID of unit 3, the one with the power pins, so KiBot can't match the footprint to its symbol by UUID. It looks the footprint up by its reference instead and finds the U1 it folded, which is the right part.

6. Open `out/kibot/pmod-ibom.html` in a browser. When you click a row of the BOM, its parts light up on the drawing of the board. You should see the parts and rows that you wrote down in step 3. Ours has 15 parts in 11 rows:

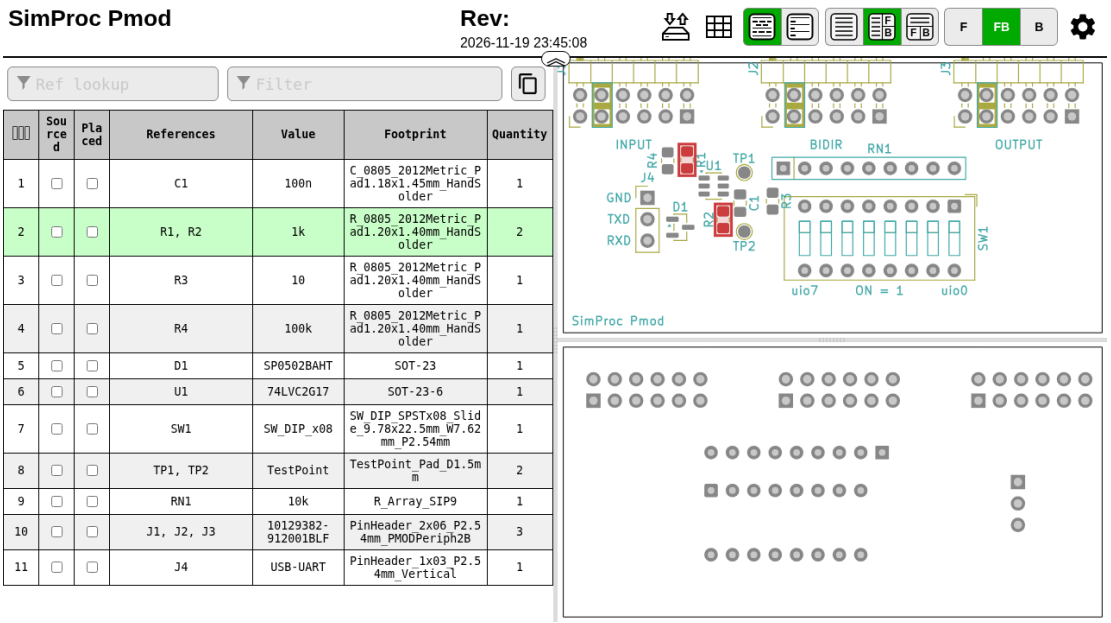


Figure 23.3: KiBot's interactive BOM for the SimProc Pmod, with the row for R1 and R2 selected and both resistors highlighted on the front of the board

KiBot's interactive BOM for our Lab 04 board, with the row of the two 1k resistors selected, so R1 and R2 are drawn in red on the front of the board (top right); the bottom is below it. The Sourced and Placed columns are checkboxes to tick as you buy and solder each row, and the page keeps the ticks in your browser's storage. Made by `scripts/screenshots/browser/ibom.cjs`.

Our board has no DNP part, so the filter changes nothing on it. What would the BOM have shown for a DNP part without the filter?

Answer: A7, in Answers at the end.

7. Open `out/kibot/pmod-report.txt`. It's plain text, and it gives the board's size and layers, the sizes that a fab's rules are about (clearance, track width, drills, vias and annular ring), the holes, and an estimate of the solder paste. Compare its holes with the count that you wrote down in Part 3, step 2. Ours lists three drill tools:

```
Drill tools (including vias and computing adjusts and rounding):

- 0.4 mm (16 mils) (6)
- 0.9 mm (35 mils) (25)
- 1.1 mm (43 mils) (39)
```

That's 70 holes, as in the drill file, but the vias and the pads on our board have holes of 0.3, 0.8 and 1.0 mm. Why is each tool 0.1 mm larger?

Answer: A8, in Answers at the end.

Some lines of the report are KiBot's defaults rather than facts about your board. Our board file records no stack-up, so the finish (HAL), the material (FR4) and the colours of the solder mask and silkscreen are KiBot's defaults. So is the "yes" after micro vias and buried/blind vias, which says what KiBot assumes the rules allow; the "Total" line under them counts the vias the board really has. Check those lines against what you'll order before you hand the report to anyone.

8. Run the Lab 05 check. You should see the two new rows `ok`:

```
  ok    ibom       pmod-ibom.html: 15 parts in 11 rows, none of them marked DNP
  ok    report     pmod-report.txt: 64 x 32 mm and 70 holes, as in the Gerbers
and pmod.drl
```

and the last line `PASS  pmod: 10 of 10 Lab 05 checks ok; add --planted REV once you've done Part 6`. The `ibom` row reads the board data inside the page and compares its BOM with the footprints on your board. It passes when every part that belongs in a BOM is listed, none of the listed parts is marked DNP, and the BOM has nothing that isn't on the board. The `report` row compares the report's board size with the outline in the Gerbers, and checks that its drill tools add up to the number of holes in the drill file. Then commit `ci/board.kibot.yaml` and push. The run's "KiBot outputs" step now makes the two files, and the artifact has a `kibot/` folder. Our local run of the workflow, with one CPU for each container, succeeded in all seven steps in 30 s, and the artifact held 23 files.

Part 6: Break it on purpose

Does your workflow stop a broken board before it makes a set of Gerbers?

1. You're about to move a track so that it passes too close to a pad of another net. Write down which step of the workflow will fail, which steps will run after it, and what the artifact will hold.
2. In the PCB Editor, select a track and drag it (press D) until it passes within 0.2 mm of a pad or track on another net, without touching it. Save. On our board we dragged the 0.4 mm +3V3 track that runs along the top edge from J1 to J2 down by 1 mm, which takes its lower edge to 0.16 mm from the upper row of J1's pads.
3. Run DRC in the editor (Inspect > Design Rules Checker, then Run DRC). You should see at least one error. Ours had two:

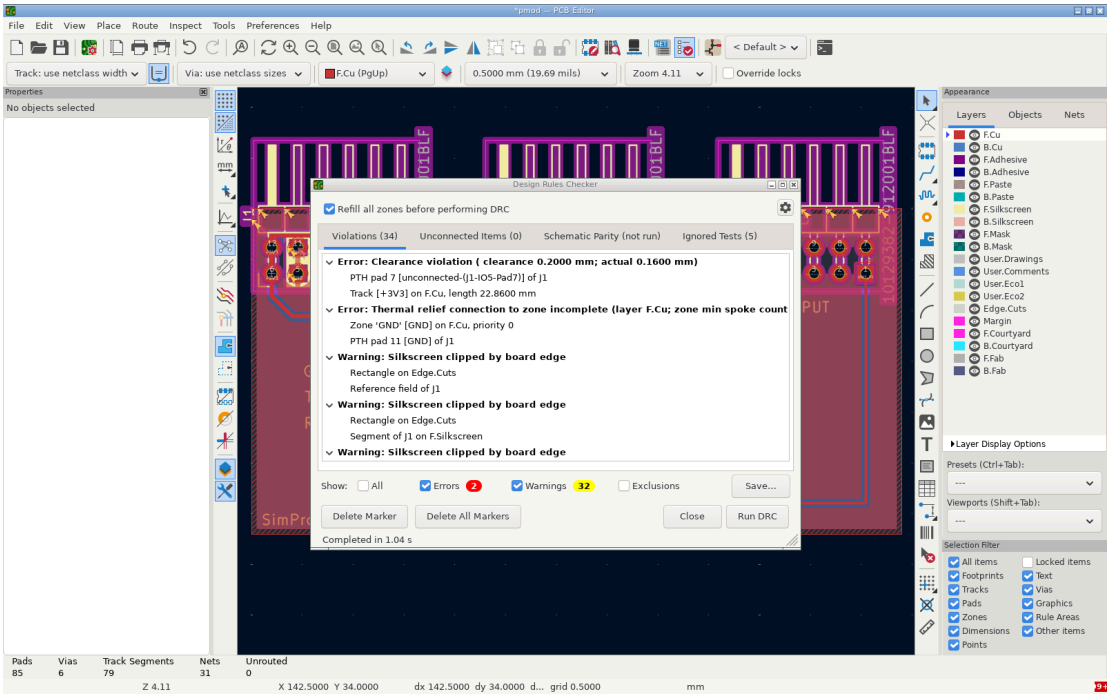


Figure 23.4: The Design Rules Checker with a clearance violation and a thermal relief error

The Design Rules Checker in KiCad 10.0.5 on our board with the track dragged down. It lists a clearance violation between the track and J1's pad 7 (0.1600 mm against 0.2000 mm), and an incomplete thermal relief on pad 11, because the track now blocks part of that pad's connection to the GND zone. The other 32 items are the warnings that the board had before.

The dialog's Schematic Parity tab says "(not run)". Does that mean your CI doesn't check parity either?

Answer: A9, in Answers at the end.

4. Run the Lab 05 check. You should see the board check fail inside the build, with the errors that your drag made:

```
  DRC errors (2):
    clearance              Clearance violation ( clearance 0.2000 mm; actual 0.1600
mm)  [PTH pad 7 [unconnected-(J1-IO5-Pad7)] of J1; Track [+3V3] on F.Cu, length
22.8600 mm]
    starved_thermal          Thermal relief connection to zone incomplete (layer
F.Cu; zone min spoke count 2; actual 1)  [Zone 'GND' [GND] on F.Cu, priority 0; PTH
pad 11 [GND] of J1]
```

and then eight rows fail. The first is `check` (`the board check fails: drc.error 2 (want 0); the fab and images stages didn't run`). `gerbers`, `drill`, `bom`, `position` and `images` fail because `build.sh` stops at the first stage that fails, so the later stages never made their files. `ibom` and `report` fail too (`` `ci/build.sh kibot` didn't run, because an earlier stage failed, as in your workflow ``), because the check, like the workflow, runs KiBot only after the build has passed.

5. Commit the broken board with a message that says what you did, and push. On GitHub you should see a red run. The step "ERC, DRC and schematic parity" fails with the same board check output as above, GitHub skips "Fabrication files", "Review images" and "KiBot outputs", and the upload step still runs, so the artifact holds `reports/` and nothing else. Our local run of the workflow did the same. Its check step exited 1, the next three steps were skipped, and the artifact had three files, `drc.json`, `erc.json` and `summary.json`.
6. Fix the board by moving the track back, or by undoing the drag with Ctrl+Z before you close the editor. Then save, run the check until it passes, commit, and push. You should see a green run on GitHub, with the full artifact again.
7. Find the hash of the commit with the planted error (`git log --oneline` lists it) and run the check with it:

   ```
   ~/pcb101/labs/lab-05-fab-outputs-in-ci/check.sh --planted <hash>
   ```

   It copies that commit's files into a temporary folder, runs `bash ci/build.sh check` there, and checks that the board check fails. You should see an eleventh row, `ok  planted  f1b503c, an earlier commit on this branch, fails the board check (drc.error 2)`, with your own hash and counts, and the last line `PASS  pmod: 11 of 11 Lab 05 checks ok`.

