

---

# horizon-eda Documentation

*Release 1.0*

**Lukas K.**

**Sep 10, 2022**



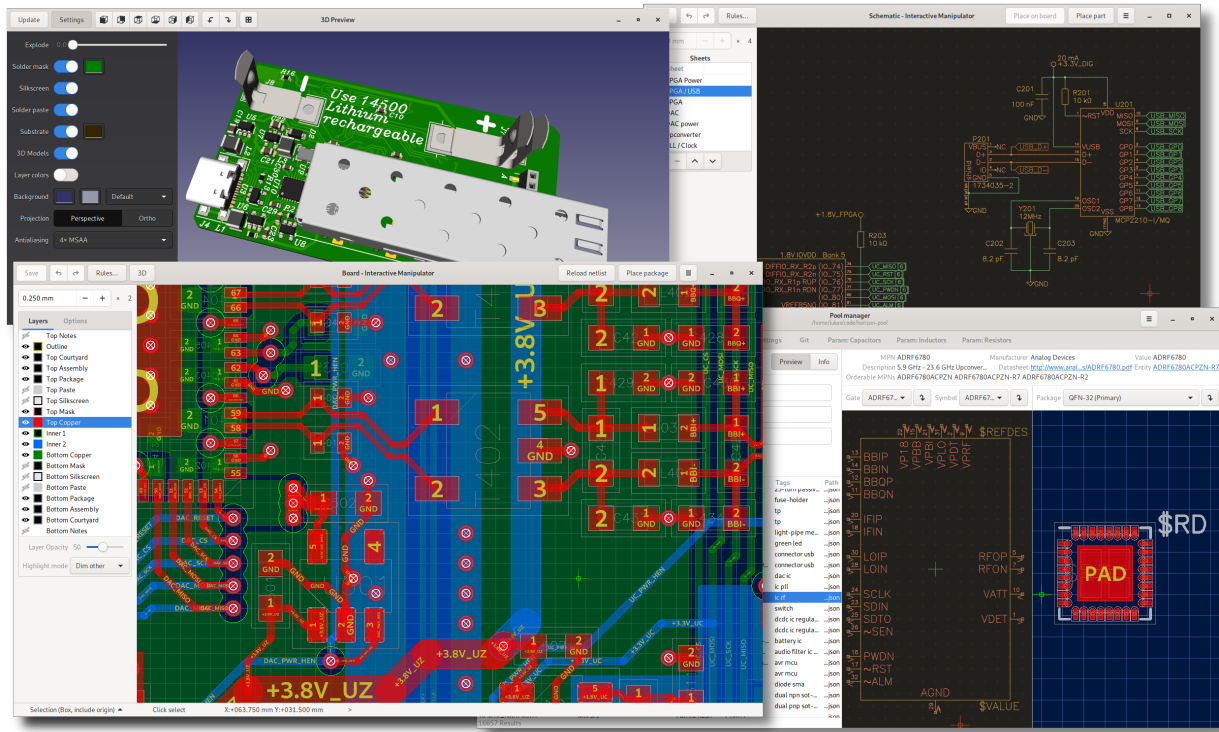
## ABOUT:

<b>1</b>	<b>Why another EDA package</b>	<b>3</b>
<b>2</b>	<b>Feature overview</b>	<b>5</b>
<b>3</b>	<b>Non-Goals</b>	<b>25</b>
<b>4</b>	<b>Made with Horizon EDA</b>	<b>27</b>
<b>5</b>	<b>Installation</b>	<b>31</b>
<b>6</b>	<b>Setting up a pool</b>	<b>33</b>
<b>7</b>	<b>Create a new Project</b>	<b>35</b>
<b>8</b>	<b>Draw a Schematic</b>	<b>39</b>
<b>9</b>	<b>Create a Board</b>	<b>43</b>
<b>10</b>	<b>Example project</b>	<b>47</b>
<b>11</b>	<b>Digi-Key API integration</b>	<b>49</b>
<b>12</b>	<b>Tools</b>	<b>53</b>
<b>13</b>	<b>Spacebar Menu</b>	<b>55</b>
<b>14</b>	<b>Grid</b>	<b>57</b>
<b>15</b>	<b>Numeric Entries</b>	<b>59</b>
<b>16</b>	<b>Drawing</b>	<b>61</b>
<b>17</b>	<b>Selection</b>	<b>65</b>
<b>18</b>	<b>Moving and the interactive manipulator</b>	<b>67</b>
<b>19</b>	<b>Layers</b>	<b>69</b>
<b>20</b>	<b>Tips and Tricks</b>	<b>71</b>
<b>21</b>	<b>Schematic Editor</b>	<b>73</b>
<b>22</b>	<b>Board Editor</b>	<b>79</b>

<b>23 Copying layout and placement</b>	<b>85</b>
<b>24 Backannotating connections</b>	<b>87</b>
<b>25 Rules</b>	<b>89</b>
<b>26 Why a Pool?</b>	<b>91</b>
<b>27 Elements of a Part</b>	<b>95</b>
<b>28 Project pool</b>	<b>97</b>
<b>29 Pool Manager</b>	<b>99</b>
<b>30 Creating a Package</b>	<b>103</b>
<b>31 Contribute to the Pool</b>	<b>105</b>
<b>32 Building on Windows</b>	<b>107</b>
<b>33 Building on Linux</b>	<b>109</b>
<b>34 Building on FreeBSD</b>	<b>111</b>
<b>35 Parameter Programs</b>	<b>113</b>
<b>36 Theory of Operation</b>	<b>117</b>
<b>37 CLI usage</b>	<b>119</b>
<b>38 Python module</b>	<b>121</b>
<b>39 File versions</b>	<b>125</b>
<b>Index</b>	<b>127</b>

Horizon EDA is an **Electronic Design Automation** package supporting an integrated end-to-end workflow for printed circuit board design including parts management and schematic entry.

Take a look at the *top features*, just *try it out* or begin by *reading from the start*.





## WHY ANOTHER EDA PACKAGE

So you may be wondering why I started horizon EDA back in 2016, given that KiCad was a thing at that time.

Let's get started with a quote from Tom Hausherr: [PCB Design Perfection Starts in the CAD Library](#)

[...] PCB design perfection starts in the CAD library.

This also applies to EDA software as in a schematic / PCB design tool can only be as good its library structure. Since the definition of library items such as symbols packages is the very foundation of any EDA software, changing these definitions is next-to-impossible without significant changes to almost all other parts of the application. Having a certain library structure in place also thus guides any further development on that EDA application.

Having used KiCad for small and medium-sized projects, my biggest pain points were the library lacking a concept of orderable parts without duplicating the symbol for each part and the schematic editor not knowing about nets. KiCad's board editor, while being quite good as a layout tool was lacking expressive design rules. Especially the first and second point didn't seem easy to alleviate without major changes that would have involved lots of discussion since these would be breaking changes.

That made me start thinking how I'd design an EDA tool that meets my wishes and is easy enough to implement from scratch as a one man show and provides a clean-slate playground for experimentation.

At the very core of these thoughts was to keep schematic and netlist representation of a design separate to allow for non-schematic based workflows such as interconnectivity tables. That lead to the decision to define pins and their direction in what's called a Unit and not in the symbol as it's common among many other EDA packages. This also makes it possible to have multiple symbols representing the same thing (such as a resistor) without any effect on the netlist. Apart from name and direction, a pin as it's defined in a Unit can also have multiple alternate names to specify multiple pin functions as they're commonly available on MCUs and FPGAs.

To define the netlist representation of an actual part, units are referenced by what's called an Entity. This reference is called Gate. For simple parts, an entity references just a single Unit that includes all pins. For some parts it makes sense to have more gates.

Parts that include multiple instances of the same functionality such as quad opamps will then reference the opamp unit 4 times as well as a unit for power supply.

On the board side of things, a packages are defined as in pretty much every EDA package out there - pads and graphical items such as silkscreen, reference designator and assembly outline. Pads however are defined by a padstack describing copper, soldermask and other layers in terms of shapes and polygons. This greatly facilitates odd-shaped pads as they can be drawn as-is without resorting to hacks such as using multiple pads to make up one actual pad. To avoid having a custom-drawn padstack for every pad size, padstacks are usually accompanied by a short script written in a custom stack-based language to adjust their size as well as other properties such as corner radius or solder mask expansion.

The pads of a package are mapped to the pins defined in the units referenced by an entity in what's called a Part. In order to mapped to something orderable, Parts have fields for the manufacturer name and the manufacturer's part number (MPN) among other details such as datasheet link or description.

All aforementioned references between items (such as entities referencing units) are by UUIDs. In Horizon EDA, all items get assigned an immutable UUID at time of their creation. This UUID is then used by other items to reference this item.

To wrap up this introduction:

My biggest weakness is that i will eventually turn any arbitrary electronics project into an excuse to write EDA software (and vice versa).

(Based on <https://twitter.com/mycoliza/status/824809235632447492>)

Next: *Feature Overview*



## FEATURE OVERVIEW

### 2.1 Sane and simple part management

Easily manage parts, packages and symbols with the *pool manager* (also see *What is a Pool?*):

The screenshot shows the 'Pool manager' application interface. The top bar includes a search bar and navigation tabs: Units, Symbols, Entities, Padstacks, Packages, **Parts**, Frames, Settings, Git, Param: Capacitors, Param: Inductors, and Param: Resistors. Below the tabs are buttons for 'Create', 'Edit', 'Duplicate', 'Create Part from Part', 'Preview', and 'Info'.

The main area is divided into two panels. The left panel is a table of parts, and the right panel shows the details for the selected part, 'ADRF6780'.

MPN	Manufacturer	Description	Package	Tags	Path
3517	Keystone	5mm fuse clip	3517	fuse-holder	..json
5000	Keystone	Miniature TH...	5000	tp	..json
5001	Keystone	Miniature TH...	5000	tp	..json
7511B11	VCC	Light Pipe, 3 ...	7511B11	light-pipe me...	..json
150060G575000	Würth Elektronik	LED green cle...	0603 LED	green led	..json
1734035-2	TE Connectivity	Mini USB-B r...	1734035-2	connector usb	..json
10118194-0001LF	Amphenol FCI	10118194-00...	connector usb	..json	..json
AD9122BCPZ	Analog Devices	Dual 16-Bit, 1...	CP-72-7	dac ic	..json
ADF4356BCPZ	Analog Devices	6.8 GHz frac...	QFN-32	ic pll	..json
<b>ADRF6780</b>	<b>Analog Devices</b>	<b>5.9 GHz - 23...</b>	<b>QFN-32</b>	<b>ic rf</b>	<b>..json</b>
AP2280-1WG-7	Diodes Inc.	Single Chann...	SOT-23-5	switch	..json
AP3015	Diodes Inc.	micro power ...	SOT-23-5	dc/dc ic regula...	..json
AP3015AKTR-G1	Diodes Inc.	micro power ...	SOT-23-5	dc/dc ic regula...	..json
AP9211	Diodes Inc.	Single chip Li...	U-DFN2030...	battery ic	..json
AS3320	Alfa	Voltage Contr...	DIP18, 7.62 ...	audio filter ic ...	..json
ATtiny5 DIP	Microchip	AVR MCU wit...	DIP8, 7.62 m...	avr mcu	..json
ATtiny5 SOIC	Microchip	AVR MCU wit...	SOIC-8	avr mcu	..json
B340A-13-F	Diodes Inc.	3.0A surface ...	SMA	diode sma	..json
BC846AS	Diodes Incorporated	Dual NPN sm...	SOT-363	dual npn sot...	..json
BC856AS	Diodes Incorporated	Dual PNP Sm...	SOT-363	dual pnp sot...	..json
RI M184CG01SM1D	Murata	Ferrite bead...	C0603	...	..json

The right panel shows the details for 'ADRF6780':

- MPN: ADRF6780
- Manufacturer: Analog Devices
- Description: 5.9 GHz - 23.6 GHz Upconver...
- Datasheet: <http://www.anal.../ADRF6780.pdf>
- Entity: ADRF6780ACPZN-R7
- Orderable MPNs: ADRF6780ACPZN ADRF6780ACPZN-R7 ADRF6780ACPZN-R2
- Gate: ADRF67...
- Symbol: ADRF67...
- Package: QFN-32 (Primary)

The bottom right shows a schematic diagram of the ADRF6780 component, highlighting the 'PAD' area.

Assign pins to pads in the part editor:

Save

Part Editor

×

MPN

ADRF6780

Inherit

orderable MPNs

ADRF6780ACPZN, ADRF6780ACPZN-R7, ADRF...

Value

Inherit

Manufacturer

Analog Devices

Inherit

Datasheet

http://www.analog.com/media/en/techn

Inherit

Description

5.9 GHz - 23.6 GHz Upconverter

Inherit

Tags

ic × rf × +

Inherited:

Inherit

Base part

none

Entity

[ADRF6780ACPZN-R7 / Analog Devices](#)

Package

QFN-32

Change

3D Model

QFN-32-1EP\_5x5mm\_P0.5mm.step ▼

Inherit

Parametric data

Table

None ▼

Gate ▼	Pin	Mapped
Main	AGND	✓
Main	BBIN	✓
Main	BBIP	✓
Main	BBQN	✓
Main	BBQP	✓
Main	IFIN	✓
Main	IFIP	✓
Main	LOIN	✓
Main	LOIP	✓
Main	PWDN	✓
Main	RFON	✓
Main	RFOP	✓
Main	SCLK	✓
Main	SDIN	✓
Main	SDTO	✓
Main	VATT	✓
Main	VDET	✓
Main	VP18	✓
Main	VPBB	✓
Main	VPBI	✓
Main	VPDT	✓
Main	VPLO	✓
Main	VPRF	✓
Main	~ALM	✓
Main	~RST	✓
Main	~SEN	✓

0 pins not mapped

Pad ▼	Gate	Pin
1	Main	VDET
2	Main	VPDT
3	Main	VPRF
4	Main	AGND
5	Main	RFOP
6	Main	AGND
7	Main	RFON
8	Main	AGND
9	Main	VPRF
10	Main	VATT
11	Main	BBQN
12	Main	BBQP
13	Main	BBIP
14	Main	BBIN
15	Main	VPBB
16	Main	PWDN
17	Main	~RST
18	Main	IFIN
19	Main	AGND
20	Main	IFIP
21	Main	VPBI
22	Main	VP18
23	Main	SDIN
24	Main	SCLK
25	Main	SDTO
26	Main	~SEN
27	Main	VPLO
28	Main	LOIN
29	Main	AGND
30	Main	LOIP
31	Main	VPLO
32	Main	~ALM
PAD	Main	AGND

0 pads not mapped

Map

Unmap

Auto map pads

Select pads

Select pin

Copy from other part

## 2.2 Easy part creation

Simply add pins as they're listed in the datasheet:

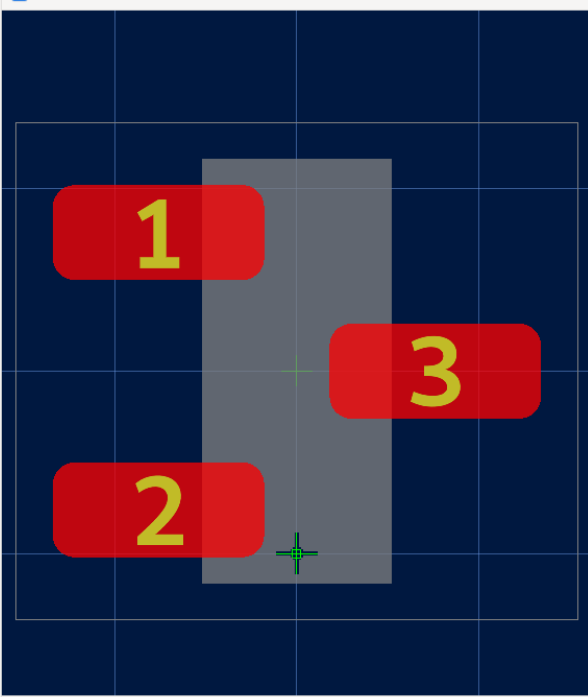
Next
Part Wizard
x

1	RESET	Input	Alt. Names	Gate	Main
2	D	Input	Alt. Names	Gate	Main
3	CLK	Input	Alt. Names	Gate	Main
4	~CLK	Input	Alt. Names	Gate	Main
5	Vee	Power Input	Alt. Names	Gate	Main
6	~Q	Output	Alt. Names	Gate	Main
7	Q	Output	Alt. Names	Gate	Main
8	Vcc	Power Input	Alt. Names	Gate	Main

Horizon also ships with script templates for importing industry-standard formats like IBIS saving you the tedious work of typing what's in the datasheet.