Finishing

You're done when `~/pcb101/labs/lab-05-fab-outputs-in-ci/check.sh --planted <your planted commit>`, run in your own repository, ends with `PASS  pmod: 11 of 11 Lab 05 checks ok`, and your repository's Actions tab shows a red run on that commit and a green run, with its `board-outputs` artifact, on a later one.

Questions:

1. A teammate's pull request moves a part. Which files in the run's artifact would you look at first, and why not the `.kicad_pcb` diff?

   Answer: A10, in Answers at the end.

2. CI builds Gerbers from the same commit on two days, and their checksums differ. Did the board change?

   Answer: A11, in Answers at the end.

Submit: nothing for this lab. Keep the workflow; the capstone asks for a green run.

Going further (optional)

- Save the list of outputs in the project as a jobset and run it with `kicad-cli jobset run`, so that the list lives in a file next to the board instead of in `build.sh`. We haven't tried a jobset in this course yet.

Next on the Course path: read Reviewing and ordering a board, then do Lab 06: Review a seeded board.

Answers

A1. Part 1: Add the workflow

So that the checks run the way the course's own checks do, as an ordinary uid, with KiCad's default library tables copied into an empty home folder. `build.sh` copies the tables before it runs any stage. A job `container:` runs as the image's own user unless you override it, and wafer.space's workflow overrides it with `--user root`. Outputs in CI shows that root in this image has no library tables, so its warning counts differ from the course's. The course's own `labs.yml` workflow also starts its containers with `docker run`, through `kicad.sh`, for the same reason.

A2. Part 1: Add the workflow

It would be green. A step fails only when its command exits with a status other than 0, and the TODOs are comments, so every stage of `build.sh` succeeds. KiBot finds no outputs listed in `ci/board.kibot.yaml` and exits 0 as well. The artifact has no drill file, BOM, position file, renders, copper plots or KiBot outputs in it, which is why the lab's check looks at the files the build made as well as at its exit status.

A3. Part 2: Keep KiCad's own files out of git

The cache was committed four minutes before the `.gitignore` that lists it, and a `.gitignore` only stops git from adding files; it doesn't affect a file git already tracks. Removing it from the index with `git rm --cached fp-info-cache` and committing would let the ignore rule apply.

A4. Part 3: Finish the fabrication files and review images

They're the test points, TP1 and TP2. Their footprint, `TestPoint_Pad_D1.5mm`, is a bare pad with nothing to place on it, and it carries the attribute "Exclude from position files" (`exclude_from_pos_files` in the board file), so the pick-and-place file skips it. The BOM comes from the schematic and lists them anyway.

A5. Part 4: Look at what the fab will get

You'd change the drill file's origin. The drill file and the Gerbers line up only through their coordinates. If the board's plot settings use the drill/place origin, the drill file needs `--drill-origin plot`, and if they don't, it needs `absolute`. Fabrication outputs shows this offset on the DIP dock.

A6. Part 5: Add KiBot's outputs

Because KiBot's DRC, with its default settings, doesn't fail on everything that `check_board.py` fails on, and because its KiCad is a different build. We tried KiBot's checks on Lab 04′s reference board through `kibot.sh`. With `erc: true` and `drc: true` under `preflight:`, KiBot fails only on errors, but KiCad reports a missing footprint as a parity warning. With R3 deleted from the board, KiBot printed `WARNING:(W145) (missing_footprint) Missing footprint R3 (10)` and exited 0, which is the green run that Outputs in CI warns about. With its `warnings_as_errors: true` option it fails on every warning instead, and the unchanged board failed on its 32 silkscreen warnings (`DRC warnings: 32, promoted as errors`, exit status 5). `check_board.py` failed the board without R3 (`FAIL  parity.warning: 1 (want 0)`) and passed the unchanged one. KiBot's `filters` option can turn chosen kinds of warning into errors, one kind at a time, so you could make KiBot fail on nearly the same things, but the course would then have two definitions of a passing board to keep in step.

The other reason is that KiBot's image carries KiCad's Ubuntu build of 10.0.5, with the 10.0.4 symbol and footprint libraries, and warning counts depend on the libraries. Every count on the course's pages comes from `kicad/kicad:10.0.5`, so the check that passes or fails a board stays in that image, and KiBot's step runs only after it has passed.

A7. Part 5: Add KiBot's outputs

It would have listed the part like any other. The course's self-test marks J4 DNP in our schematic and on our board, and runs the check twice. With `dnf_filter` naming the filter, the ibom listed 14 parts in 10 rows and left J4 out. With that line taken out, it listed J4 again, and the check's `ibom` row failed with `pmod-ibom.html lists J4, which the board marks Do not populate; give the ibom output a filter for DNP parts (dnf_filter)`. The BOM that `kicad-cli` made from the schematic listed all 15 parts both times.

A8. Part 5: Add KiBot's outputs

They're the sizes of the drills rather than of the finished holes. Copper is plated onto the wall of a plated hole after it's drilled, which makes the hole smaller, so the drill has to be larger than the hole in the footprint. KiBot allows 0.1 mm for that on every plated hole, and the report says so under "Important sizes": "The above values are real drill sizes, they add 0.1 mm (4 mils) to plated holes (PTH)". Every hole on our board is plated, so all three tools are 0.1 mm larger than the holes.

A9. Part 6: Break it on purpose

No. The editor runs the parity test only when its “Test for parity between PCB and schematic” box is ticked, as you did in Lab 03, Part 4. `ci/build.sh check` runs `check_board.py`, which always passes `--schematic-parity` to `kicad-cli pcb drc` and fails on any parity item, warnings included.

A10. Finishing

Look first at the renders and the copper plots in `review/`, and compare them with the ones from the commit before. Moving one part changes its position, the tracks rerouted to reach it and the zone fills around it, and in the board file’s text diff, the lines that record those changes are hard to read as one change. KiCad files in git shows a “minor” commit to Tiny Tapeout’s demo board that changed 5,770 lines.

A11. Finishing

Not necessarily. Every Gerber, the job file and the drill file carry the date and time of the export (Fabrication outputs), so compare them with those lines removed before you conclude anything.

VI 6. Review and ordering

24 Reviewing and ordering a board . . 167
24 Mistakes the checks don't look for 167
24 The checklist 167
24 Fabricator rules and KiCad's libraries 168
24 Ordering 168
24 Duties and tax 169
24 Check yourself 169
24 Answers 170

25 Lab 06: Review a seeded board 171
25 Before you start 171
25 Part 1: What the checks say 171
25 Part 2: Review it against the checklist 172
25 Part 3: Get a quote for your own board 174
25 Finishing 175
25 Going further (optional) 176
25 Answers 176

24. Reviewing and ordering a board

This page answers: *what does a review find that ERC and DRC can't, and once a board passes both, what does it cost to have it made and shipped to Toronto, and how long does that take?*

By the end of Lab 05 your board passes every check you've run on it, and CI builds its fabrication files on every push. ERC, DRC and parity compare the board with its own schematic and with the rules saved in it, so a mistake that is in the schematic or in the rules passes all three. The Pmod check also compares the schematic with the project's pin list, but it can only check what the list asks for, so a board can pass every check and still be wrong. The first half of this page is about the review, in which a person checks the board against what it has to do. The second half is about ordering the board. It covers what you send a fabricator, the prices and build times that three fabricators publish, and the duty and tax that Canada charges when the boards arrive. In Lab 06 you review a board that we seeded with eight faults, and then you get a fabricator's quote and DFM report for your own board, without ordering it.

Mistakes the checks don't look for

Tiny Tapeout's breakout repository records two mistakes that ended up on manufactured boards:

- The TT07 breakout "has the ASIC project MUX `reset_n` and `increment` wires crossed and is thus mislabelled."
- The production run of the TT06 demo boards "had very low valued resistances which, if you wish to interact with analog projects, had to be removed or replaced."

ERC, DRC and schematic parity don't look for either kind of mistake. When two signals are swapped in the schematic, they're swapped the same way on the board, so the board still matches its schematic. A resistor's value is a field on its symbol, and none of those three checks compares it with what the circuit needs. A person finds mistakes like these by reading the schematic against the chip's pin list, the datasheets of the parts, and the supplies on either side of every connector, which together say what the board has to do.

The team doesn't yet have a member who has reviewed boards before. Until it does, the review is the course checklist below, which you first use on the seeded board in Lab 06, and a second member reads every board's `BRINGUP.md` before a kit is powered from it (Bringing up a chip).

The checklist

The course checklist is the list that Lab 06 has you review against. Each line comes from something earlier in the course, and you can check each one in KiCad, with the datasheets, the project's pin list and the board's `BRINGUP.md` open beside it.

1. **Footprints against datasheets.** Every footprint matches the package of the part you'll actually buy, including the package variant, because one part number often comes in several packages. You checked a footprint against a datasheet's package drawing in Lab 02.
2. **Pin 1.** The pin-1 marks on the silkscreen and on the fabrication layer agree with the footprint's pad 1 and with the datasheet's drawing.
3. **Connections against the pin list.** Every line goes where the project's pin list says, and a UART's TX goes to the other side's RX. Lab 03's Pmod check compares the schematic with the pin list, and it checks that each `uio` line has a switch, but not which switch it is.
4. **Decoupling.** Each capacitor is on the rail it was meant for and close to the pin it serves.
5. **I/O voltages.** On the demo board, Tiny Tapeout's SKY130 chips run their I/O at 3.3 V. The GPIO spec says "The input pins are not 5V tolerant", and gives each output's drive as 4 mA. Every line that crosses from another supply goes through a buffer that's powered from the kit's 3.3 V and rated for partial power down (Ioff). With the kit off, at most 10 µA reaches the chip's pin, under the 100 µA that SkyWater's I/O guide allows for a pin driven past its supply. On each line, the part that sends the signal (the dongle on a line into the chip, the chip on a line out) is on the buffer's input, and the

buffer's output goes through a series resistor of 330 Ω to 8.2 kΩ on a line into the chip, or 1 kΩ to 5.1 kΩ on a line out to the dongle (Protecting the chip's I/O).

6. **ESD.** Every connector that an outside cable plugs into, such as the dongle header, has an ESD part from each of its signal pins to ground. The demo board has 24 TVS diodes. Two are on its USB data lines, and the other 22 are on lines from the breakout's connector. Of those 22 lines, 14 carry the chip's analog pads and 8 are grounds. None of the 24 is on a digital project pin, so the pins your Pmod reaches have only the chip's own protection (Supplies, back-powering and ESD).
7. **Back-powering.** `BRINGUP.md`'s table of the lines that leave the board says what happens on each line when one side is powered and the other isn't. Nothing on the board can drive a pin of an unpowered chip. The plan's steps plug in anything with a supply of its own only once the kit is up, and unplug it before the kit is switched off.
8. **Test points.** Every net that `BRINGUP.md` measures has a place to put a probe. The SKY130 breakout, for comparison, has four test points, on its reset, clock, `uio1` and `uin0` lines.
9. **Fabricator rules.** Silkscreen lines, text, annular rings and clearances meet the rules of the fabricator you'll order from, which may be stricter than KiCad's libraries (next section).
10. **Mechanical fit.** The Pmod connectors are 22.86 mm apart, as Tiny Tapeout's Pmod library places them, and nothing overhangs the demo board where it would hit something.

Write each finding down with the number of the line it breaks and the part or file it's in, so that the designer can go straight to it.

Fabricator rules and KiCad's libraries

Every fabricator publishes the smallest sizes it can make. The table gives them for an ordinary 2-layer board at three fabricators, as their pages read on November 16, 2026, with the numbers from KiCad's own library convention (KLC) for comparison:

	Trace and space	Smallest drill	Annular ring	Silkscreen line
JLCPCB	0.10 / 0.10 mm	0.15 mm	0.18 mm minimum on 2 layers	at least 0.15 mm
PCBWay	0.1 / 0.1 mm	0.15 mm	0.15 mm	
OSH Park, 2 layers	0.1524 / 0.1524 mm (6 mil)	0.254 mm (10 mil)	0.127 mm (5 mil)	
KLC (KiCad's footprints)			at least 0.15 mm (rule F7.5)	nominally 0.12 mm (rule F5.1)

JLCPCB also gives a minimum text height of 1.0 mm, and a minimum of 0.15 mm between a pad and the silkscreen. KiCad's numbers differ from the fabricators' in the last two columns. KLC's minimum annular ring of 0.15 mm is under JLCPCB's 2-layer minimum of 0.18 mm, and KLC's nominal silkscreen line of 0.12 mm is under JLCPCB's minimum of 0.15 mm. KiCad's own footprints are drawn to KLC, which is written for a library that has to work for everyone rather than for any one fabricator, so a board drawn with them can still break one fabricator's rules.

We counted the silkscreen on Tiny Tapeout's DIP dock. In its footprints, 441 lines, arcs and circles are 0.12 mm wide. Of the 139 pieces of text placed on the board itself, 98 have 0.14 mm strokes, and 38 are less than 1.0 mm tall. All of those are below JLCPCB's published minimums, but the board's own DRC reports no errors, because the rules saved in the board ask only for text at least 0.8 mm high and 0.08 mm thick, and have no rule for the width of a silkscreen line.

A DFM (design for manufacture) report is a fabricator's own check of your files, and it's where you'd find out whether the fabricator prints a 0.12 mm line anyway, thickens it or flags it. That's one reason Lab 06 has you get one. We haven't been able to get a DFM report ourselves (Lab 06 says why), so we can't tell you which of those JLCPCB does. If your board's design rules have your fabricator's numbers in them, DRC checks the board against those numbers before you upload anything.

Ordering

Ordering a bare board means uploading a zip of the Gerbers and the drill file (Fabrication outputs) to a fabricator's quote page, choosing the options (layers, thickness, finish, colour and quantity), and reading the price, the build time and the shipping choices that the page gives you. Lab 06 asks you to record those,

and the DFM report as well. We haven't checked whether each fabricator asks you to make an account before it shows a price, and Lab 06 has you record that too.

Prices change, so treat these as what the fabricators' own pages said on November 16, 2026, in US dollars:

- JLCPCB advertises "From $2.00 / 5 pcs" with a "Build Time: 24 hours", and its comparison page gives "From $2.00" for 5 boards of 1 to 4 layers at 100 × 100 mm.
- PCBWay advertises "$ 5.00 10 pcs 1-2 layer", with "Build Time:24 hours".
- OSH Park charges $5 per square inch for a set of 3 two-layer boards, made in 9 to 12 calendar days, with ENIG finish, 1.6 mm thick, and "shipped for free to anywhere in the world". Its Super Swift service is $10 per square inch, in 4 to 5 business days.

OSH Park's price depends only on the board's area, so you can work it out yourself. A board that spans the demo board's three I/O Pmod sockets is about 61 mm wide, which is two 22.86 mm spacings plus the width of one connector. If it's 30 mm tall, it's 2.8 square inches, and three of them cost about $14, delivered. The DIP dock, at 85.05 × 85.05 mm, is 11.2 square inches, so three of them cost $56.06.

The build times don't include shipping. JLCPCB's shipping table is an image that we couldn't read, so we have no confirmed transit time or shipping price to Canada. From the build times and OSH Park's lead time, we'd expect one to three weeks between ordering and holding the boards. The course doesn't require you to order a board. If you want to, the capstone's Going further section suggests ordering with your cohort, after a second person has read your `BRINGUP.md`.

Assembly by the fabricator is optional in PCB101, and most course boards are meant to be soldered by hand (Parts you can solder by hand). If you do want it, JLCPCB's economic assembly has an $8.18 setup fee, $1.53 for the stencil, $3.07 to load each "extended" part and $0.0016 per solder joint (its price page, updated October 22, 2026). PCBWay advertises "Only $ 29 in total for 1-20pcs assembly".

Two Toronto companies also make boards. Candor Industries, at 125 Martin Ross Avenue, makes quick-turn, multilayer, impedance-controlled and RF boards. Bittele Electronics has its headquarters in Toronto and its production in Markham, and it does fabrication, parts and assembly. Neither publishes prices or a student programme. Candor quotes on request, and Bittele quotes through its online ordering system.

Duties and tax

Canada charges no duty on bare printed circuit boards. In the CBSA's 2026 Customs Tariff, tariff item 8534.00.00, "Printed circuits", has a most-favoured-nation rate of "Free". The boards can still be taxed, and whether they are depends on how they're shipped and what the shipment is worth. These are the CBSA's rules for low-value shipments, from its page modified 2025-11-25:

Shipment	Duty and tax free up to	Above that
Courier, from the US or Mexico	CA$40	duty free up to CA$150, but taxes apply; above CA$150, both
Courier, from any other country	CA$20	duties and taxes apply
Mail, from any country	CA$20	duties and taxes apply

JLCPCB's own page says "All duties and taxes for international shipments are the responsibility of the customer" and lists Canada's threshold as 20 CAD (updated November 10, 2026). The tax in Ontario is the 13 % HST. So a courier order of boards from China worth more than CA$20 pays 13 % HST on arrival and no duty on the boards themselves. The courier may also charge a brokerage fee for collecting the tax, which we haven't researched. The thresholds are per shipment, so if a cohort pools its boards into one order, the total value of that order is what counts.

Check yourself

1. A board passes ERC, DRC and parity with no errors, but the chip's UART TX is wired to the TX pin of the dongle header. Which check should have caught it?

   Answer: A1, in Answers at the end.

2. Your board uses KiCad's library footprints, and you're ordering from JLCPCB. The footprints' silkscreen is drawn with 0.12 mm lines. What do you do?

   Answer: A2, in Answers at the end.

3. A courier delivers a CA$35 order of bare boards from China to you in Toronto. What do you pay at the border, before any brokerage fee?

   Answer: A3, in Answers at the end.

4. Without opening OSH Park's site, what would three copies of a 50 × 50 mm 2-layer board cost there?

   Answer: A4, in Answers at the end.

Next: Lab 06: Review a seeded board.

Answers

A1. Check yourself

ERC, DRC and parity can't catch it. The schematic and the board agree with each other, so parity is clean, and no rule in the board says which pin should be which. Lab 03's Pmod check catches it, because it compares the schematic with SimProc's pin list. When we made a board with TX and RX swapped this way, the check failed its `ui[0]` and `uo[0]` rows. In a review, it breaks line 3 of the checklist, connections against the pin list.

A2. Check yourself

The footprints are right by KiCad's convention, but 0.12 mm is below JLCPCB's published minimum line width of 0.15 mm. Read what JLCPCB's DFM report says about your upload. If it flags the silkscreen, widen the lines in a copy of the footprint, or accept that JLCPCB may print them poorly.

A3. Check yourself

You pay no duty, because printed circuits are tariff-free. The order is above the CA$20 courier threshold for a country other than the US or Mexico, though, so the 13 % HST applies, and 13 % of CA$35 is CA$4.55.

A4. Check yourself

50 mm is 1.97 inches, so the board is 3.88 square inches, and at $5 per square inch for a set of three that's about $19.38 (in US dollars, shipped).

25. Lab 06: Review a seeded board

In this lab you review a board you didn't draw. It's a SimProc Pmod like yours, with its bring-up plan, and we planted eight faults in it. The checks you already have look for fewer than half of them. You'll run those checks first, then go through the course checklist one line at a time against the board and its plan, and write each fault you find in `REVIEW.md`. Then you get a fabricator's quote for your own board, without ordering it, and record the price, the build time and the fabricator's own check of your files in `QUOTE.md`.

The lab's files are in `labs/lab-06-review-a-seeded-board/` in the course repo.

Before you start

- Finish Lab 05. Part 3 of this lab uploads the Gerbers that your Lab 05 workflow built.
- Read Reviewing and ordering a board, row 11 of the course path. Its ten-line checklist is the one you review against, and `REVIEW.md` asks for the number of the line each fault breaks.
- Don't open `labs/lab-06-review-a-seeded-board/solutions/` until you've finished. It lists the faults.

The lab takes about 1 hour 15 minutes, roughly 10 minutes for Part 1, 40 for Part 2 and 25 for Part 3. Those times are our estimates. We did time the checks themselves, with the container limited to one CPU. On a copy of the seeded board, the board check took 19 s and Lab 04's check took 20 s. The Lab 06 check took under a second, because it only reads your two files and your board.

Part 1: What the checks say

What do the checks you already have find on this board?

1. Copy the seeded board out of the course repo, so that KiCad's own files and anything you change stay out of it, and copy the two templates into your board repository:

```
cp -r ~/pcb101/labs/lab-06-review-a-seeded-board/board ~/seeded-review
cd ~/simproc-pmod
cp    ~/pcb101/labs/lab-06-review-a-seeded-board/REVIEW.md    ~/pcb101/labs/lab-06-
review-a-seeded-board/QUOTE.md .
```

2. Will the board check pass on a board with eight faults in it? Write down your guess, then run it on the copy:

```
cd ~/seeded-review
~/pcb101/labs/common/kicad.sh python3 /w/labs/common/check_board.py .
```

   You should see it pass, with the same 32 DRC warnings as Lab 04's reference board:

```
                errors  warnings  excluded
  ERC                0         0         0
  DRC                0        32         0
  unconnected        0         0         0
  parity             0         0         0

  DRC warnings by type: silk_edge_clearance 24, silk_over_copper 6, silk_overlap 2

PASS  pmod.kicad_pro
```

3. Now run Lab 04's check, which also compares the board with SimProc's pin list:

```
~/pcb101/labs/lab-04-protect-and-plan/check.sh
```

You should see the board check and the bring-up plan check pass. In the Pmod check, the connectors row fails, the power row passes, and the pins row says that the pins aren't checked until the connectors pass:

```
  FAIL  connectors   J1 (102.00, 56.35), J2 (124.86, 56.35), J3 (148.22, 56.35)
                     their spacing is 22.86 and 23.36 mm; it must be 22.86 mm
   ok     power          +3V3 on pins 6 and 12, GND on pins 5 and 11, of every
Pmod connector
  FAIL  pins         not checked until the connectors pass
...
FAIL  pmod.kicad_pro: board check PASS; 1 of 3 Pmod checks ok; bring-up plan PASS
```

That's your first finding. Write it as row 1 of the table in `REVIEW.md`: `J3` under Where, `10` under Line (mechanical fit), what's wrong, and what to change.