## 2.3 Easy package creation

Get a head start on creating packages by choosing from over 20 IPC-compliant footprint presets:

Generate		IPC-7351B	Dual	Single	Quad	Grid	x
CHIPARRAY	<b>Small outline transistor (JEDEC TO-236AB)</b>						
LEDSC	<b>Body</b>						
MELF	Length D	2.900 mm	-	+			
MOLDED	Width E1	1.300 mm	-	+			
CAPAE	Style	3 pins		5 pins			
SO	<b>Lead</b>						
SOD	Pitch e	0.950 mm	-	+			
SODFL	Length L	0.365 mm	-	+	±	0.235 mm	- +
SOJ	Width b	0.420 mm	-	+	±	0.120 mm	- +
QFP	Span E	2.370 mm	-	+	±	0.270 mm	- +
SON	<b>Calc</b>						
QFN	Density	Most		Nominal		Least	
PSON	Fabrication	0.100 mm	-	+			
PQFN	Placement	0.100 mm	-	+			
BGA	<b>Generate</b>						
SOT223	Round-off	None	0.01 mm	0.02 mm	0.05 mm		
<b>SOT23</b>	Padstack	Rectangular		Rounded rectangular			
DIP							
SIP							
PGA	<b>Lead span</b> Distance measured from lead termination to lead termination.						
							<input checked="" type="checkbox"/> Auto fit  as per IPC-7351B, Table 3-2 Flat Ribbon L and Gull-Wing Leads (greater than 0.625 mm pitch)

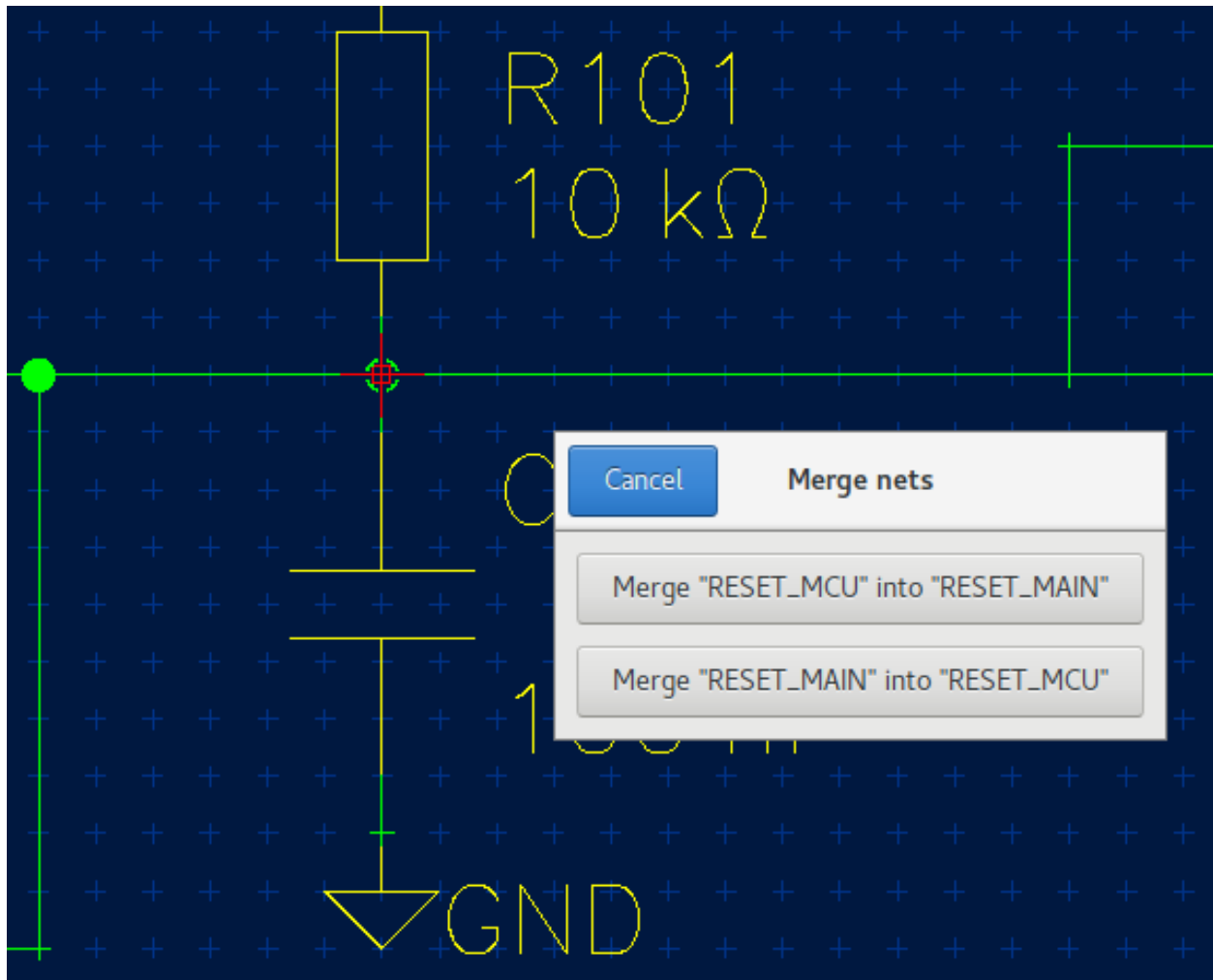
Importing KiCad footprints is supported as well.

## 2.4 Loves beginners and power users alike

Just press the spacebar and get a list of all the actions you can perform. These actions can be bound to customizable single key shortcuts or to vim-like multi key sequences.

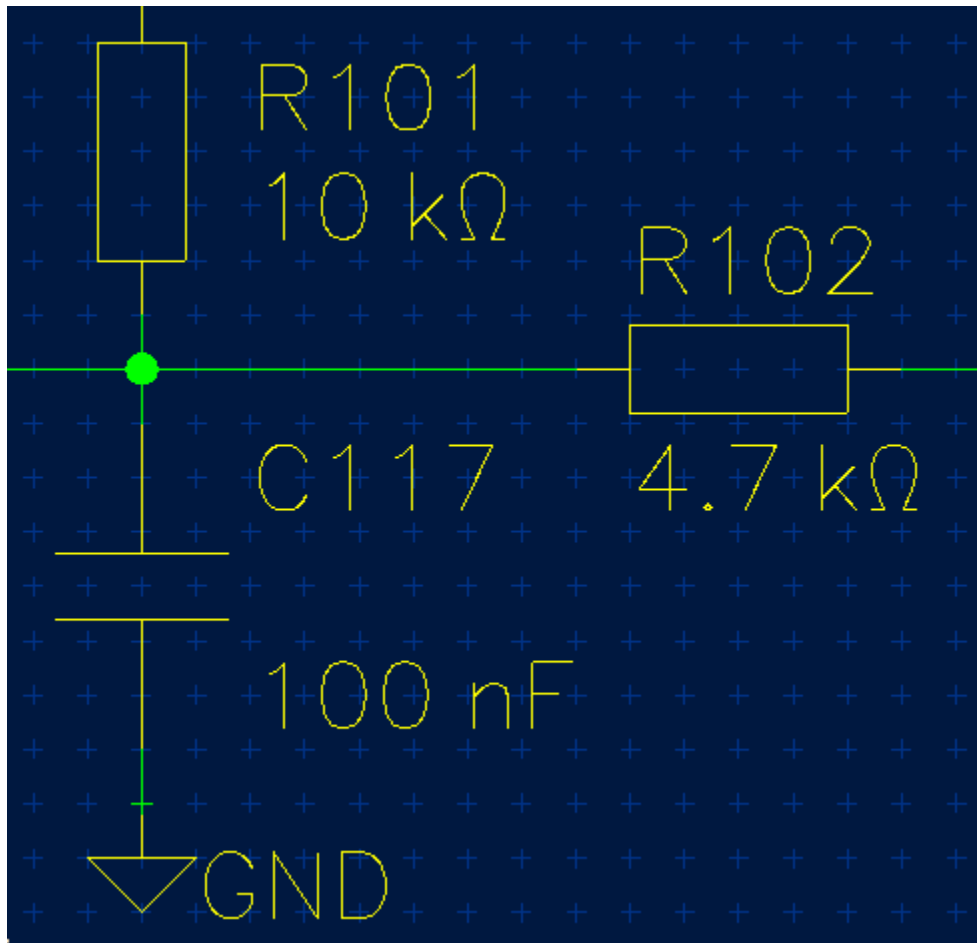
## 2.5 A schematic editor that knows what you're doing

Schematics aren't just about lines and labels. Horizon's schematic editor knows about nets and asks you when merging them:

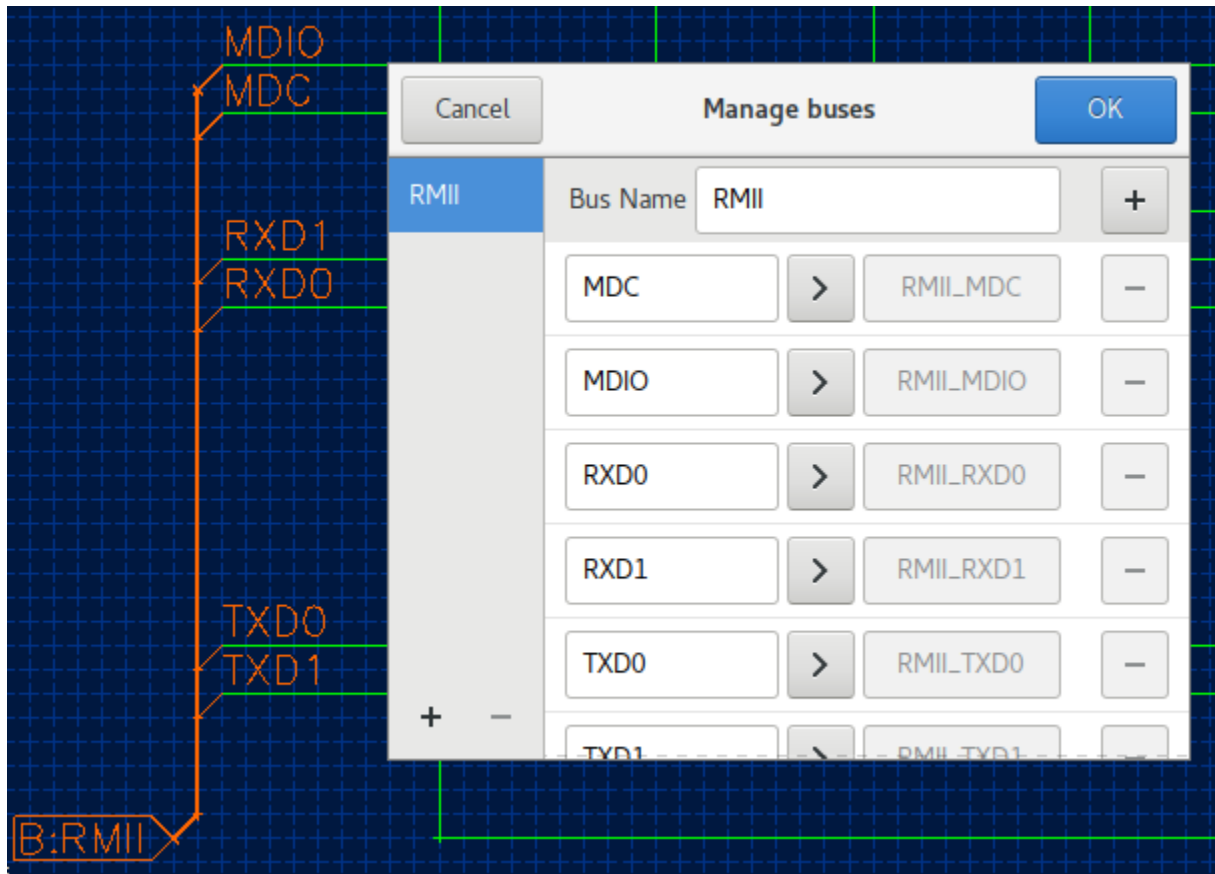


It places junctions where they should be:

It also reorients texts automatically, so you don't end up with hard-to-read reference designators:

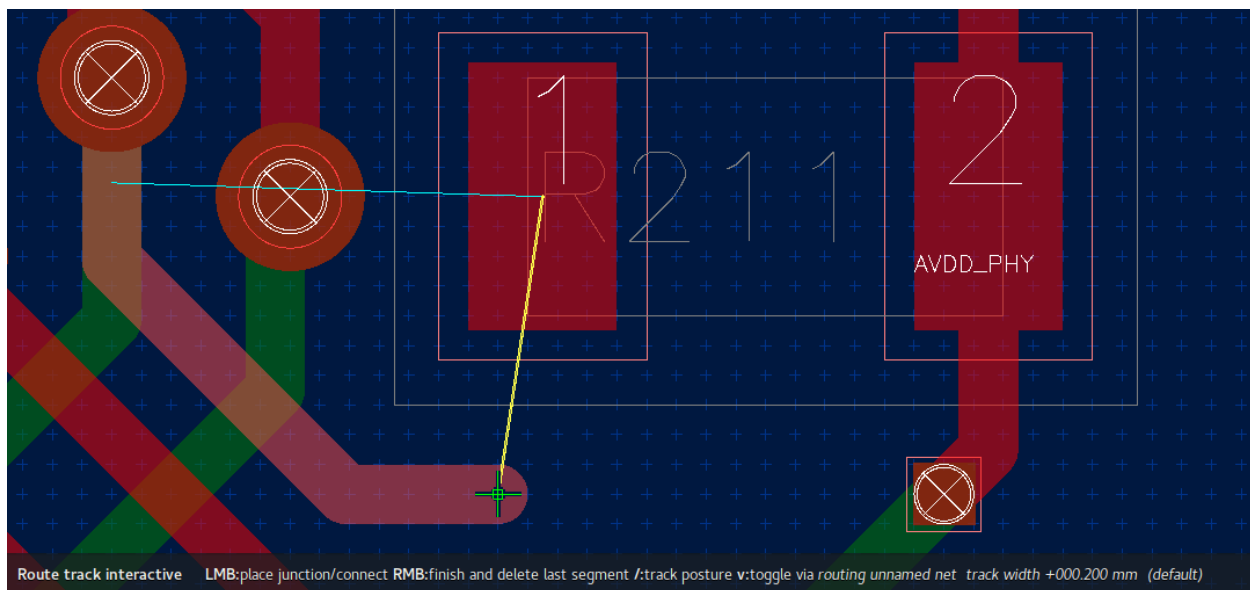


Buses aren't foreign to horizon either:



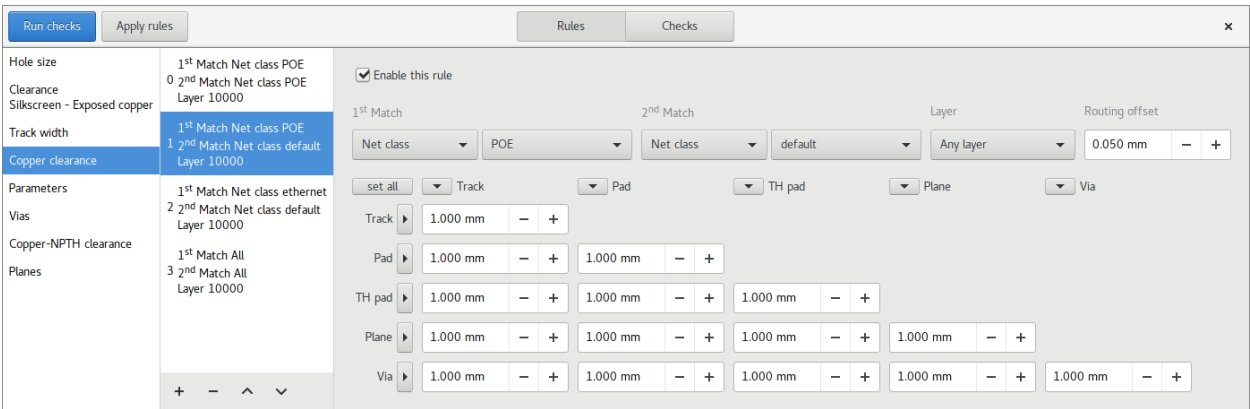
## 2.6 Interactive router with online DRC

By using the interactive router originally developed for KiCad, routing tracks becomes a breeze. Of course, it respects your design rules. Routing differential pairs is supported as well.

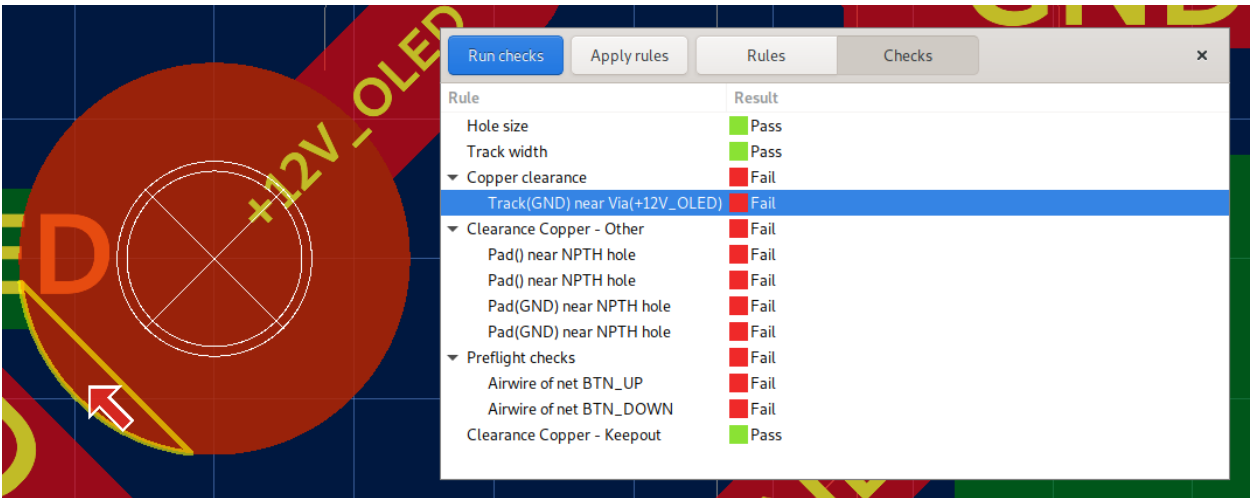


## 2.7 Powerful rules

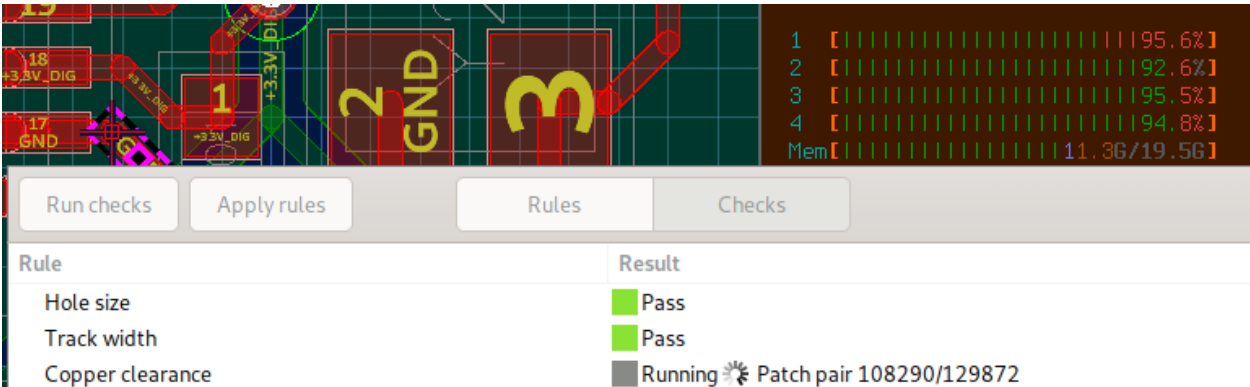
With powerful and flexible rules, horizon can check and adjust your design to meet its constraints:



If something doesn't meet your rules, horizon exactly tells what's wrong in which place:



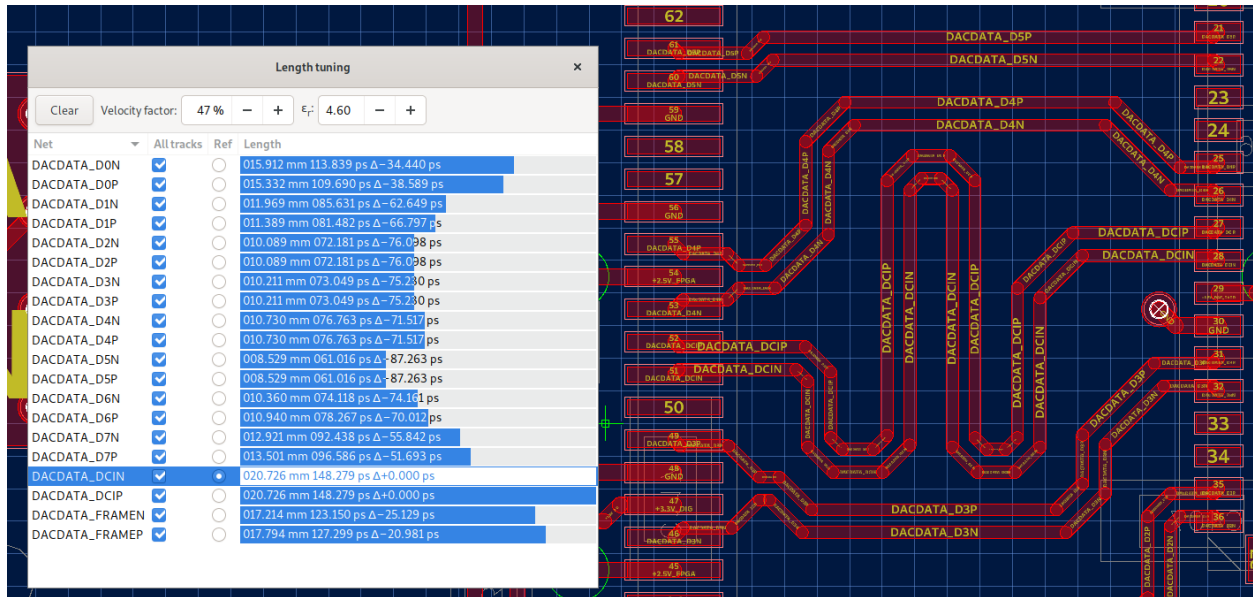
Multithreaded DRC makes use of all CPU cores:





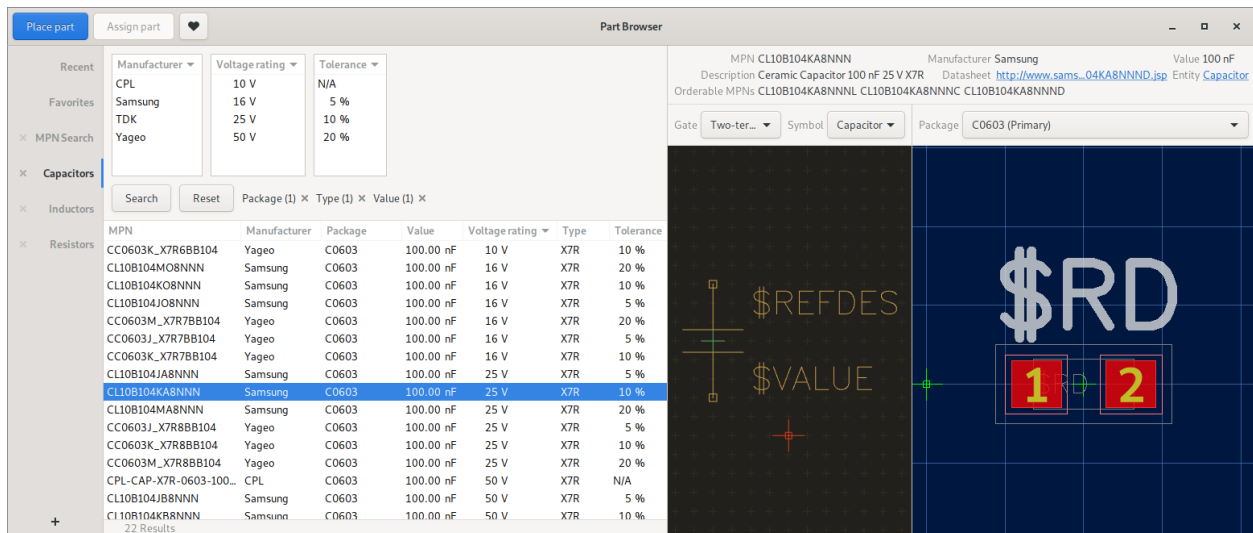
## 2.8 Interactive length tuning

Interactively measure and tune individual tracks, differential pairs or buses:



## 2.9 Parametric Search

Parametric part search helps you to quickly find passives:



## 2.10 Stock information

Real-time stock information powered by [Kitspace's partinfo](#):

**Part Browser**

MPN LP5907MFX-1.8/NOPB      Manufactu  
Description 250 mA Ultra-Low-Noise, Low ...      Datashe

Gate **Three-t...** ▼      Symbol **Three-t...** ▼      Pac

**Digikey Stock**

1,072
481
129
... 391
825
172
r Not found
r <b>241,460</b>
r 348,244
N/A
N/A
... 3,293
2,577
N/A
1,198
4,540
4,126
... Not found
... 0

**250-mA ultra-low-noise low-IQ low-dropout (...)**

**Digikey [296-41463-1-ND](#)**      Stock: 241,460

1	\$0.550
10	\$0.475
25	\$0.444

**Digikey [296-41463-6-ND](#)**      Stock: 241,460

1	\$0.550
10	\$0.475
25	\$0.444

**Farnell [3007714](#)**      Stock: 8,422

1	£0.402
50	£0.351
100	£0.211

**Mouser [595LP5907MFX1.8NPB](#)**      Stock: 28,289

1	\$0.540
10	\$0.450
100	\$0.271

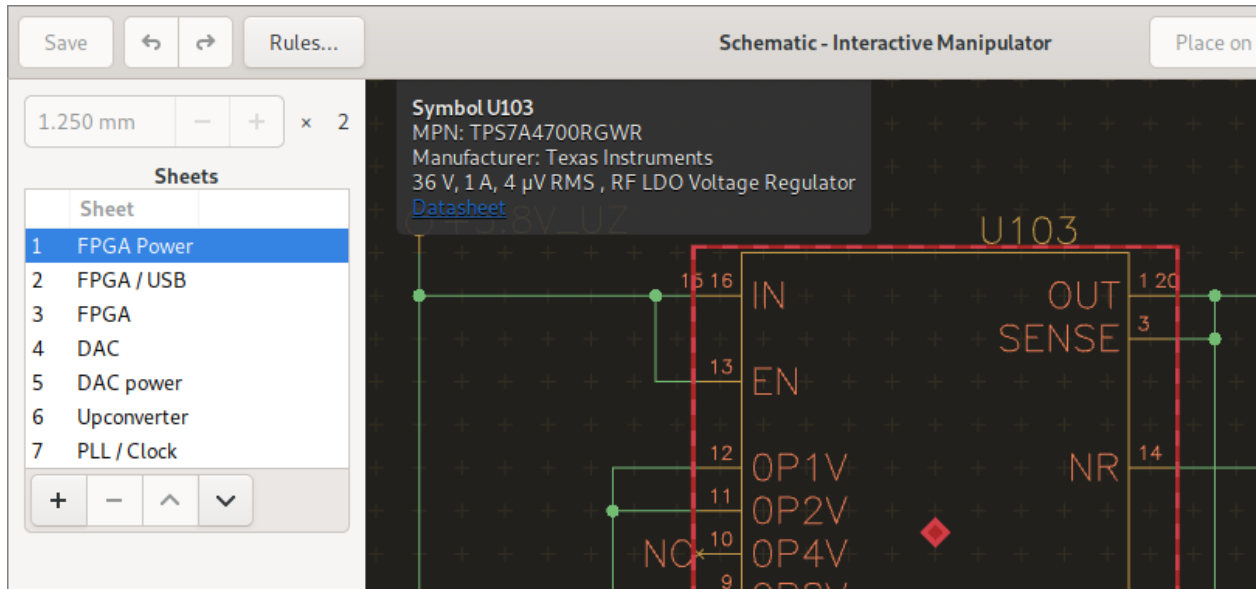
**Newark [28AH3785](#)**      Stock: 8,382