4. Two of the three checks passed, and the third checked only the connectors and the power pins. What does that tell you about the rest of the board?

   Answer: A1, in Answers at the end.

Part 2: Review it against the checklist

Which faults does a person find that the checks don't?

1. Open `~/seeded-review/pmod.kicad_pro` in KiCad, and open both the Schematic Editor and the PCB Editor. Keep the checklist from Reviewing and ordering a board, SimProc's pin list (`simproc.json`) and the board's `BRINGUP.md` open beside them.

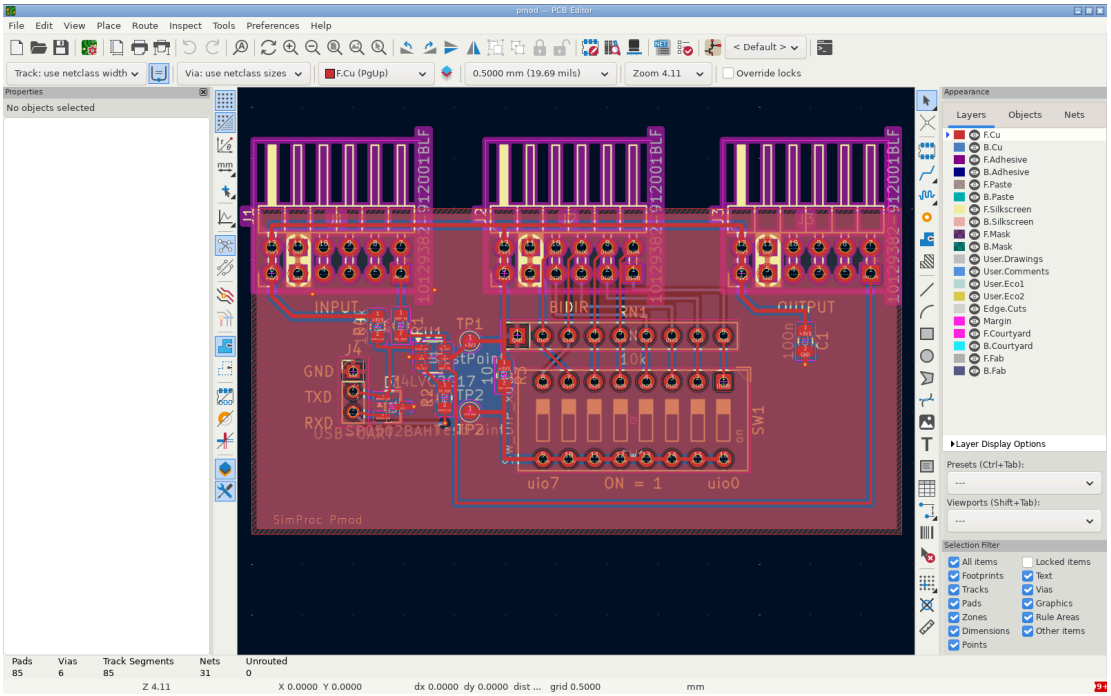


Figure 25.1: The seeded SimProc Pmod in the PCB Editor

The seeded board as it opens in KiCad 10.0.5's PCB Editor. It has the same parts as your Lab 04 board, in much the same places.

2. Go through the checklist one line at a time. For each line, look at every part on the board and every step of the plan that the line applies to. Here is where to look in KiCad:

- For a footprint (line 1), select the part on the board. The status bar along the bottom names its footprint in full, and the Properties panel has it as Library Link; the name gives the package and its pitch. Compare it with what will plug in or be soldered there.

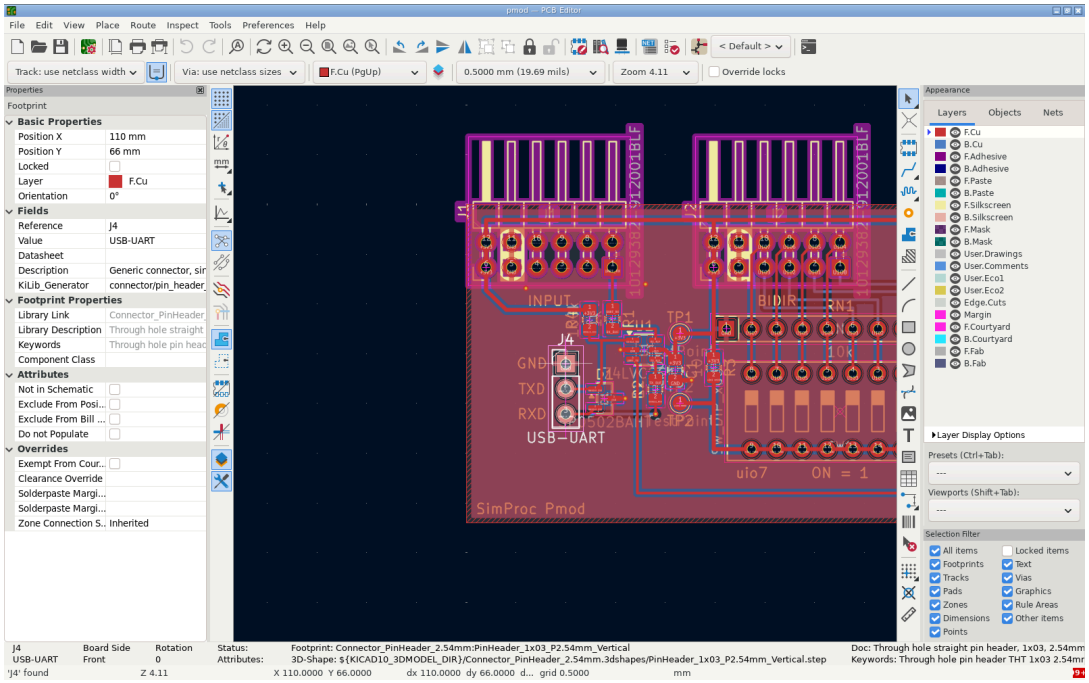


Figure 25.2: J4 selected on Lab 04′s reference board, with its footprint named in the status bar

J4 selected on Lab 04′s reference board (Ctrl+F, then J4), in KiCad 10.0.5. The status bar reads `Footprint: Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical`*; Footprint Properties (E) gives the same name as its Library link.*

- For connections (line 3), click a net in the Schematic Editor or a track in the PCB Editor to highlight it, follow it from the Pmod connector to every part it reaches, and compare with the pin list and with the silkscreen's labels.
- For decoupling (line 4), find each capacitor in the PCB Editor and the supply pin it serves, and see how far apart they are.
- For I/O voltages (line 5), follow each UART line between J4 and the Pmod connectors. Work out which end sends the signal, then check that this end reaches U1′s input, that U1′s output goes through a series resistor to the other end, and that the resistor's value is in the range that checklist line 5 gives for a line in that direction. U1′s symbol numbers its signal pins but doesn't name them, so use the Pin Functions table in its datasheet to find which pins are inputs.
- For ESD (line 6), and for any part with more than two pins (line 1 again, for the footprint's pad numbers), compare which pin goes where with the part's datasheet, and with the symbol's pin names where it has them. The symbol's Datasheet field links to a document when it's filled in. On some symbols, such as SW1′s, it's empty. U1′s links to a TI document, scyt129e, rather than to the SN74LVC2G17 datasheet (SCES381), so search for SCES381 yourself.
- For the plan (lines 7 and 8), read `BRINGUP.md` step by step against the board. Check the order in which things are plugged in and switched off, and whether every point it measures is on the board.
- For the fabricator's rules (line 9), open File > Board Setup, look under Design Rules > Constraints, and compare the numbers there with the fabricator's numbers on the checklist page. On your own board they're JLCPCB's, because Lab 03′s template set them:

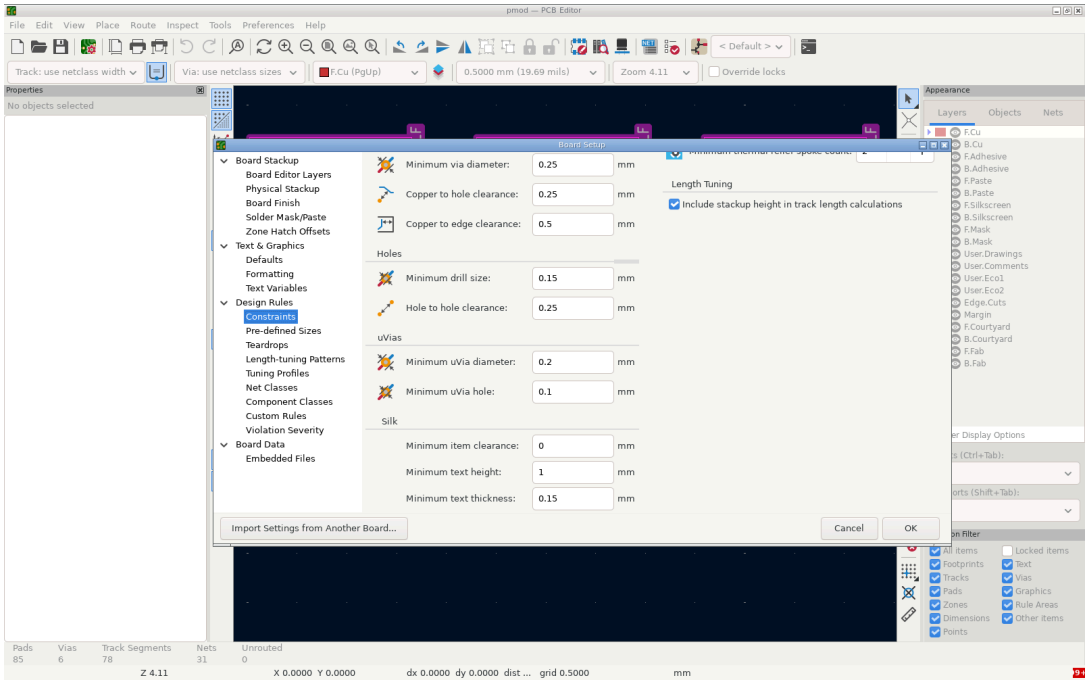


Figure 25.3: Board Setup's Constraints page, showing the Silk rules

Board Setup > Design Rules > Constraints on Lab 04's reference board, scrolled to the Silk rules. The minimum text height is 1 mm and the minimum text thickness is 0.15 mm, which are JLCPCB's published limits.

3. For each fault you find, add a row to `REVIEW.md`: under Where, the reference designator of the part (or the file, such as `BRINGUP.md` or `pmod.kicad_pro`); under Line, the number of the checklist line it breaks; then what's wrong and what to change. Write it for the designer, who wasn't there when you found it.
4. Run the Lab 06 check from your board repository:

```
cd ~/simproc-pmod
~/pcb101/labs/lab-06-review-a-seeded-board/check.sh
```

It prints a line for each row of your table and says whether that row finds a planted fault. A row finds one when its Where names the right part or file and its Line is the line that the fault breaks, and each row can find at most one fault. The first two rows of our reference review look like this:

```
REVIEW.md
  row 1   J4                          line 1   finds a planted fault
  row 2   SW1                         line 3   finds a planted fault
```

You should see at least five rows that find a planted fault, and `ok  faults  5 of 8 planted faults found (5 needed)` or better. Until you've filled in `QUOTE.md`, the last line still says FAIL. The check holds only hashes of the answers, so it tells you whether a row finds a fault but not which fault it finds. If a row you're sure of doesn't find one, check that its Where names the part's reference designator or the file the fault is in, as the template asks, and read the checklist line again.

Part 3: Get a quote for your own board

What would your board cost to make, how long would it take, and what does the fabricator's own check say about your files?

1. On GitHub, open the last green run of your Lab 05 workflow and download its `board-outputs` artifact. Unzip it, and zip the contents of `fab/` (the Gerbers, the job file and the drill file) on their own, which is what a fabricator's upload page expects:

```
cd ~/Downloads
unzip board-outputs.zip -d board-outputs
cd board-outputs/fab && zip ../../simproc-pmod-fab.zip *
```

Note the commit the run was for; `QUOTE.md` asks for its hash.

2. Before you upload, write down the board size and the number of layers you expect the quote page to read from your files. Our board's outline is 64 × 32 mm on 2 layers.
3. Choose a fabricator from Reviewing and ordering a board (JLCPCB, PCBWay or OSH Park), upload the zip on its quote page, and leave the options at a plain 2-layer board unless you have a reason to change one. Don't order. Read the price, the build time, the shipping choices to Toronto, and the result of the fabricator's file check (the DFM report), and copy each into `QUOTE.md`, with the DFM warnings word for word. Record whether the site asked you to make an account before it showed a price; we haven't checked that for any of them.

   We couldn't do this part ourselves, because the machine we wrote the course on doesn't upload files to outside sites, so there is no quote page or DFM report from us here. The reference `QUOTE.md` in `solutions/` shows the format, with JLCPCB's advertised "From $2.00 / 5 pcs" in place of a real quote.
4. Did the quote page read the size and layer count you predicted? If it didn't, find out why before you go on, because the fabricator will make the board it read from your files.
5. Run the Lab 06 check again. You should see the quote's three rows `ok`, with your own board's numbers in the last:

```
QUOTE.md
ok    fields    every field filled in, no TODO
ok    values    date, price, build time, quantity and commit all readable
ok    board     2 layers and 64 x 32 mm, as your board has
```

The check compares Layers and Board size with your board's `.kicad_pcb`, so a quote for some other board fails the `board` row, and so does a quote in which the fabricator misread your files.

Finishing

You're done when `~/pcb101/labs/lab-06-review-a-seeded-board/check.sh`, run in your own repository, ends with `PASS  review PASS (5 of 8 planted faults found); quote PASS` or a higher count, and `REVIEW.md` and `QUOTE.md` are committed and pushed.

Questions:

1. In `~/seeded-review`, untick Locked for J3 in the Properties panel, put it back at 22.86 mm from J2 (X 147.72 mm) the way you placed it in Lab 03, lock it again, save, and run Lab 04's check there again. The check now passes every `uio` row. Look at the pin numbers it prints for `uio[6]` and `uio[7]`. What does it check about those lines, and what doesn't it?

   Answer: A2, in Answers at the end.
2. Of the faults you found, which would Lab 04's check have reported if J3 had been in the right place?

   Answer: A3, in Answers at the end.
3. The seeded `BRINGUP.md` switches the kit off before the dongle comes off J4. How much current can the dongle push into the chip while the kit is off, and which checklist line does the order break?

   Answer: A4, in Answers at the end.
4. The seeded board's "SimProc Pmod" legend is 0.8 mm high, and none of the board's 32 DRC warnings is about it. Why didn't DRC warn about it, when your own board's DRC would?

   Answer: A5, in Answers at the end.

Submit: nothing for this lab. Keep `REVIEW.md` and `QUOTE.md` in your repository.

Going further (optional)

- Review a classmate's Lab 04 board in a pull request on their repository, against the same checklist, with the renders and plots their workflow built. This needs someone working at the same pace, which is why it isn't a required part.
- Get quotes from a second and a third fabricator for the same zip, and compare their DFM reports. The files are the same, so where one fabricator flags something another doesn't, the difference is in their processes.

Next on the Course path: read Boards for chips: carriers, motherboards and characterization boards, then do the Capstone: A board for a team chip.

Answers

A1. Part 1: What the checks say

It tells you very little. The board check compares the board with its own schematic and with the rules saved in the board, so a fault that is in the schematic, or in the rules, passes it. The bring-up plan check only looks for a filled-in plan with an expected value at every step; it can't tell whether the steps are safe. The rest is for a person to find in a review. Reviewing and ordering a board starts with two of Tiny Tapeout's own mistakes that ended up on manufactured boards, and ERC, DRC and parity don't look for either kind.

A2. Finishing

It checks that each `uio` line has a switch to the 3.3 V supply and a pull-down to GND, but not which switch it is. On the seeded board `uio[6]` goes to switch 8 (`pins 8-9`) and `uio[7]` to switch 7 (`pins 7-10`), the reverse of the other six and of the silkscreen. Anyone who sets the switches by the legend gets the wrong bit period whenever those two bits differ. Tiny Tapeout's TT07 breakout had the same kind of fault, two lines crossed and so mislabelled.

A3. Finishing

Two of them, U1 and C1. U1′s second buffer, turned round, fails the `uo[0]` row with "U1 (74LVC2G17) is turned round on this line: its output, 2Y (pin 4), faces J3 pin 1, and its input, 2A (pin 3), faces J4 pin 3 (the dongle's RXD)". With the kit on, U1′s output would drive the same line as SimProc's TX, and nothing would drive the dongle's RXD. C1′s distance from U1 fails the `ui[0]` row with "no decoupling at U1′s VCC: the nearest capacitor from +3V3 to GND, C1, has its nearest pad 35.50 mm from U1 pin 5, more than 3 mm". The `uo[0]` row stops at the turned-round buffer before it looks for a capacitor, so only `ui[0]` reports C1. With J3 put back, those two rows are the only ones that fail, and the last line is `FAIL  pmod.kicad_pro: board check PASS; 15 of 17 Pmod checks ok; bring-up plan PASS`. Counting J3 itself, the checks find three of the eight faults, and the other five need a person to find them.

A4. Finishing

At most 10 µA, the Ioff of U1, which sits between the dongle and the chip and is unpowered while the kit is off. That's under the 100 µA that SkyWater allows (checklist line 5), so U1 keeps the current into the chip's pins within that limit in either order. The order breaks line 7, which asks for anything with a supply of its own, such as the dongle, to be unplugged before the kit is switched off. Until the dongle comes off, it drives U1′s unpowered input. It also puts up to 3.3 V / 100 kΩ = 33 µA into the kit's +3V3 through R4, the 100 kΩ pull-up from its TXD to +3V3. Your own `BRINGUP.md` from Lab 04 unplugs the dongle first, so that none of this current flows.

A5. Finishing

DRC checks the board against the rules saved in the board, and the seeded board's Board Setup lowers the minimum text height and thickness to 0.8 mm and 0.1 mm, below JLCPCB's 1.0 mm and 0.15 mm. With your template's rules, the same text gives a `text_height` and a `text_thickness` warning.

VII Capstone

26 Boards for chips: carriers, motherboards and characterization boards . . . 179

26 A board for an analog pin . . . 179
26 A motherboard for the breakout . . . 180
26 Boards for chips from other shuttles . . . 181
26 Check yourself . . . 182
26 Answers . . . 182

27 Capstone: A board for a team chip . 183

27 Before you start . . . 183
27 The two briefs . . . 183
27 Part 1: Requirements first . . . 184
27 Part 2: The board . . . 185
27 Part 3: Decisions and the bring-up plan . . . 187
27 Part 4: Review it and push . . . 187
27 Finishing . . . 187
27 Going further (optional) . . . 188
27 Answers . . . 188

26. Boards for chips: carriers, motherboards and characterization boards

This page answers: *the Pmod you drew is one kind of board a chip needs. What are the others, and what does each ask of you that a Pmod didn't?*

The Pmod you drew in Labs 03 and 04 plugs into Tiny Tapeout's demo board, and the demo board does a lot of the work for it. The demo board powers the chip, selects the project through the multiplexer on the chip, usually provides the clock, and reaches every project pin through its DIP switches, its 7-segment display, its Pmods and its RP2350. Your Pmod adds only the hardware that one project needs on top of all that, which is why it made a good first board. The other boards that a team ends up needing take over more of the demo board's jobs, or carry a different chip altogether. This page goes through them in order, from the kind closest to what you've done to the kind furthest from it, so that when the team needs one you'll know what it involves. The capstone's two briefs are a Pmod and the first kind below, a board that drives one of the chip's analog pins.

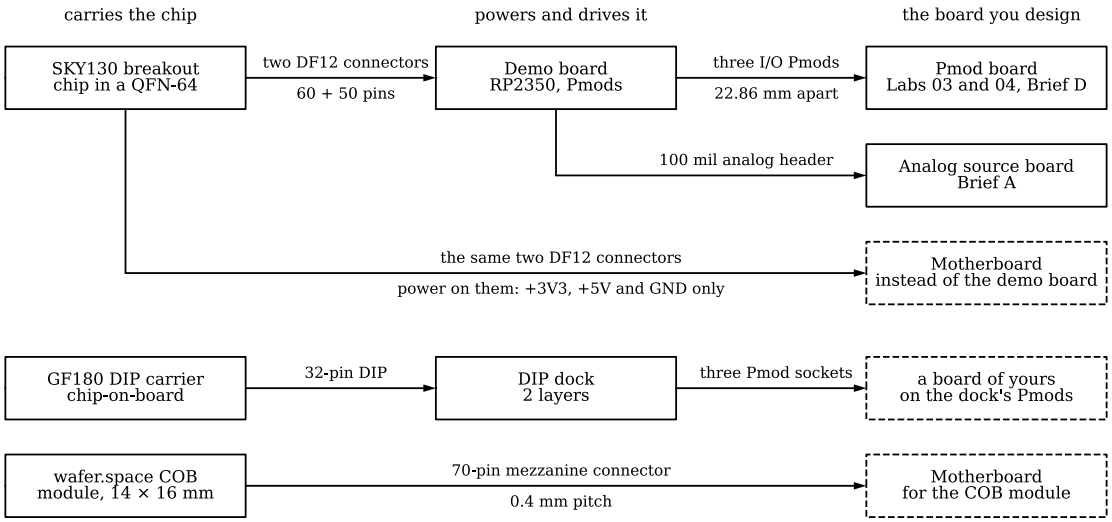


Figure 26.1: A diagram of where each kind of board attaches: the SKY130 breakout to the demo board through two DF12 connectors, the demo board to a Pmod board and to an analog source board, the breakout to a motherboard, a GF180 DIP carrier to the DIP dock, and a wafer.space chip-on-board module to a motherboard through a 70-pin mezzanine connector

Where each kind of board attaches, from the chip outwards. The solid boxes are boards that exist, or that you design in this course; the dashed ones are boards a team could design next. Made by `scripts/figures/board_kinds.py`.

A board for an analog pin

CrossWalk, one of the team's designs, is an 8-bit SAR ADC with two analog pins (its `info.yaml` says `analog_pins: 2` and `clock_hz: 4000000`). Testing it means driving an analog pin with a voltage you know, cleanly enough that the converter's errors aren't hidden under the source's.