1	\$0.540
10	\$0.450
25	\$0.390

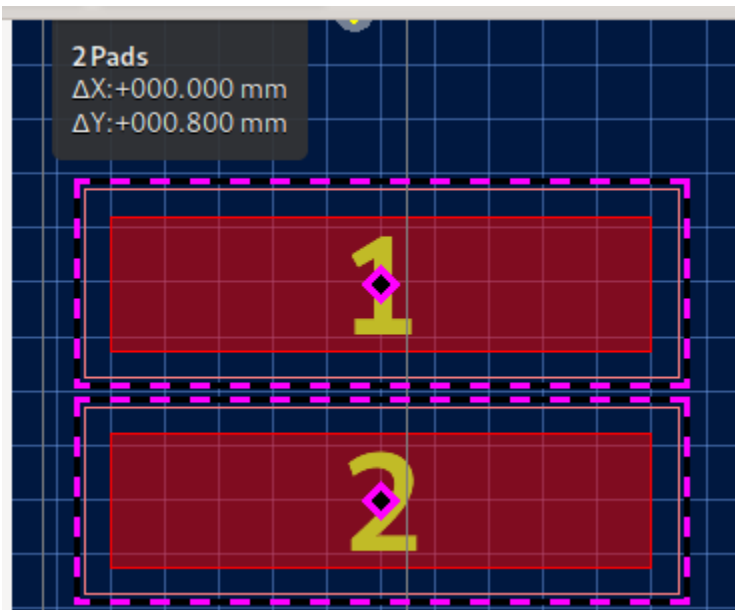
Information provided by [partinfo](#)

## 2.11 Smart Head-up Display

Instantly know what you're looking at with direct link to datasheets:



Measures pad distance and much more:



## 2.12 Industry-standard fabrication outputs

When your design is ready for fabrication, simply export industry-standard RS-274X gerber and NC-Drill files:

Generate

Fabrication outputs

×

NC-Drill

Mode

Merge PTH & NPTH

▼

Drill suffix

.txt

Gerber Settings

Outline width

0.010 mm

—

+

Output Settings

Base filename

hubble

Directory

output/gerber

Browse...

Generate Zip

☒

Resulting filenames are the concatenation of base filename and suffix.

Gerber Layers

<input checked="" type="checkbox"/> Outline	Suffix	.gko
<input checked="" type="checkbox"/> Top Paste	Suffix	.gtp
<input checked="" type="checkbox"/> Top Silkscreen	Suffix	.gto
<input checked="" type="checkbox"/> Top Mask	Suffix	.gts
<input checked="" type="checkbox"/> Top Copper	Suffix	.gtl
<input checked="" type="checkbox"/> Bottom Copper	Suffix	.gbl
<input checked="" type="checkbox"/> Bottom Mask	Suffix	.gbs
<input checked="" type="checkbox"/> Bottom Silkscreen	Suffix	.gbo
<input checked="" type="checkbox"/> Bottom Paste	Suffix	.gbp

Wrote layer Bottom Paste to gerber file /home/lukas/code/hubble/output/gerber/hubble.gbp

Wrote layer Bottom Silkscreen to gerber file /home/lukas/code/hubble/output/gerber/hubble.gbo

Wrote layer Bottom Mask to gerber file /home/lukas/code/hubble/output/gerber/hubble.gbs

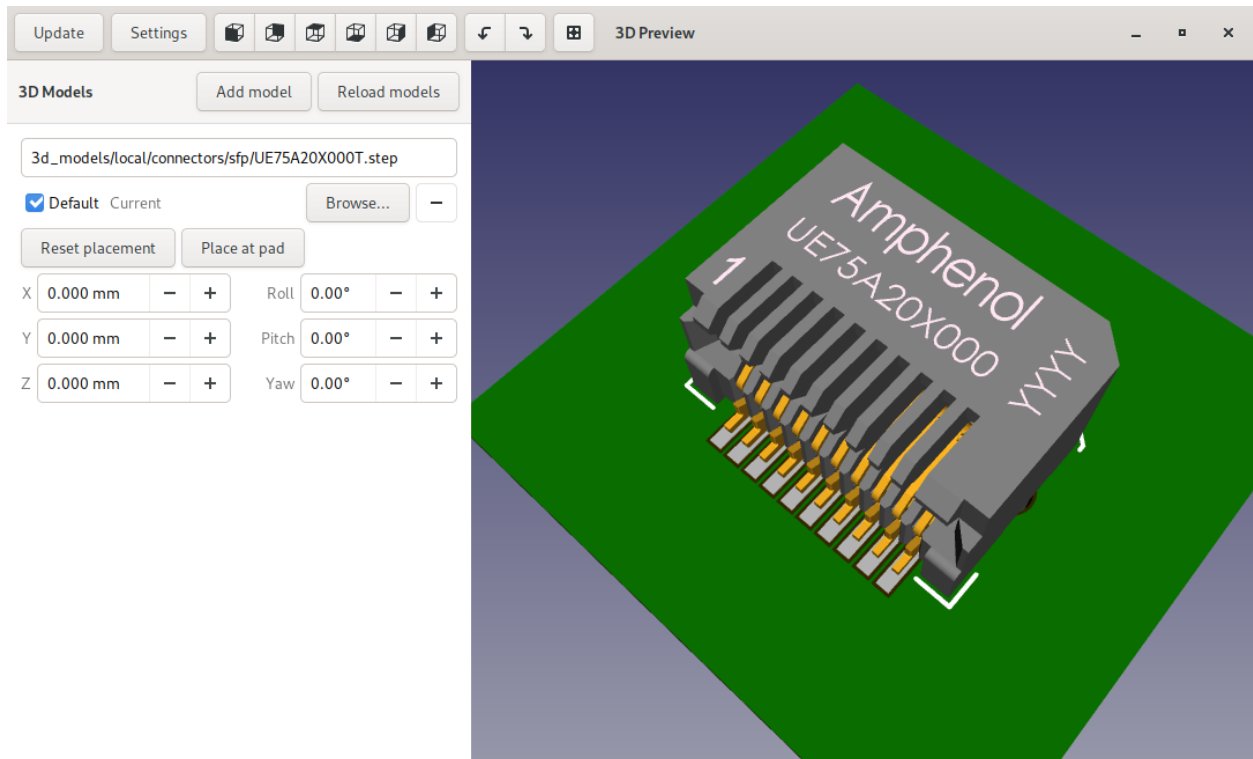
Wrote layer Bottom Copper to gerber file /home/lukas/code/hubble/output/gerber/hubble.gbl

Wrote layer Top Copper to gerber file /home/lukas/code/hubble/output/gerber/hubble.gtl

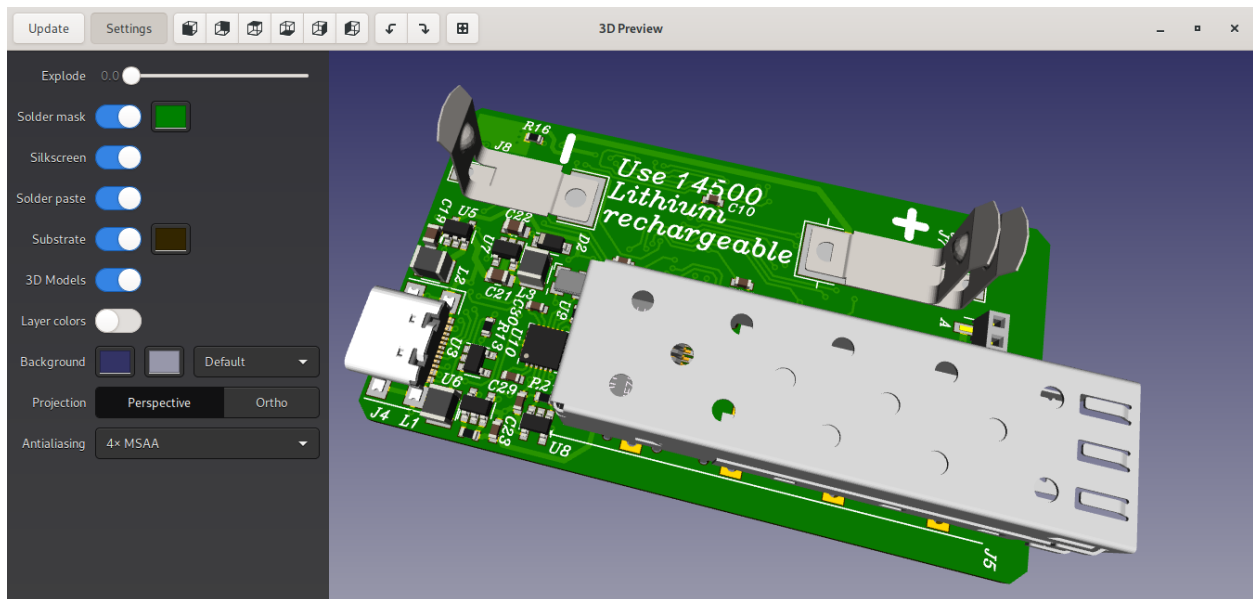
Wrote layer Top Mask to gerber file /home/lukas/code/hubble/output/gerber/hubble.gts

## 2.13 Mechanical CAD integration

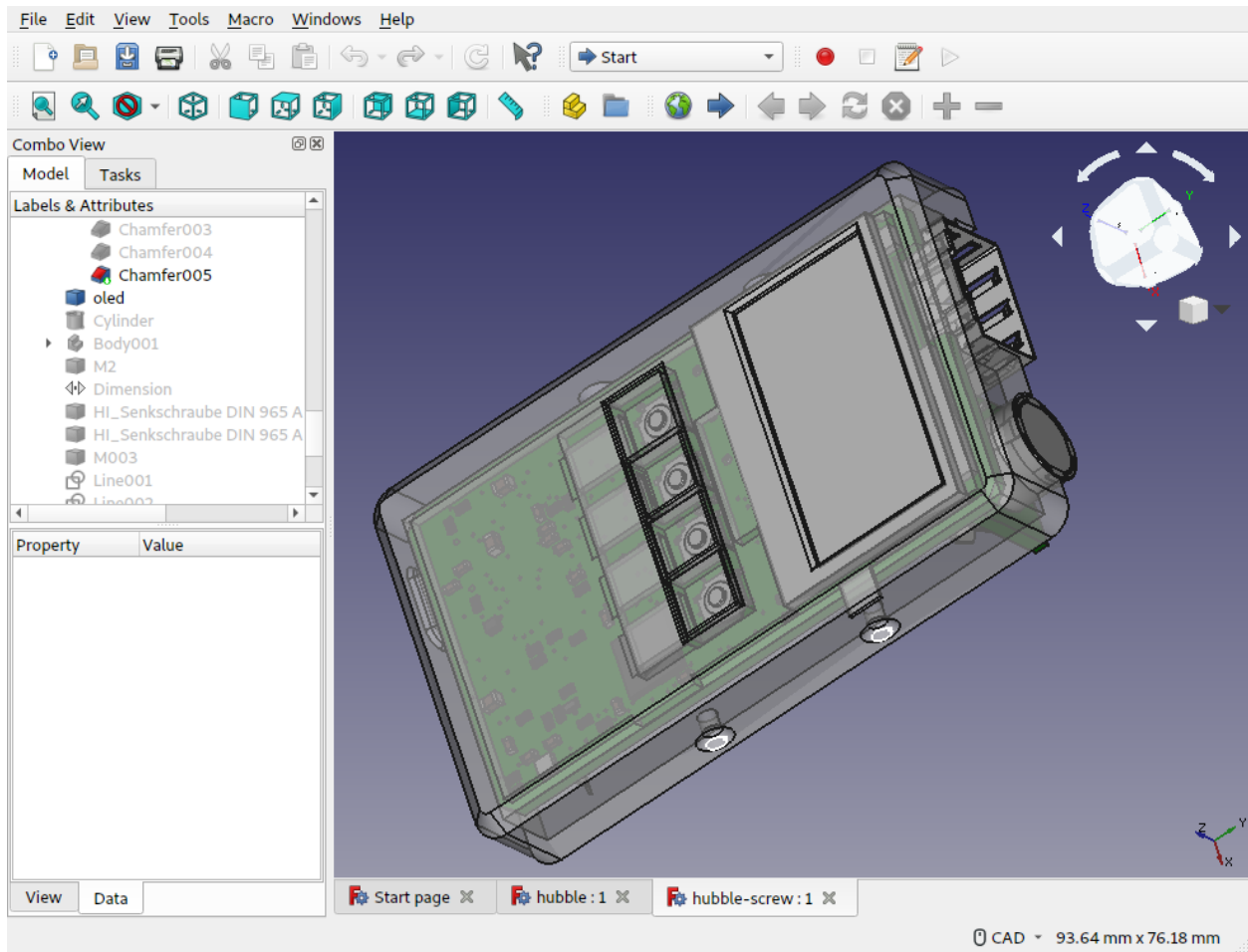
Extend packages into the 3rd dimension by adding a 3D model in industry-standard STEP format:



Look at your board as if you were holding it in your hands to make sure everything fits as intended:

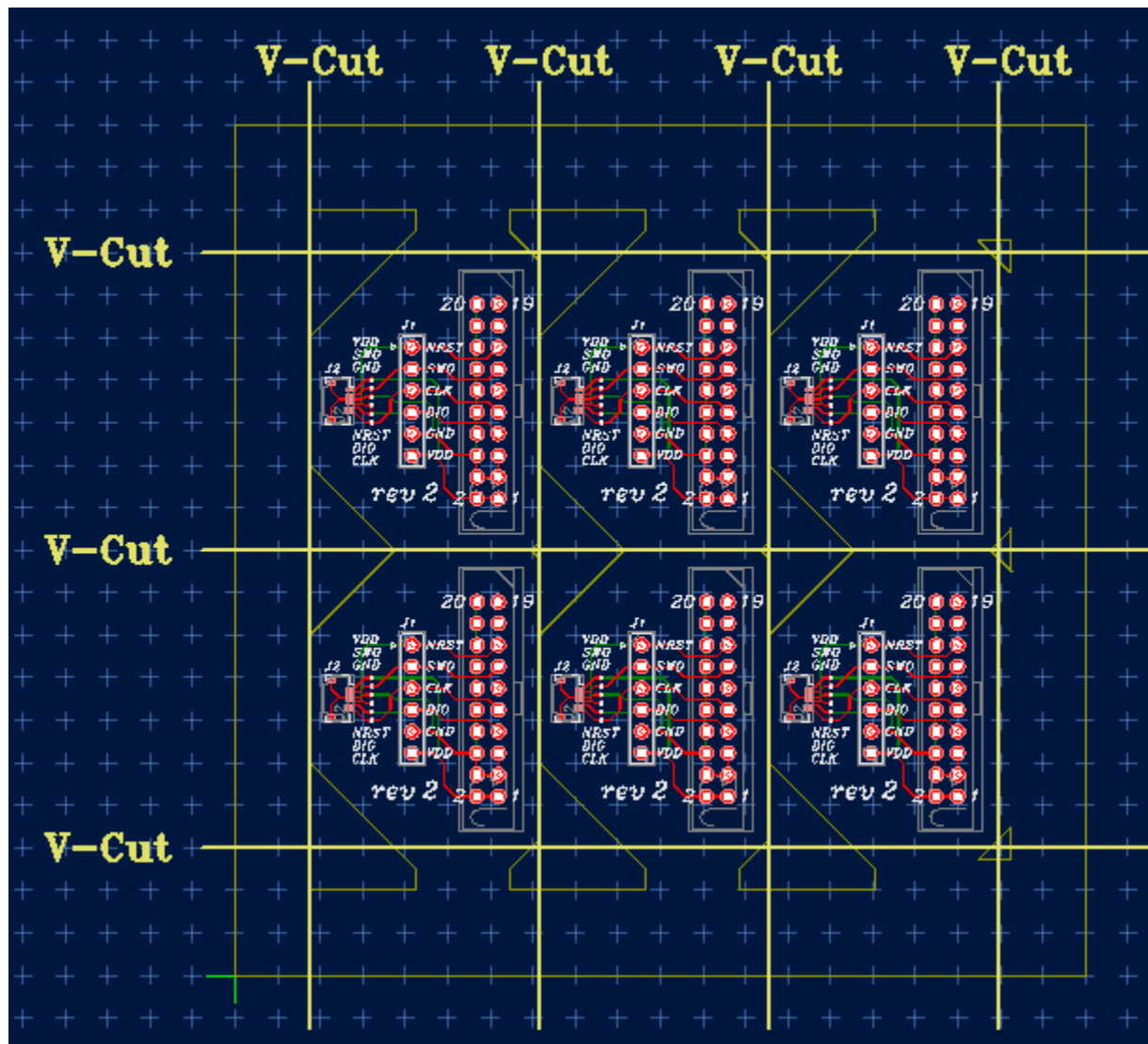


When the design is done, export the board and all models as STEP file for use in mechanical CAD:



## 2.14 Painless panelisation

Easily arrange multiple copies of one board or multiple boards on one panel to save money when ordering small PCBs:



All boards on a panel are linked to the original design, so they'll update when the the original design changes.

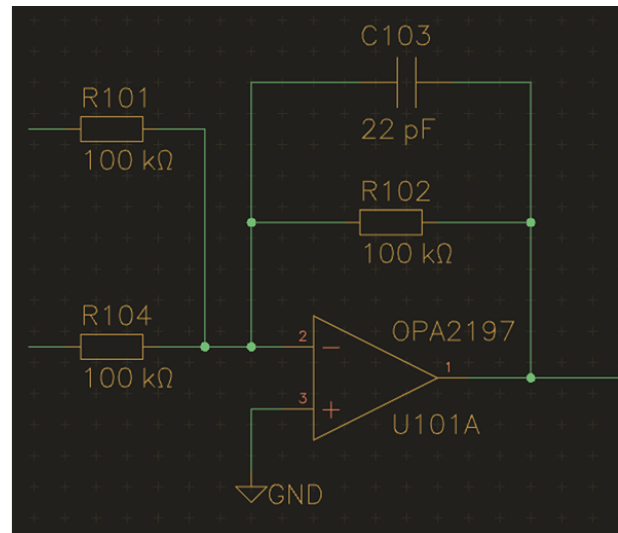
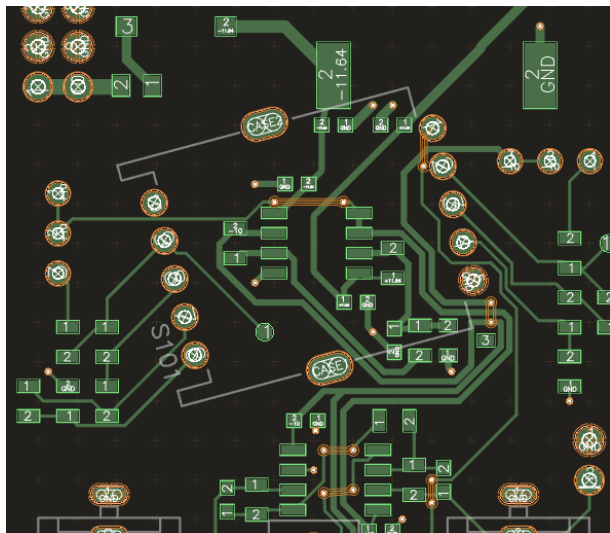
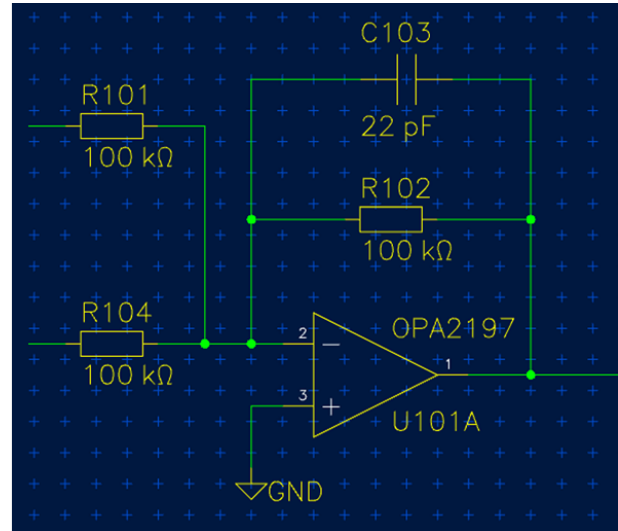
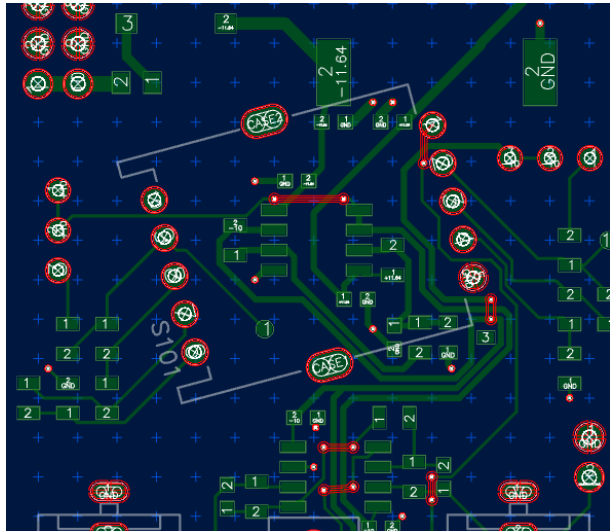
## 2.15 Versatile input device handling

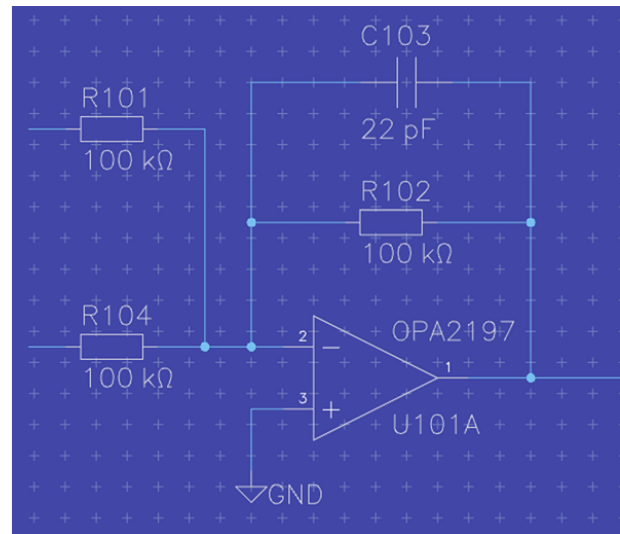
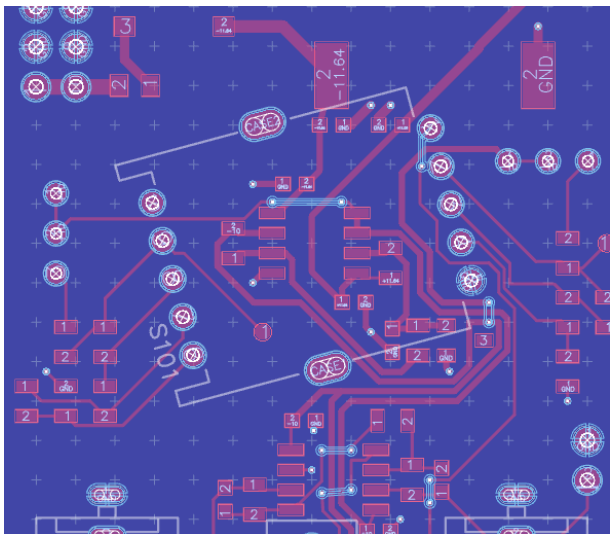
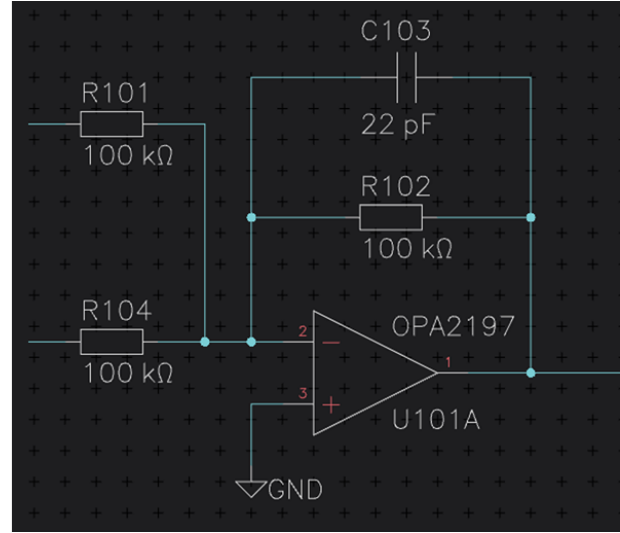
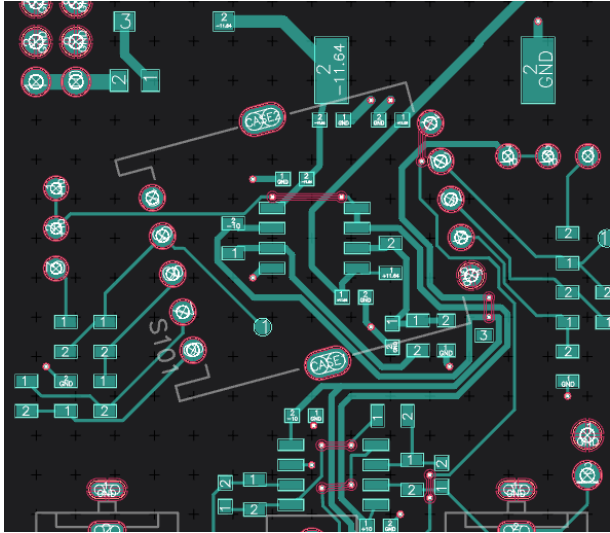
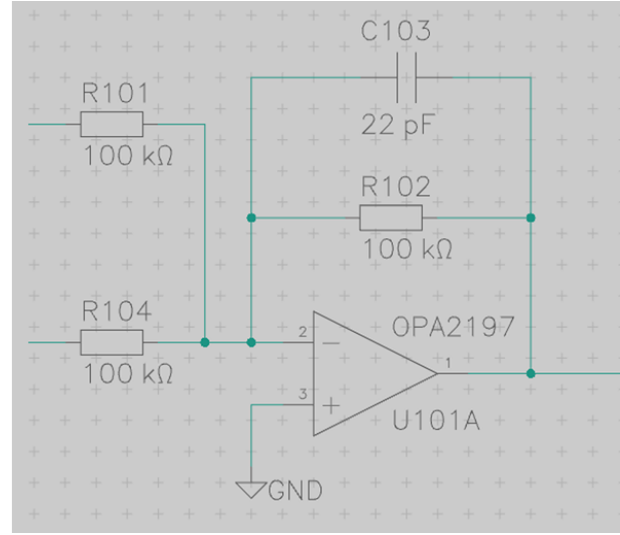
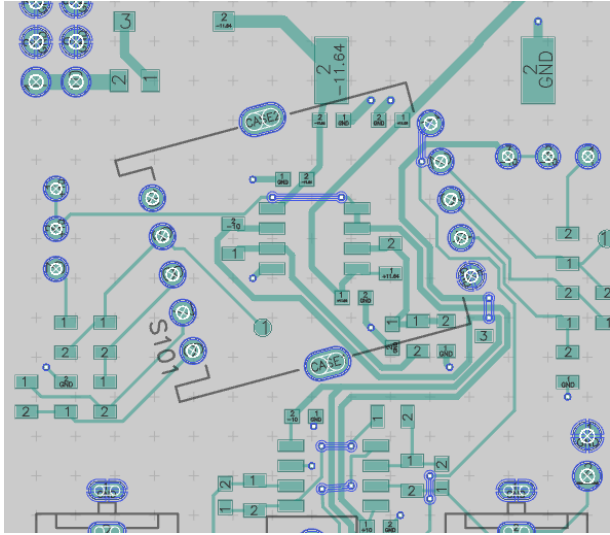
With Horizon EDA, you can make best use of modern laptop’s pointing devices. Apart from pixel-precision zooming and panning using touchpads or trackpoints, you can directly manipulate 2D and 3D views using touchscreen gestures such as pinch-to-zoom.



## 2.16 Make it yours

Decoration affects people, and people are different – do your own thing or select from the existing color schemes.





Keyboard shortcuts are fully customizable as well.