On the demo board, each analog line comes out on a 100 mil header and is "pulled down with a large 10 MegaOhm resistance". The SKY130 chips have 12 pins for analog, and a project may use up to 6 of them (demo board guide). Inside a SKY130 chip, each of a project's analog pins is connected to its pad

through an analog switch, `tt_asw_3v3`, and Tiny Tapeout's analog spec gives the limits of that path as "Resistance: < 500 ohm", "Capacitance: < 5 pf" and "Max current: 4 mA".

A source board's design note starts from those numbers. The two examples of the arithmetic below are ours rather than Tiny Tapeout's.

- The 10 MΩ pull-down and your source's output resistance form a voltage divider. A source with 10 kΩ of output resistance loses 10 kΩ / 10 MΩ = 0.1 % of its voltage, which is a quarter of one step of an 8-bit converter (1/256 is 0.39 %). A buffered source has an output resistance of a few ohms, so it loses a millionth of its voltage or less, far less than a step.
- The path's resistance and capacitance set a time constant. With only the path's own 500 Ω and 5 pF it's 2.5 ns, but a 10 kΩ source charging the same 5 pF makes it 50 ns. Settling to within half a step of 8 bits (1/512) takes ln 512 ≈ 6.2 time constants, about 310 ns, which is longer than one 250 ns cycle of CrossWalk's 4 MHz clock. So if CrossWalk samples its input for a single clock cycle, a 10 kΩ source won't have settled by the end of it.

The capstone's Brief A is the simplest form of this board. It drives one analog pin from a known, filtered source, has a test point on the source and one on the pin, and comes with a design note that does the arithmetic above for your own parts.

A full characterization board, which measures how good the converter is, is a bigger design, and we've left it for PCB201, a second course that would follow this one. Such a board needs a test signal that is more linear than the converter it tests. Steve Arar's article on the sine-wave histogram method (All About Circuits, October 20, 2023) uses a sine for this and not a ramp, because "the linearity of the ramp signal produced by typical signal generators is limited to 8 to 10 bits", which is not much better than an 8-bit converter itself, while "we can filter out the harmonics of a sinusoidal signal". You drive the converter with a sine slightly larger than its full-scale range, so that "we can be sure that the input exercises all codes" (the article's example puts 1.1 V on a 1 V converter). The sine's frequency mustn't be a subharmonic of the sampling rate, and the example uses 390.3 Hz against 40 kHz, with 80,000 samples. You then count how often each code appears and divide each count by the count an ideal converter would give, and "Subtracting one from the bin counts therefore produces the DNL information."

The board also needs a voltage reference of its own. Horowitz and Hill describe a programmable laboratory source built around a DAC, and they write that in a circuit like that the voltage reference is usually the biggest source of noise. The ADR445 that they chose for it has about 2 µV peak-to-peak of noise from 0.1 to 10 Hz (*The Art of Electronics*, 3rd ed., chapter 13).

There's no CrossWalk silicon to test yet. CrossWalk is tagged for Tiny Tapeout's TTSKY26d shuttle, which closes on 2026-11-30. Its chips are expected on 2027-05-25, and the estimated shipping date is 2027-06-25 (chips page). Until then a board for CrossWalk is a design exercise, but if you design it now, while the chip is still being designed, the chip and the board can each still change to suit the other.

A motherboard for the breakout

For anyone making their own board, Tiny Tapeout's FAQ says "we recommend that you design your PCB to interface with our breakout board", and that the breakout "will provide you with 2 connectors where you get all of the signals of the chip." A board that mates with the breakout's two DF12 connectors takes the demo board's place, so it has to do all of the demo board's jobs: power the breakout, select the project through the chip's multiplexer, provide the clock and reset, and drive and read the pins.

It can't reach the chip's own supplies, though. We followed every power net of the SKY130 breakout (rev 3.2, `breakout-pcb` at 91a2bf1), and the only supplies on its two connectors are +3V3, +5V and ground. The 60-pin connector J1 has one +3V3 pin and 24 ground pins, and the 50-pin J3 has two +5V, two +3V3 and 13 ground. The chip's three rails stay on the breakout. `VCCD` (1.8 V) comes from the breakout's one fitted regulator through a 0 Ω link, and `VDD_IO` and `VDDA` come from +3V3 through 0 Ω links. So a motherboard can measure the breakout's total current on +3V3, and it can change `VDD_IO` and `VDDA` together by changing +3V3, but it can't measure or change one rail on its own. For that you'd need the rework that the breakout's silkscreen describes: taking off an 0805 link and fitting the header J2, which isn't fitted on the boards Tiny Tapeout ships (Bringing up a chip shows its empty pads). That rework is on the board that carries the kit's only chip, so if the team ever needs it, practise first on a breakout without a chip.

Drawing a motherboard is more work than drawing a Pmod, too. Tiny Tapeout's breakouts and demo board all have 4 copper layers, and the DF12 connectors have a 0.5 mm pitch, both the breakout's (their footprints are `HRS_DF12NB-60DS-0.5V_51` and `HRS_DF12NB-50DS-0.5V_51`) and the mating halves that the demo board carries and a motherboard would need. That's far finer than anything this course has

you solder (Parts you can solder by hand). And because the motherboard now selects the project, the multiplexer's control lines are yours to get right. Tiny Tapeout got two of them wrong on its own TT07 breakout, which, in the README's words, "has the ASIC project MUX `reset_n` and `increment` wires crossed and is thus mislabelled".

A motherboard also has to make the clock, which on the demo board is usually the microcontroller's job. On the RP2040 version of the demo board, the microcontroller could "generate the clock signal for your design, up to 50 MHz" (PCB spec). Tiny Tapeout's GPIO spec gives the official limits of the SKY130 pins as 33 MHz for an output and 66 MHz for an input, and it reports a worst-case round trip through the multiplexer of 20 ns, measured on TT3.5 silicon. A motherboard that clocks the chip itself, or reads back an output in the same clock cycle, has to stay inside those numbers.

Boards for chips from other shuttles

The team's chip designs so far are all on Tiny Tapeout's SKY130 and IHP shuttles, which deliver each chip in a QFN on a breakout. Other ways of getting a chip made deliver the die mounted on something else, or not mounted at all, and the board you'd design around it changes with that.

Tiny Tapeout's GF180 shuttles

These shuttles don't package the die. In the FAQ's words, they "have the die bonded to a PCB directly (called chip-on-board, or COB)". Tiny Tapeout's "TT GF180 CoB DIP Breakout" carries the die on a board in the shape of a 32-pin DIP, and brings out "all the I/O (8 in, 8 out and 8 bidirectional)", "the project clock and reset" and "the Tiny Tapeout project MUX, to select the design". It also has a CH32X035 microcontroller that can select designs and clock projects, though you can leave both jobs to the board that the breakout plugs into instead. That board is the DIP dock, whose fabrication files you went through in Fabrication outputs. Its README says it "is very simple, and designed using only two layers", and it has the same DIP switches, 7-segment display, Pmods, and manual clock and reset as the demo board. It's a board you could have drawn after Lab 03.

wafer.space

wafer.space runs shuttles on GF180MCU, and each slot you buy on one gets you 1,000 dies. On November 16, 2026, its site listed Run 3 at $8,000 for a 3.93 × 5.12 mm slot, with a purchase deadline of 9 December 2026 and parts shipping in Q2 2027. Chip-on-board packaging was a $1,500 add-on ($1.50 per die) that "Requires the default pad ring for wire bonding", and the site listed no QFN option.

wafer.space's chip-on-board repository (Apache-2.0) shows what the board side of that looks like. For a full slot, the die is bonded to a 14 × 16 mm breakout with 74 pads, following "the Tiny Tapeout convention of 74 pads", and the breakout plugs into a motherboard through a 70-pin mezzanine connector with a 0.4 mm pitch. The repository has example motherboards in KiCad.

The repository's table for Run 1 maps each bond pad on the die to a pad on the breakout and to a Tiny Tapeout function. Bond pad 0 goes to breakout pad 1, `ctrl_ena`; bond pads 9 to 16 go to breakout pads 10 to 17, `uo[0]` to `uo[7]`; bond pads 19 to 24 carry `analog[0]` to `analog[5]`; and bond pads 17 and 73 are VDD IO. The wire-bonding rules for Run 1 say "No wires may cross", "Wire angle must be not greater than 45°" and "Maximum wire length must be less than 3 mm". A wire must also be at least 1 mm long or twice the die thickness, whichever is greater, and "No solder mask is allowed in the area to be bonded". wafer.space calls wire bonding "a very manual process", because "there are no real exchange formats yet that tie the die layout to the breakout PCB". To help, it wrote a script that reads the die's GDS file and the breakout's KiCad file together and "walks pin-for-pin around the padframe" to match each pad on the die to its bond pad on the board.

ChipFoundry's chipIgnite

ChipFoundry's chipIgnite shuttles are on SkyWater 130. A tapeout costs "$14,950" for up to 15 mm² of die with "38 fully -configurable I/Os", and it comes with an evaluation board and the "Option of 100 QFN-packaged parts or Bare Die". The CI2612 shuttle tapes out on December 7, 2026, for delivery on May 25, 2027. A QFN from chipIgnite would need a board much like Tiny Tapeout's breakout, decoupling and all, and that's a board for PCB201 rather than this course.

A bare die of your own

Packaging a bare die locally is possible. The Toronto Nanofabrication Centre's January 2026 price list has a TPT HB16 wire bonder at $49 an hour for UofT users, and staff service at $60 an hour from 9 am to 5 pm. CMC Microsystems packages dies in DIP 40, CFP 24, CQFP 44, 80 and 120, and CPGA from 69 to 209 pins. It also wire-bonds bare dies directly to a substrate and does flip-chip, and "All of the above services are also accessible to non-subscribers", by quotation. Europractice publishes the bonding rules that an assembly house works to (pad size and pitch, wire length and angle, no crossing wires) in its ASIC Package Design Rules. Every page of that document is marked confidential, although it's posted publicly, so we link to it rather than copy from it. From die to package explains what a bond plan is.

None of these boards is a course exercise, because the team has no chip from these shuttles yet. If it gets one, a chip-on-board module will need a motherboard with a fine-pitch connector, a QFN will need a carrier with its own decoupling, and a bare die will need a package and a bond plan before it needs a board.

Check yourself

1. Your source board drives a CrossWalk input from a 47 kΩ potentiometer wiper with no buffer. Using the numbers on this page, what does the demo board's pull-down cost you?

   Answer: A1, in Answers at the end.

2. Someone suggests a motherboard for the SKY130 breakout so the team can measure `VCCD` on its own. What does the board file say about that?

   Answer: A2, in Answers at the end.

3. Why does Tiny Tapeout's GF180 chip sit on a DIP carrier, when the SKY130 chips come in a QFN?

   Answer: A3, in Answers at the end.

Next: Capstone: A board for a team chip.

Answers

A1. Check yourself

A pot's output resistance is largest with the wiper in the middle, where the two halves of the track are in parallel, so for a 47 kΩ pot it's about a quarter of 47 kΩ, roughly 12 kΩ. Against the 10 MΩ pull-down that loses about 0.12 % of the voltage, which is about a third of an 8-bit step, and the loss changes as you turn the knob. A buffer after the pot removes it.