## 2.17 There's much more

- OpenGL-accelerated rendering
- Undo/redo
- Copy/paste, even between instances
- Filled planes
- Arbitrary pad shapes
- Import DXF Artwork
- Export a Bill of Materials (BOM)
- Export pick&place files



## NON-GOALS

To limit the project's focus, some things are explicitly out of scope.

### 3.1 Autorouter

Writing a good autorouter is a lot of work that's more well spent on other aspects of the application as experience has shown that an autorouter is rarely useful for small to medium-sized boards.

### 3.2 Simulation

Schematic design for PCBs and schematic design for SPICE-type simulation are very different as in that schematics for simulation will often simplify aspects of the real world such as replacing an ADC input with its equivalent circuit. On a personal side I'm perfectly happy with LTSpice in terms of user interface and SPICE core and don't see much scope for new developments in that space.

### 3.3 Raytracer

Other EDA applications recently gained a custom raytracer for rendering pretty 3D visualisations of circuit boards. For horizon that's out of scope as the OpenGL-based 3D view is pretty enough for checking for the board for issues such as forgotten solder mask and getting an idea of what it'll look like assembled. Any more pretty visualisation is best taken care of by exporting to 3D modelling software such as blender.

### 3.4 On File formats

Many people complain that there's no commonly agreed on standard format for schematics and boards in the industry. The file formats for these are application files formats, meaning that they'll need to support each and every knob and button the application has. Adopting another application's file format for horizon EDA would therefore result in horizon EDA being a bad clone of the other application.

JSON has been chosen as a serialization format as it directly maps to common data structures such as maps and arrays (opposed to XML) and is easily manipulated in almost every environment.

Next: *Installation*

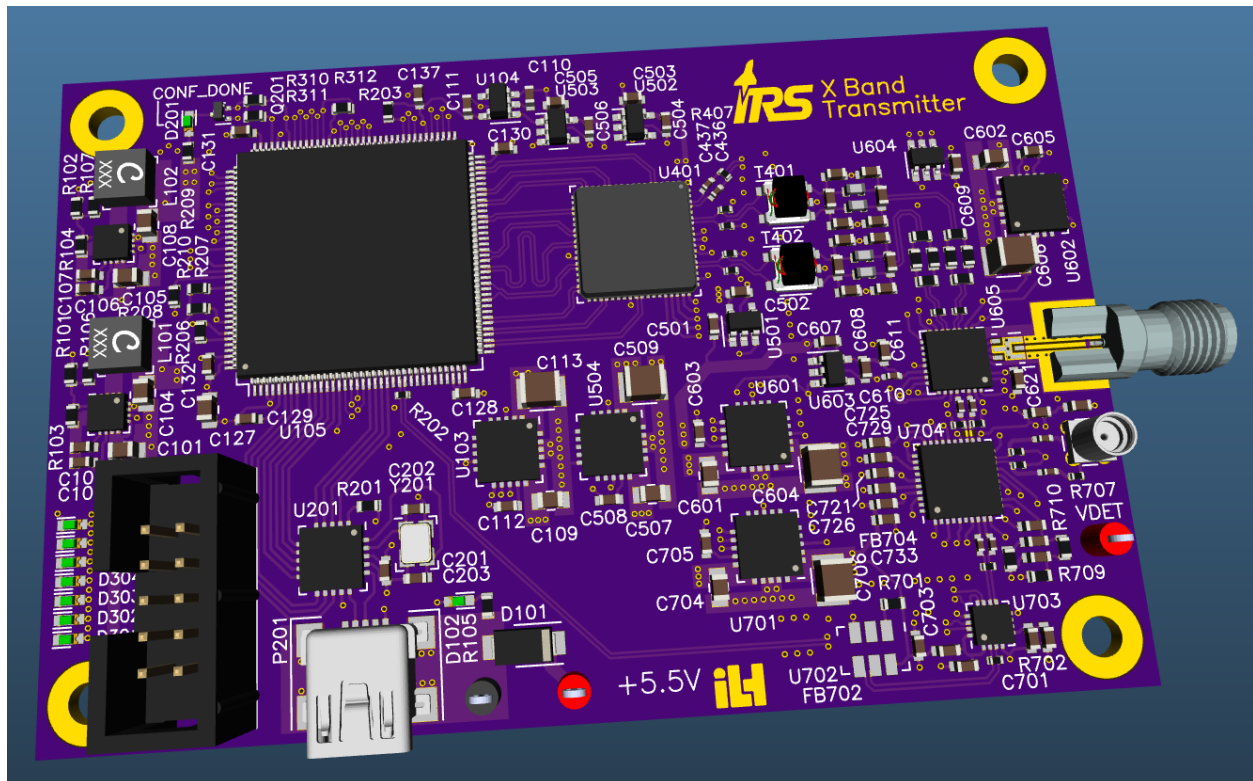


## MADE WITH HORIZON EDA

This list of projects made with Horizon EDA. Open a [Pull request](#) if you want to see your project on this page.

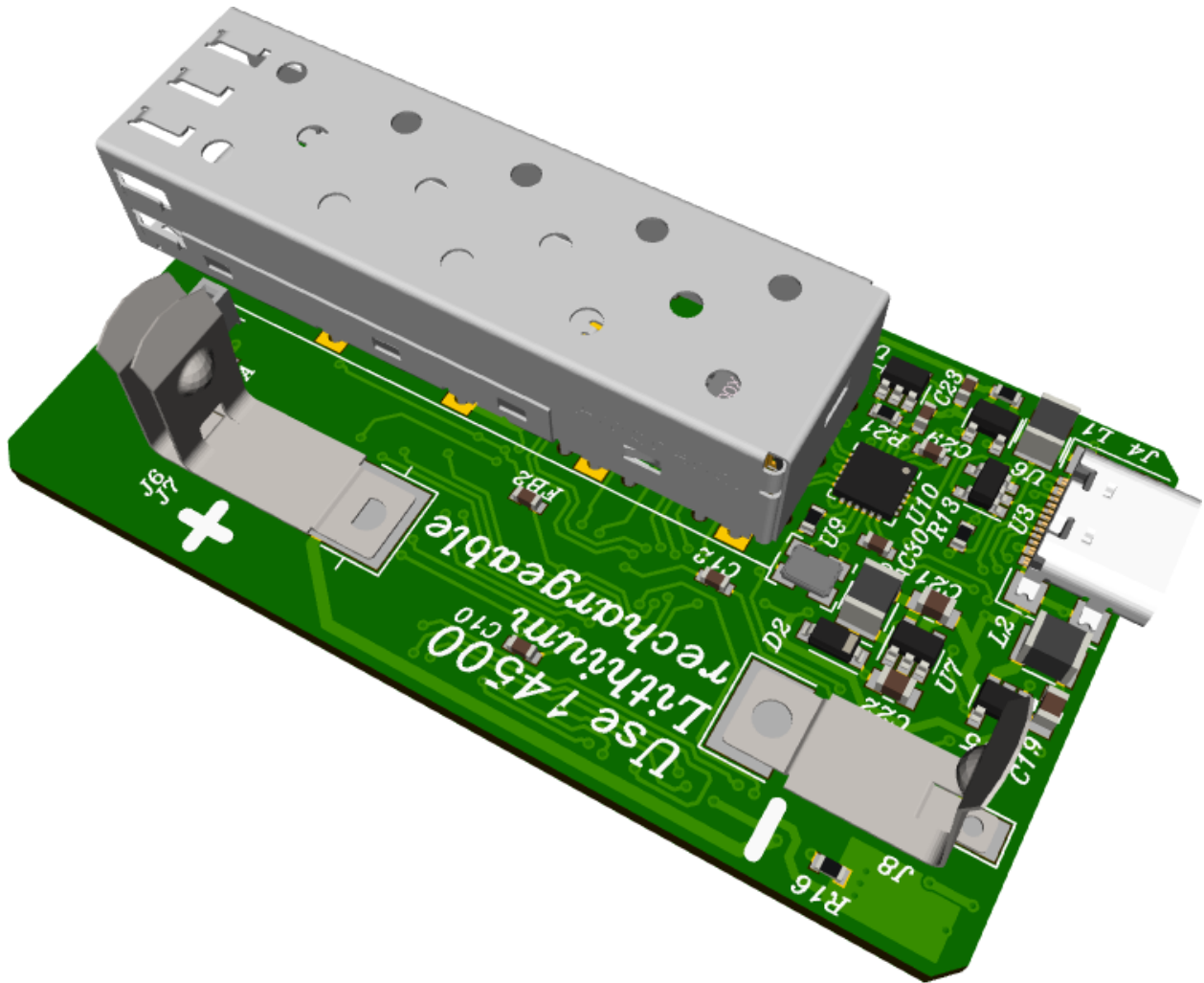
## 4.1 X-Band Transmitter

Repository



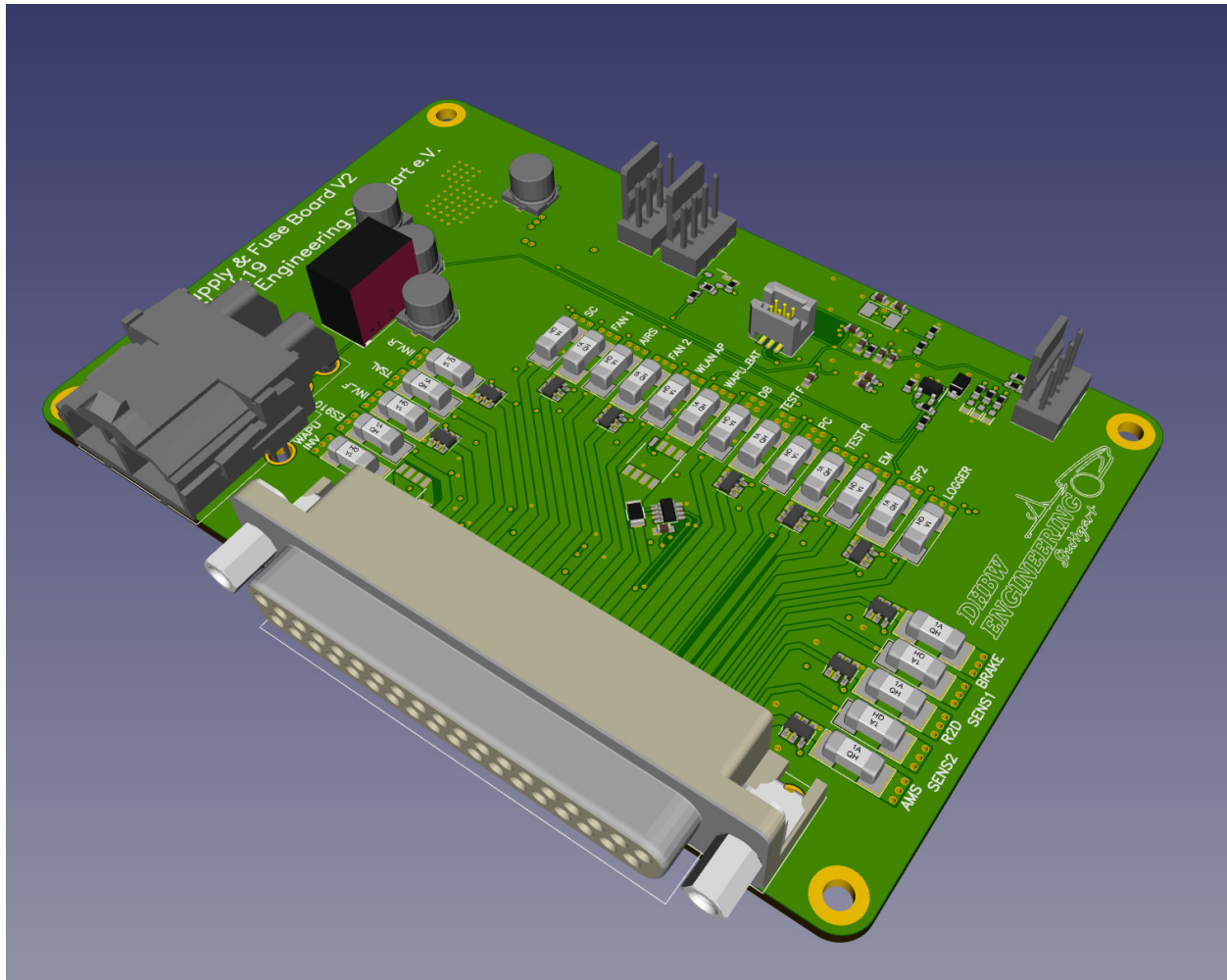
## 4.2 Hubble (SFP Multitool)

Repository





### 4.3 Fuse board of the Formula Student Racecar “eSleek19” of DHBW Engineering Stuttgart e.V.





## INSTALLATION

So you wanna give horizon a test drive? Great! Here's how.

### 5.1 Stable release

A known-good snapshot from ongoing development.

#### 5.1.1 Windows

Download and run the MSI installer from [GitHub releases](#).

#### 5.1.2 Linux

Keep in mind that binary packages provided by your distribution might be out of date.

##### Flatpak

Get the latest stable release from [Flathub](#).

##### Debian, Ubuntu

Debian builds are hosted on the [Selfnet mirror](#).

To add the repository, first download the [GPG key](#) and save it somewhere, for example in `/usr/local/share/keyrings`.

Then add this line to `/etc/apt/sources.list` or a new file in `/etc/apt/sources.list.d/`, replacing `<distro>` with either `ubuntu` or `debian` and `<release>` with the release name you're running.

See the [directory listing](#) for the list of currently supported distributions/releases.

```
deb [signed-by=/usr/local/share/keyrings/horizon-eda-debian.gpg] https://mirror.selfnet.  
de/horizon-eda/<distro>-<release>/ <release> main
```

```
sudo apt-get install horizon-eda-upstream
```

### Arch Linux

For Arch Linux, there's an [AUR package](#).

### NixOS

Horizon EDA is [packaged](#) for NixOS.

```
nix-env -iA horizon-eda
```

### 5.1.3 FreeBSD

Horizon EDA is available in the FreeBSD [ports](#).

```
sudo pkg install horizon-eda
```

### Build from source

If you want to compile it yourself, download the source tarball from [GitHub releases](#) and follow the instructions in *Building on Linux*.

## 5.2 Development version

Usually works, but might break occasionally, so use at your own risk. Recommended if you want to get the latest in features and bug fixes.

### 5.2.1 Windows

Grab the latest build from the [Selfnet mirror](#) and unzip it somewhere. Note that these are 64bit binaries. The download URL is also shown on GitHub Actions.

### 5.2.2 Linux

Clone the repository and see *Building on Linux* for instructions on how to build horizon on Linux.

### 5.2.3 FreeBSD

Clone the repository and see *Building on FreeBSD* for instructions on how to build horizon on FreeBSD.

Next: *Setup a pool*

## SETTING UP A POOL

There are two ways of setting up a pool. If you are familiar with the version control system git and want to be in direct control of your repository you can use the git workflow below, if not, just use the pool manager.

### 6.1 Git

If you're familiar with git, just clone clone [horizon-pool](#) somewhere and you're good to go. You're supposed to use git to keep your local copy up to date and submit new parts.

### 6.2 Pool manager

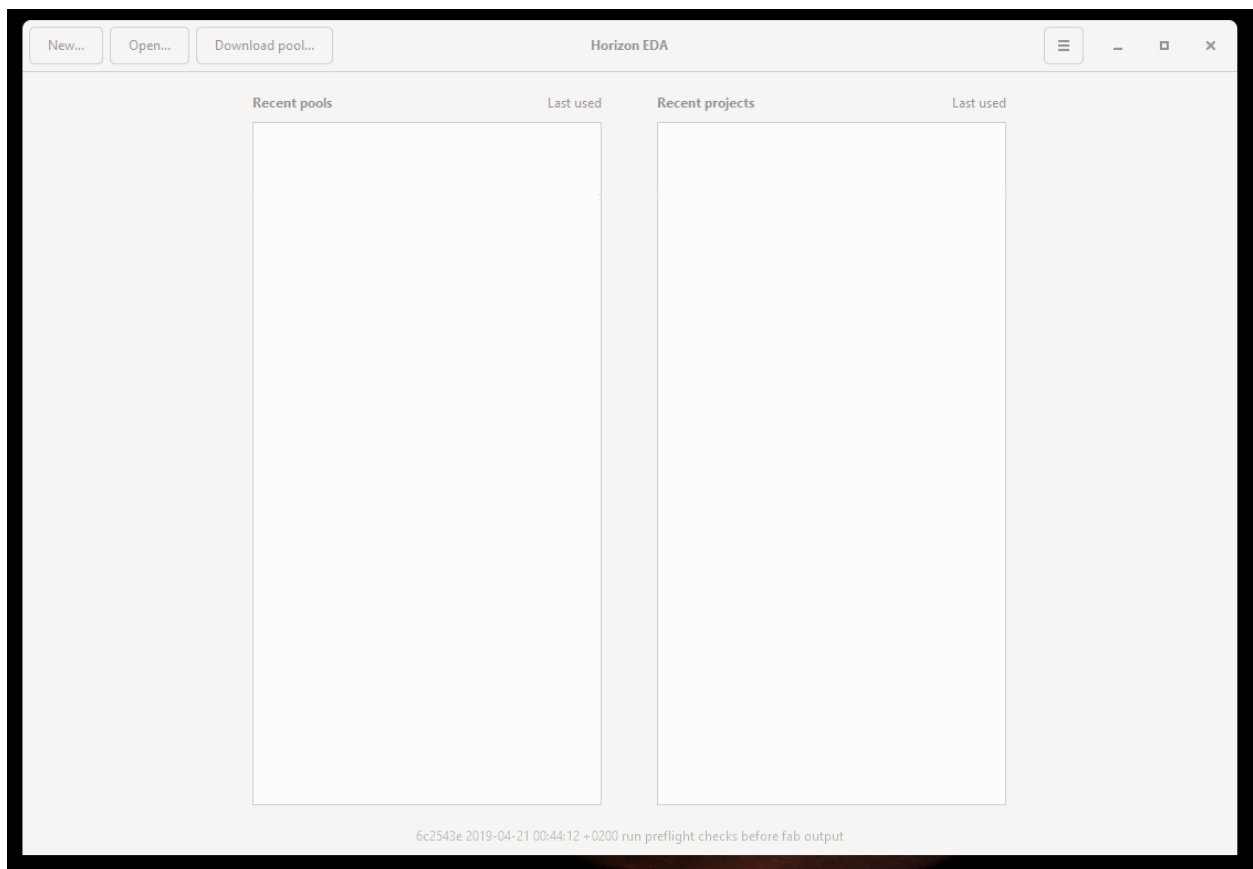
Don't know how to git? No problem! Double-click `horizon-eda.exe` or launch `./horizon-eda` from your shell and click on 'Download...' to download the pool. The default pool `horizon-eda/horizon-pool` is the one you want to use. The pool manager will assist you in keeping your pool up-to-date, see the "Remote" tab. It will also assist you by creating a fork, branches, commits and pull requests on your behalf so you can contribute to the pool without any git knowledge.

Next: *Create a new Project*



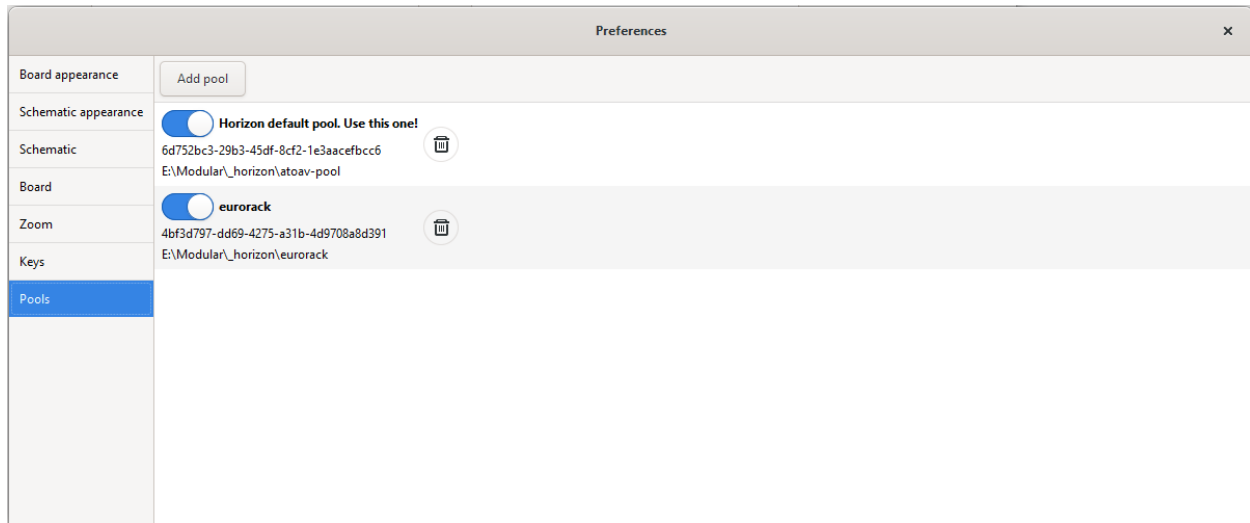
## CREATE A NEW PROJECT

When you start Horizon EDA you should see a window like this one



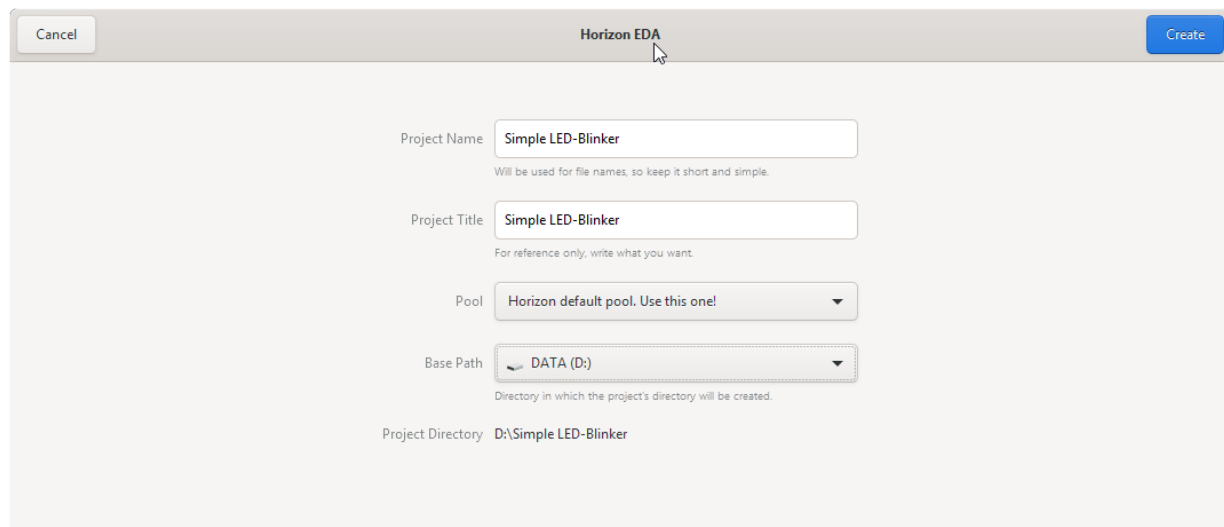
Make sure you have a pool set up and added

Click the menu icon in the top right corner and open the preferences dialog. In the Pool section selected the pool you want to use for your project. If you don't have a pool here you can open a pools pool.json by clicking on "Open..." in the main window. When you're done the preferences dialog should look like this:



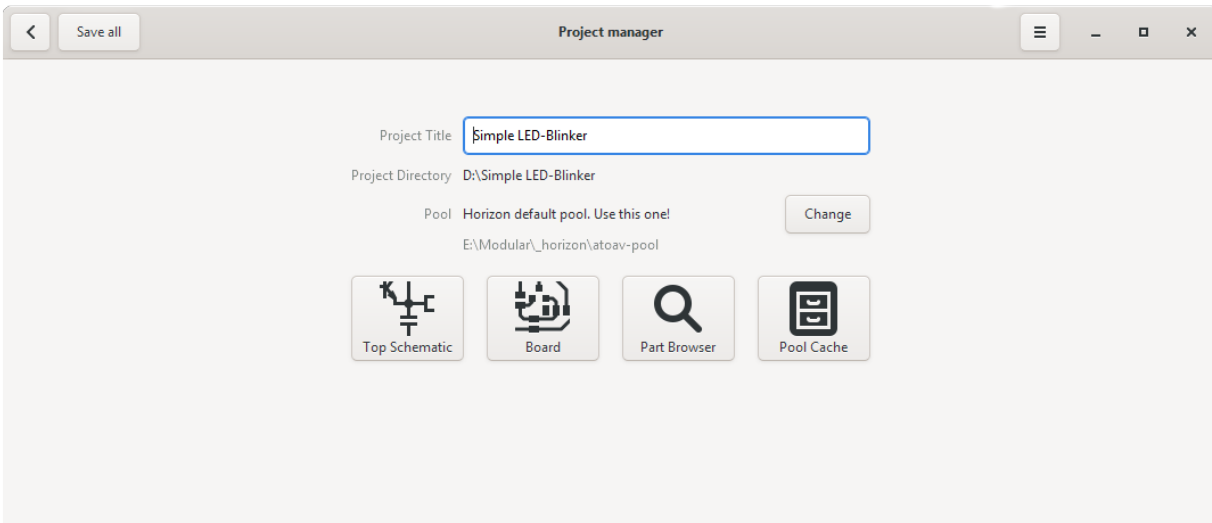
Now, hit “New...” and then “Project” to create a new project.

In the Project Dialogue Window you can select a project name and a location where you want to store the project folder. Additionally you can change the active pool:



In the Project manager window that opens now, you can create a Schematic, a Board, Browse for Parts or Manage the Pool Cache.





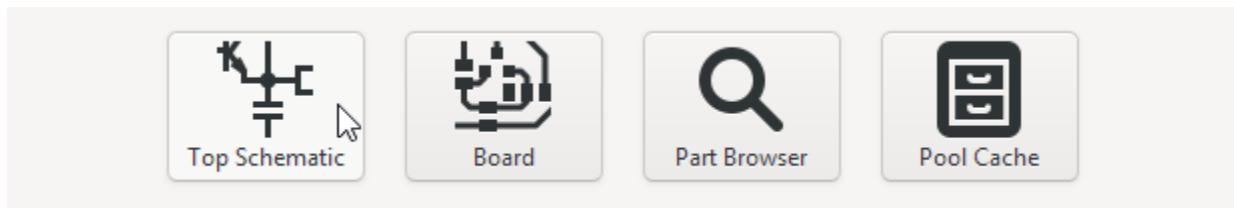
The Pool Cache stores a copy of all parts that you used in your project and helps you to protect your projects from outside changes (e.g. by updates in the Pool). When you *want* to update the parts, or remove unused parts you can do so in the Pool Cache Window.

Next: [Draw a Schematic](#)



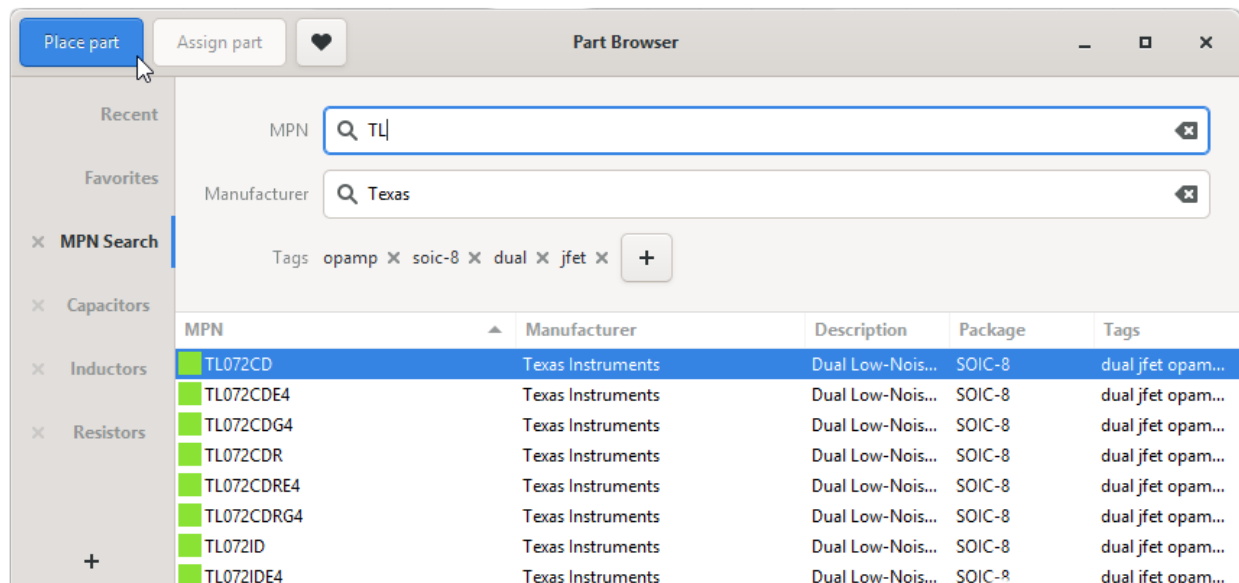
## DRAW A SCHEMATIC

In the Project Manager click on the Top Schematic button, to open the Schematic editor:



### 8.1 Placing Parts

You can then start placing parts by clicking the “Place Part” button in the top bar, or by simply typing `p p` (remember it as “Place Part”). This opens the Part Browser, where you can search your whole pool for the Parts you need and place them on the schematic.