A2. Check yourself

A motherboard can't do it with the breakout as it's built. The only supplies on the DF12 connectors are +3V3, +5V and ground, and `VCCD` stays on the breakout, fed from its regulator through a 0 Ω link, R14. To measure it you'd have to remove that link and fit J2 on the breakout, which is rework on the board that carries the kit's only chip.

A3. Check yourself

Tiny Tapeout's GF180 shuttles don't package the die. It's bonded straight to a board (chip-on-board), and Tiny Tapeout's carrier for it is a small board in the shape of a 32-pin DIP, which plugs into the DIP dock the way a DIP chip would.

27. Capstone: A board for a team chip

In the capstone you take one board for a team chip to the point where it could be ordered. You draw it, check it, review it and have your CI workflow build its fabrication files, and you write a design note that says why every part is there and a bring-up plan that says what to expect at every step. You choose one of two briefs. Brief D is a bring-up Pmod for a digital project, and if you built the SimProc Pmod in Labs 03 and 04, you can carry that board through. Brief A is a small source board that drives one analog pin of a chip from a known, filtered voltage. Everything goes in a repository of your own.

The capstone's files are in `labs/capstone-a-board-for-a-team-chip/` in the course repo.

Before you start

- Finish Lab 06. The capstone uses the board template, the protection rules, the CI workflow and the checklist from Labs 03 to 06.
- Read Boards for chips: carriers, motherboards and characterization boards, row 13 of the course path. Its section on a board for an analog pin has the arithmetic Brief A asks for.
- Decide on your brief, from the two below.

The capstone takes about 4 hours for Brief D on the SimProc board you already have: roughly 30 minutes for the requirements, 1 hour for the board's changes, 1 hour 30 minutes for the decisions and the bring-up plan, and 1 hour to review, commit and push. Those times are our estimates, but we did time the check itself, which took 30 s on our Brief D reference with one CPU. Brief A starts a new board, so expect it to take longer; we haven't timed it.

The two briefs

Brief D: a bring-up Pmod for a digital project

Brief D is for SimProc, Sentry, or a digital Tiny Tapeout project of your own. The board plugs into the demo board's I/O sockets and adds the hardware that the project needs and the demo board doesn't have, the way the SimProc Pmod adds a header for a USB-UART dongle and eight switches that set SimProc's bit period. It must have:

- every pin the project uses brought to the right socket, as its pin list says;
- a buffer with Ioff, powered from the kit's 3.3 V, on every line that crosses from another supply, then a resistor of 330 Ω to 8.2 kΩ on a line into the chip, or 1 kΩ to 5.1 kΩ on a line out of it;
- an ESD part to GND on every signal pin of a connector that a cable plugs into;
- a test point for every signal pin of that connector, anywhere on its line from the connector through the buffer and the resistor to the Pmod pin. One on GND doesn't count, and nor does one on the kit's 3.3 V, such as one at the far end of a pull-up;
- in `DESIGN.md`, what happens on each line when either side is unpowered, which firmware mode and switch settings the board assumes, and a check that the project's clock and its fastest signals stay within the pins' ratings, which Tiny Tapeout's GPIO spec gives as 33 MHz for an output and 66 MHz for an input;
- if the project's pinout is awkward for the demo board, a line in `DESIGN.md` on what it cost the board.

With the kit off, the buffer on a line that crosses from another supply is unpowered. Ioff is the most current a pin of the buffer passes while the buffer's supply is at 0 V. For the 74LVC2G17 on our board it's at most 10 µA, under the 100 µA that SkyWater's I/O guide allows into a chip pin held above its supply. On a line into the chip, the resistor limits the current if something else, such as the RP2350, drives the line against the buffer; on a line out of the chip, it limits the current into an unpowered receiver such as a dongle's RX pin. The buffer's inputs take up to 5.5 V, so it also brings a 5 V signal down to 3.3 V for the chip, whose inputs aren't 5 V tolerant. The capstone check's pmod row runs Lab 04's check. It accepts only the 74LVC1G17, 74LVC1G34, 74LVC1G125, 74LVC2G17, 74LVC2G34, 74LVC2G125, 74AUP1G17, 74AUP1G34 and 74AUP1G125 as the buffer, so a line with a translator in its place fails.

For a project other than SimProc, write its pin list in the form of `labs/common/pins/simproc.json`, as Lab 03's Going further describes, and pass it to the check with `--pins`.

Brief A: a source board for one analog pin

Brief A is for CrossWalk or an ASIC Zoo cell. The board plugs onto J13, the demo board's analog header, which is a 2 × 16 header with a 2.54 mm pitch. It sets a voltage you know with a trimmer or a DAC, adds a buffer if you want one, and drives one analog pin with that voltage through an RC filter at the pin. It must have:

- a 2 × 16 connector that mates J13, with GND on J13's ground pins (1, 6, 9, 14, 17, 22, 25, 30 and 31) and, if the board needs power, 3.3 V from pin 32;
- exactly one of J13's analog lines driven, and nothing on the lines that are ground on the SKY130 breakout;
- a series resistor into that line and a capacitor from it to GND;
- a test point on the source and one on the pin;
- in `DESIGN.md`, how the source settles into the pin's path and what error the demo board's 10 MΩ pull-down adds. The path goes through an analog switch, has under 500 Ω and under 5 pF, and carries 4 mA at most.

Twelve of J13's lines carry an analog pad of the chip on the SKY130 QFN breakout. We found which ones by following each line from J13 to the breakout's connector in Tiny Tapeout's board files (`tt-demo-pcb` at bbf3dd1, `breakout-pcb` at 91a2bf1):

J13 pin	4	5	7	10	12	13	29	28	26	23	21	20
Line	/C	/D	/F	/G	/J	/K	/X	/W	/U	/T	/R	/Q
Pad	an0	an1	an2	an3	an4	an5	an6	an7	an8	an9	an10	an11

Pins 2, 3, 8, 11, 15, 16, 18 and 19 are ground on that breakout. J13 is marked DNP in the demo board's files at bbf3dd1, so before you order anything, someone with a kit should check whether J13 is fitted on its demo board.

Part 1: Requirements first

What does your board have to do, in numbers?

1. Start in your repository. For Brief D, that's your Lab 03 to 06 repository. For Brief A, make a new one from the same template, the way Lab 03, Part 1 does, and copy into it the `.github/`, `ci/` and `.gitignore` that you finished in Lab 05.
2. Copy the design note's template in:

   ```
   cp ~/pcb101/labs/capstone-a-board-for-a-team-chip/DESIGN.md .
   ```

3. Fill in the title and the `Brief:` line at the top. Then write the Requirements as a numbered list of what the board has to do, taken from the brief and from your project, and give each requirement its number where it has one, such as a pin, a voltage, a current or a rate. Leave the Decisions for Part 3.
4. Commit `DESIGN.md`, because the check runs Lab 05's check first, and that fails on any file git doesn't track. Then write down which rows you expect to fail, and run the check with your brief:

   ```
   ~/pcb101/labs/capstone-a-board-for-a-team-chip/check.sh --brief D
   ```

   We ran it on a repository with one commit that held Lab 04's reference board, Lab 05's CI files and the unfilled template, and got these rows:

   ```
     ok    ci         Lab 05's check passes: the board check, every output, and .gitignore
    FAIL   design        7 lines still say TODO; only 1 numbered requirement, and at
   least 3 are needed; only 1 numbered decision, and at least 3 are needed; decision 1
   has no 'Meets:' line naming requirements; decision 1 gives no value with its unit;
   ```

```
requirement 1 isn't met by any decision
  FAIL  history      1 commit changes the schematic or the board; at least 2 needed
  ok    pmod        PASS  pmod.kicad_pro: 17 of 17 Pmod checks ok; bring-up plan PASS
  FAIL  test points  no test point on J4 pin 2 (/DONGLE_TXD) or J4 pin 3 (/DONGLE_RXD),
or along their lines to the Pmod pins
```

You should see the design row name what's still missing from your requirements and decisions. Your history row will differ from ours, because a repository carried through Labs 03 to 05 already has several commits that change the board. When we worked through the labs the way you have, it read `ok    history      5 commits change the schematic or the board`.

Part 2: The board

Does your board do what your requirements say?

1. Draw or change the board. For Brief D on the SimProc board, that's a test point on each dongle line; ours puts TP3 on `ui[0]` between J1 and R1, and TP4 on `uo[0]` between R2 and J4. For Brief A it's a new schematic and board.
2. For Brief A, place the connector before anything else, the way Lab 03 placed the Pmods. Your board sits on top of the demo board, so the socket goes on its underside. Select it and flip it to the bottom (F), then rotate it (R) until its pin 1 lies over J13′s pin 1 and its other pins follow J13′s. Before you look, write down which way you expect pin 2 to point from pin 1. KiCad's `PinSocket_2x16_P2.54mm_Vertical` can be placed eight ways, at four angles on either side, and only one of them, flipped and turned 270°, puts its pads over J13′s pad for pad. Our generator checked that against J13 in the demo board's file. Once the socket is in place, lock it.

   Answer: A1, in Answers at the end.
3. Route the board, refill the zones, and run DRC with the parity test ticked until it's clean. Commit as you go. The check wants at least two commits that change the schematic or the board, so that your work shows in the git history.
4. Run the check again. You should see the ci row pass, and the rows that check your brief's board pass as well. For Brief A, the bringup row still fails, because you write `BRINGUP.md` in Part 3. Our rows for Brief D were:

```
  ok    pmod        PASS  pmod.kicad_pro: 17 of 17 Pmod checks ok; bring-up plan PASS
  ok    test points  J4 pin 2: TP3, J4 pin 3: TP4
```

   and for Brief A:

```
  ok    connector    J1 mates J13: GND on its 9 GND pins, pin 32 on +3V3
  ok    pin          J1 pin 4 (J13 line /C, an0 on the SKY130 breakout), net /AN0
  ok    filter       R1 (1k) from /SET, C1 (10n) to GND
  ok    test points  TP1 on /SET, TP2 on /AN0
```

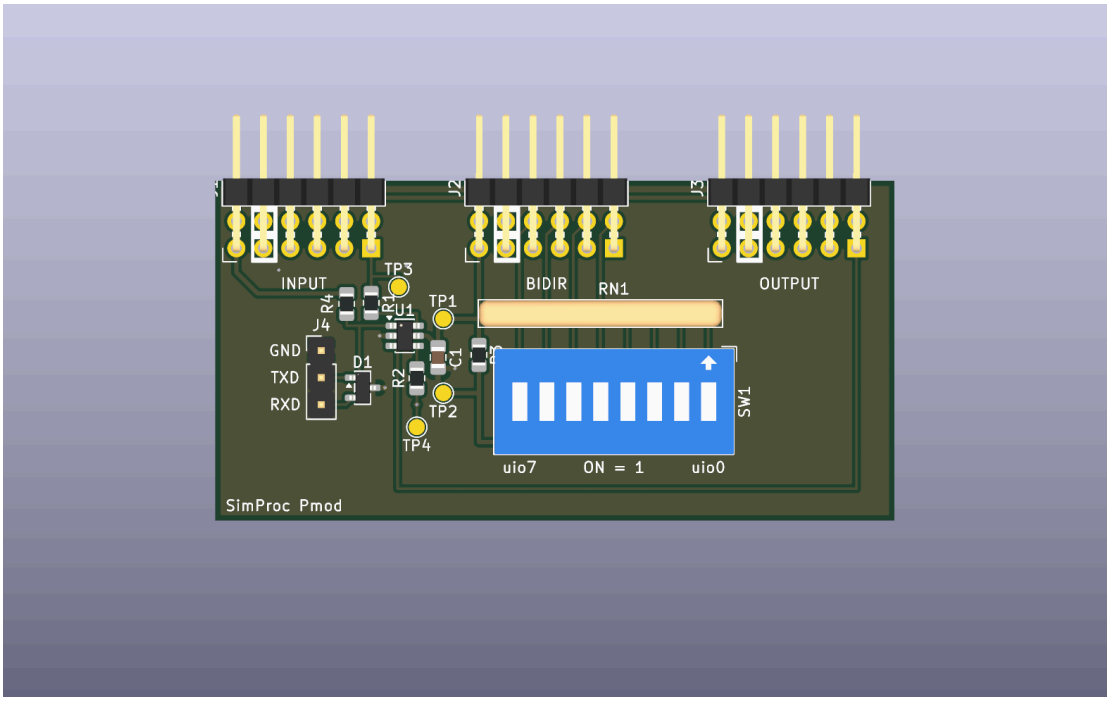


Figure 27.1: The Brief D reference board from the top

Our Brief D board, rendered by `kicad-cli` 10.0.5 with the parts' 3D models. It's Lab 04's SimProc Pmod with TP3 added to the right of R1, and TP4 below R2.

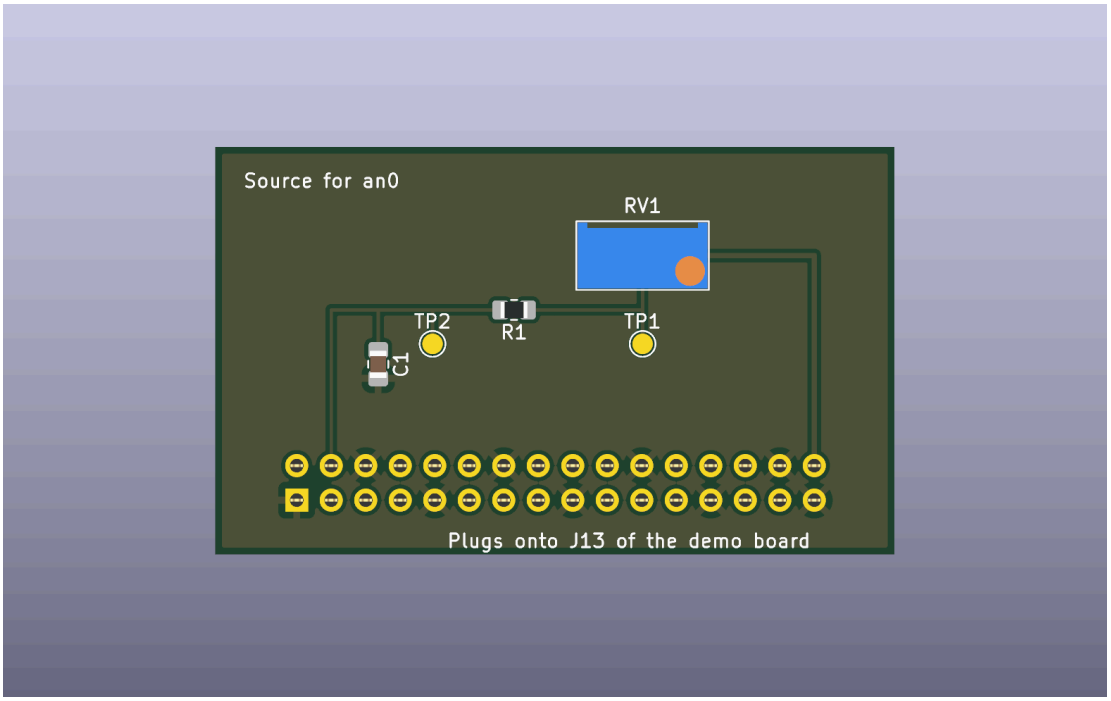


Figure 27.2: The Brief A reference board from the top

Our Brief A board, 50 × 30 mm, seen from the top. J1 runs along the bottom edge on the underside, so only its pads show. RV1 is a 1 kΩ trimmer, R1 and C1 are the filter at the pin, and the track down the left side goes to J1's pin 4.

Part 3: Decisions and the bring-up plan

Why is each part there, and what will you read at each step?

1. Write the Decisions in `DESIGN.md` as a numbered list, with one item for each part or choice. Each item gives the value that justifies it, with its unit, and has a line starting `Meets:` that names the requirements it meets. Every requirement needs at least one decision. Work every number out in the note itself, as our references do. For our R1, the 1 kΩ resistor after the buffer on `ui[0]`, that might be "3.3 V × 36 kΩ / 37 kΩ = 3.21 V against the RP2350′s 36 kΩ pull-down, above the 2.31 V the chip needs for a high", and for the pull-down on Brief A, with a 1.25 kΩ source, "1.25 kΩ / 10 MΩ = 0.0125 %".
2. For Brief A, take care over the current that the pin's sampling capacitor draws on average. It's roughly $C \cdot V \cdot f_s$, and the voltage it drops across your source's resistance is an error at the pin. Before you work it out, write down whether you think it matters for an 8-bit converter. Our reference assumes a sample rate of 400 kS/s, because CrossWalk's documentation doesn't give one yet. At that rate the current is 5 pF × 3.3 V × 400 kHz = 6.6 µA, and across 1.25 kΩ it drops 8.3 mV, which is 0.64 of one 12.9 mV step. That's why our trimmer is 1 kΩ and not 10 kΩ.

   Answer: A2, in Answers at the end.
3. Fill in Checks (what you checked and how) and Open questions (what you couldn't settle, and who could). Our Brief A reference lists four open questions, one of them the DNP mark on J13.
4. Write `BRINGUP.md`. For Brief D, start from your Lab 04 plan and change the steps that measure the UART lines so that they measure at your new test points; in ours, step 6 measures at TP4 and step 7 at TP3. For Brief A, copy Lab 04′s template and fill it in for your board, where the lines that leave the board are the analog pin, 3.3 V and GND. As in Lab 04, every step needs an `Expect:` with a value and its unit.
5. Commit, and run the check. You should see `ok  design  ...` with your own counts, for example `9 requirements, 11 decisions, every requirement met`, and for Brief A `ok  bringup  PASS  BRINGUP.md: 6 of 6 parts ok`.

Part 4: Review it and push

Would your board pass the review you gave the seeded one?

1. Review your own board against the checklist from Lab 06, line by line, and write what you checked under Checks in `DESIGN.md`. Fix what you find. For a line that doesn't apply, say why, as "there is no IC, so nothing needs decoupling" does in our Brief A note.
2. Commit, push, and wait for your workflow's run. You should see it green, with the `board-outputs` artifact.
3. Run the check one last time on the commit you pushed. You should see every row `ok` and the last line `PASS  pmod, Brief D: 5 of 5 capstone checks ok`, or for Brief A `PASS  source, Brief A: 8 of 8 capstone checks ok`, with your own KiCad project's name in place of `pmod` or `source`.

The check can't tell whether your requirements are the right ones, whether a decision's reasoning holds, or whether the values in `BRINGUP.md` are what you'll read. It takes a person reading `DESIGN.md` and `BRINGUP.md` to judge those, and nobody should power a kit with your board on it until someone other than you has read the plan.

Finishing

You're done when `~/pcb101/labs/capstone-a-board-for-a-team-chip/check.sh --brief D` (or `--brief A`), run in your own repository on the commit you pushed, ends with `PASS`, and that commit's run on GitHub is green.

Questions:

1. Brief D asks for test points on the dongle lines, and not on the project's clock. Why can't a Pmod give you a test point on the clock?

   Answer: A3, in Answers at the end.

2. For Brief A, why put a 10 nF capacitor right at the pin? With the source's largest resistance, 1.25 kΩ, its RC corner is at 12.7 kHz, far above anything a trimmer turned by hand can do.

   Answer: A4, in Answers at the end.

Submit: the URL of your repository and the hash of the commit the check passed on.

Going further (optional)

- Order your board with your cohort, after a second person has read `BRINGUP.md`, and bring it up once there's a chip to bring it up with. Reviewing and ordering a board has the costs. SimProc is on TTSKY25b, so it could be on a bench before the TTSKY26d chips, which are expected in May 2027.
- For Brief A, add a buffer between the trimmer and the filter, and work out how much of the sampling droop it removes. If you use KiCad's `MCP6001R`, a 1 MHz op-amp in SOT-23-5, check its supply pins. Its pin 2 is V+ and its pin 5 is V−, the reverse of `MCP6001-OT`, which is the plain MCP6001 in the same package. A swap like that, between two variants of one part in the same package, is the kind of fault you looked for in Lab 06.
- Do both briefs.

Next on the Course path: this is the last row. A second course, PCB201, would take up what PCB101 left out: 4-layer boards and decoupling for a QFN, characterization boards with a reference of their own, and high-speed signals.

Answers

A1. Part 2: The board

Seen from above, J13′s pin 2 is 2.54 mm in −Y from pin 1 (up, on the screen) and pin 3 is 2.54 mm in +X (to the right), with the odd pins in one row and the even pins in the other. Flipping a footprint to the underside mirrors it, so a placement that looks right can be the mirror image of the one that mates. The PCB Editor draws each pad's number on the pad, so you can check that your pin 2 is above pin 1 and pin 3 is to its right, as on J13, and that pin 32 ends up at the far end of pin 2′s row, diagonally across from pin 1.

A2. Part 3: Decisions and the bring-up plan

It does. With a 10 kΩ trimmer, the source's resistance can reach a quarter of 10 kΩ, with the wiper in the middle, plus R1′s 1 kΩ: 2.5 kΩ + 1 kΩ = 3.5 kΩ. The same 6.6 µA then drops 23 mV, nearly two steps. The charge that a single sample takes from a 10 nF capacitor is small (5 pF / 10 nF = 0.05 %), but at a high sample rate the capacitor never fully recovers between samples, so the average current decides the error. A buffer amplifier between the trimmer and the filter would make the source's resistance small, and adding one is the second item under Going further.

A3. Finishing

The clock doesn't come out on the Pmod sockets. On the demo board (`tt-demo-pcb` at bbf3dd1), each I/O socket carries eight project pins, GND on pins 5 and 11, and +3V3 on pins 6 and 12, and nothing else. The clock, which the RP2350 usually provides, reaches the chip without passing through any of them. To probe it, use the test point that the breakout has on its clock line.

A4. Finishing

It's there mostly as a reservoir of charge. When the pin's analog switch closes, the capacitance behind it, up to 5 pF, charges from whatever is closest, and with 10 nF at the pin that charge moves the voltage by only 0.05 %. The capacitor filters as well, so it also keeps noise picked up on the board, and noise from the trimmer's wiper, out of the pin.