## 8.2 Connecting Parts

Once you placed parts you can activate the “Draw net line” tool by typing `n` and drawing the connections. You can also simply drag out a pin and begin connecting pins:

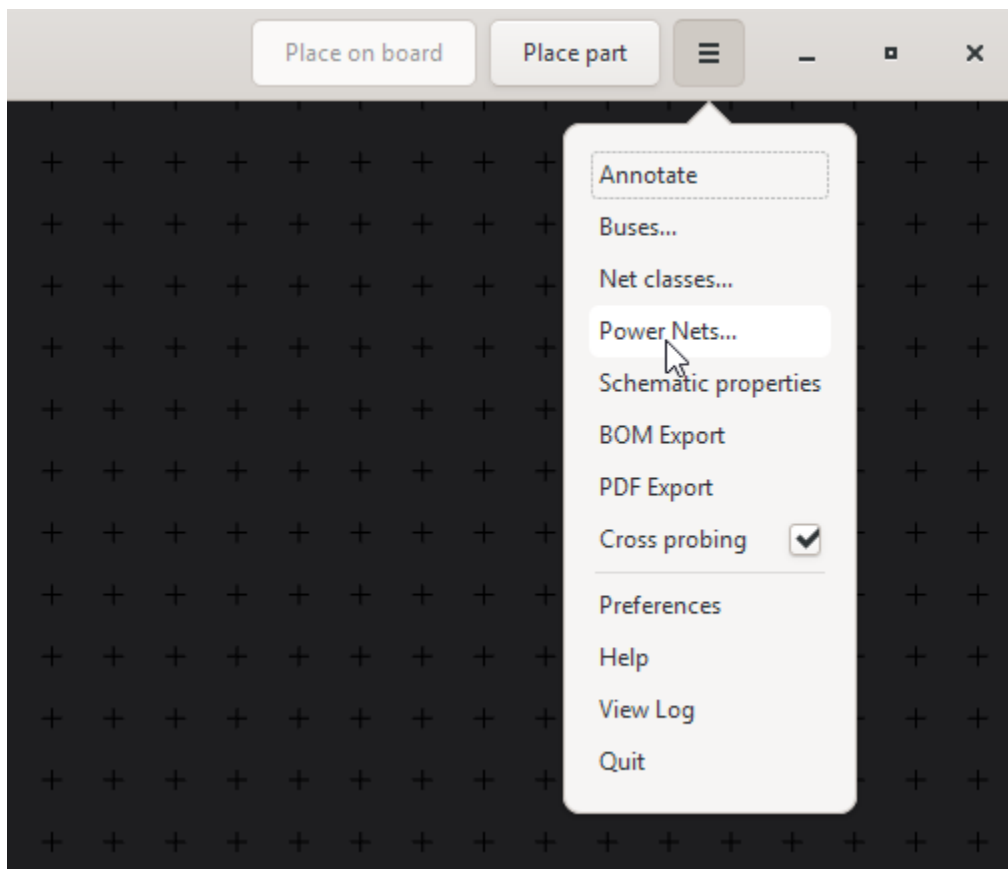
## 8.3 Basic Movement

You can move parts and nets by selecting them and typing `m` or by using the `←/↑/↓/→` arrow keys. Rotate with `r` and mirror with `e`

If you forget any of these keys, just press `Spacebar` to open the *Spacebar Menu* and search for the command you are looking for.

## 8.4 Power Nets

Because Circuits make much more sense if you use power nets you can create new power nets by clicking on the entry in the application menu:

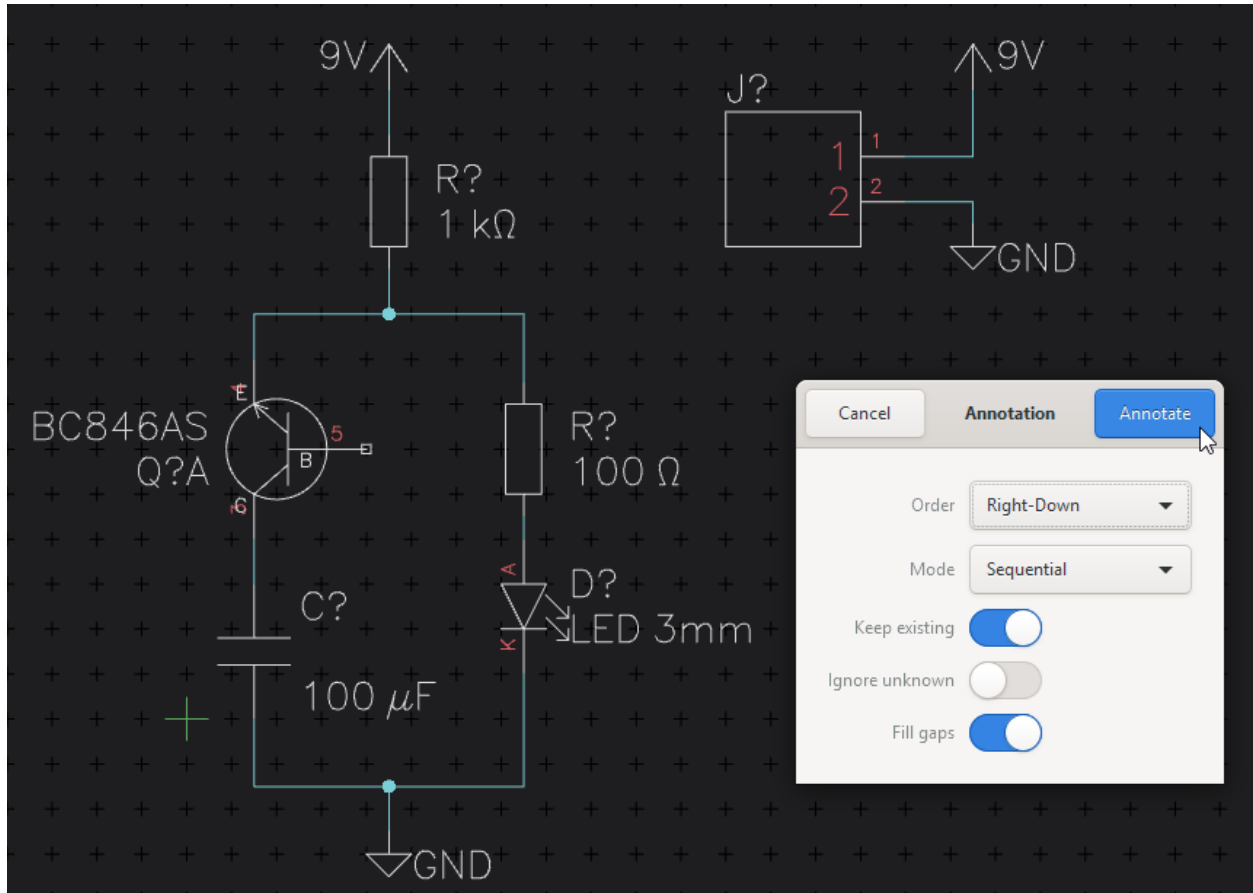


In the window that pops up press the “Add power net” button, give the net a name and select a symbol style.

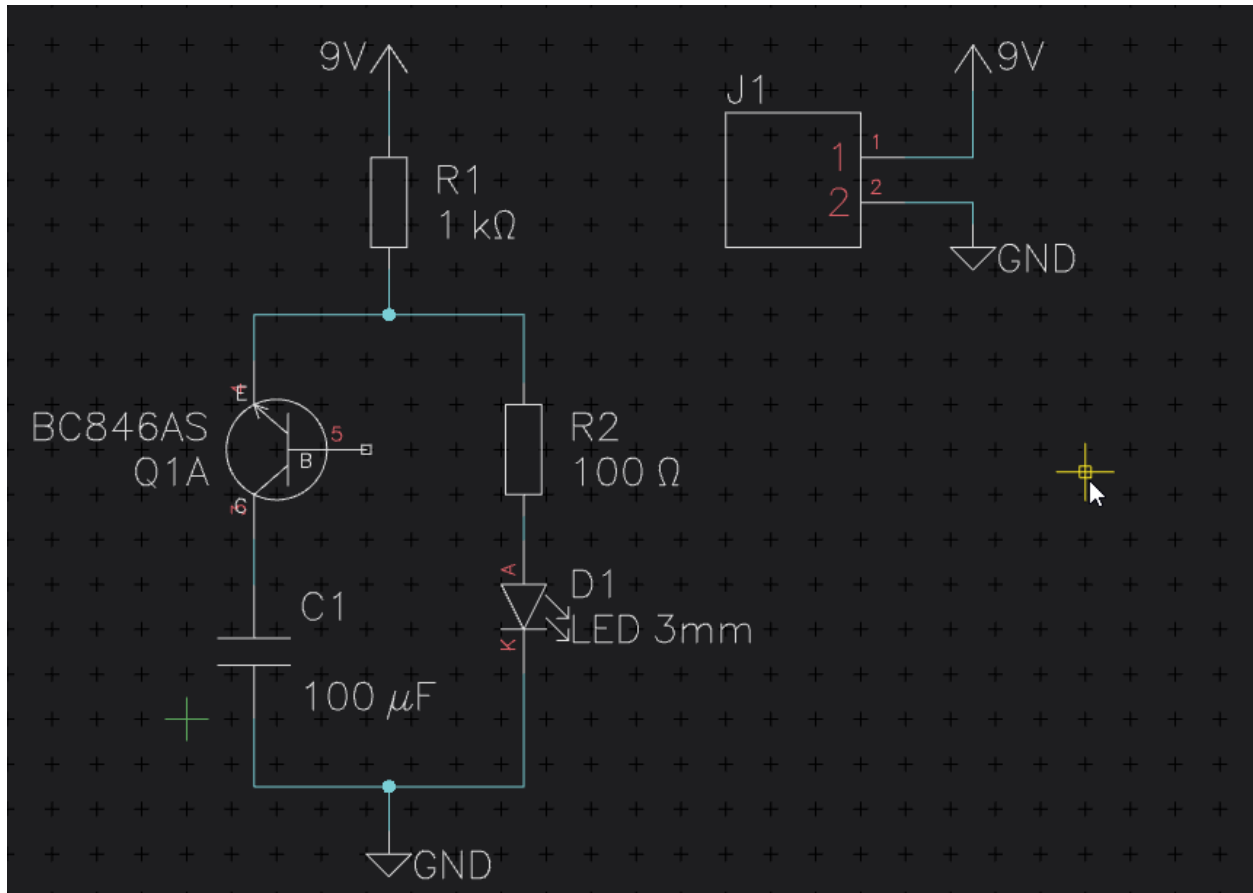
Place power nets with the “Place power symbol” Tool (type `p o`).

## 8.5 Annotation

Run the Annotation Tool to give all un-annotated parts (signified by the “?” in their reference designator) a proper number. So we go from this:



to this:

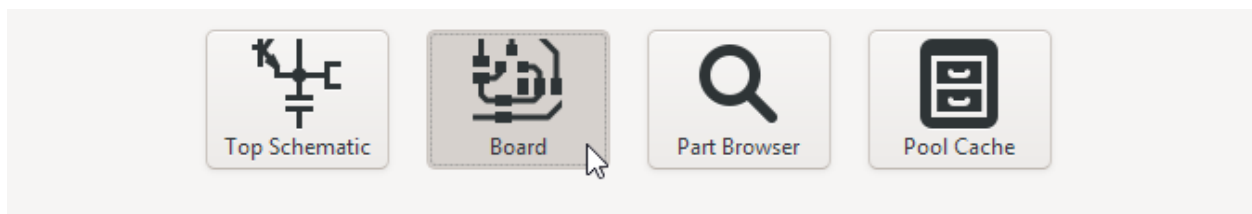


Once you are done, click on “Save” so you can start to layout the schematic you just created.

Next: [Create a Board](#)

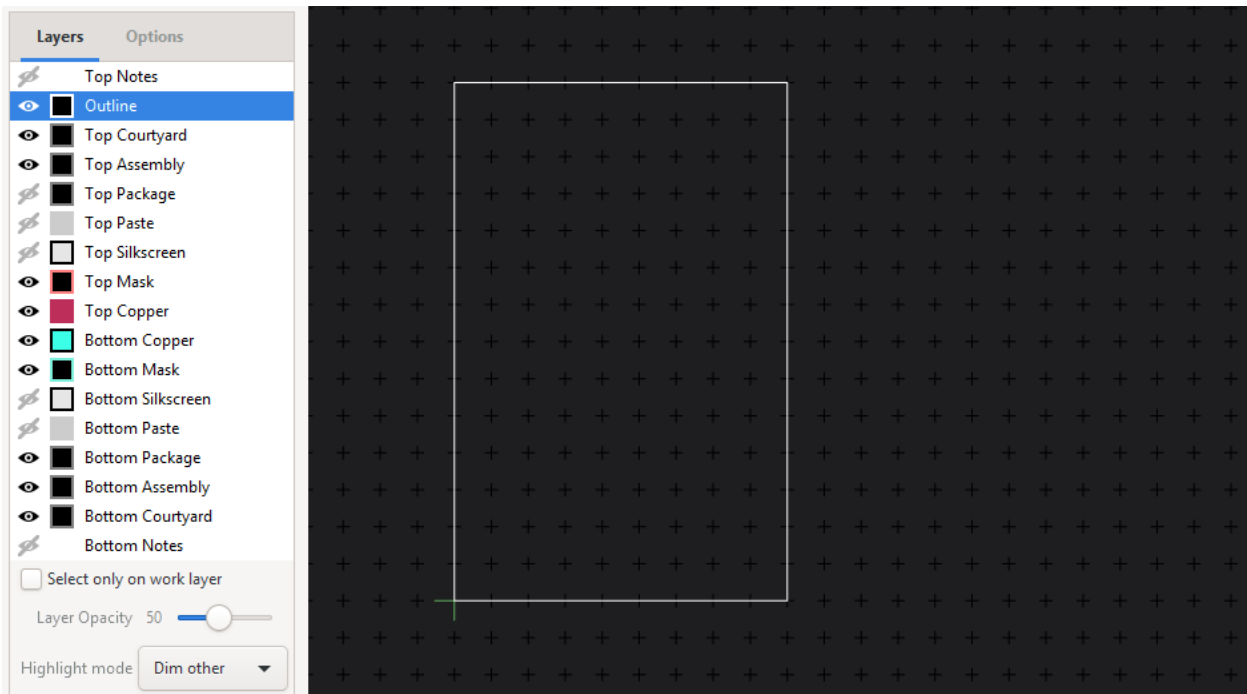
## CREATE A BOARD

Open the board editor from the Project manager.



### 9.1 Draw a Board Outline

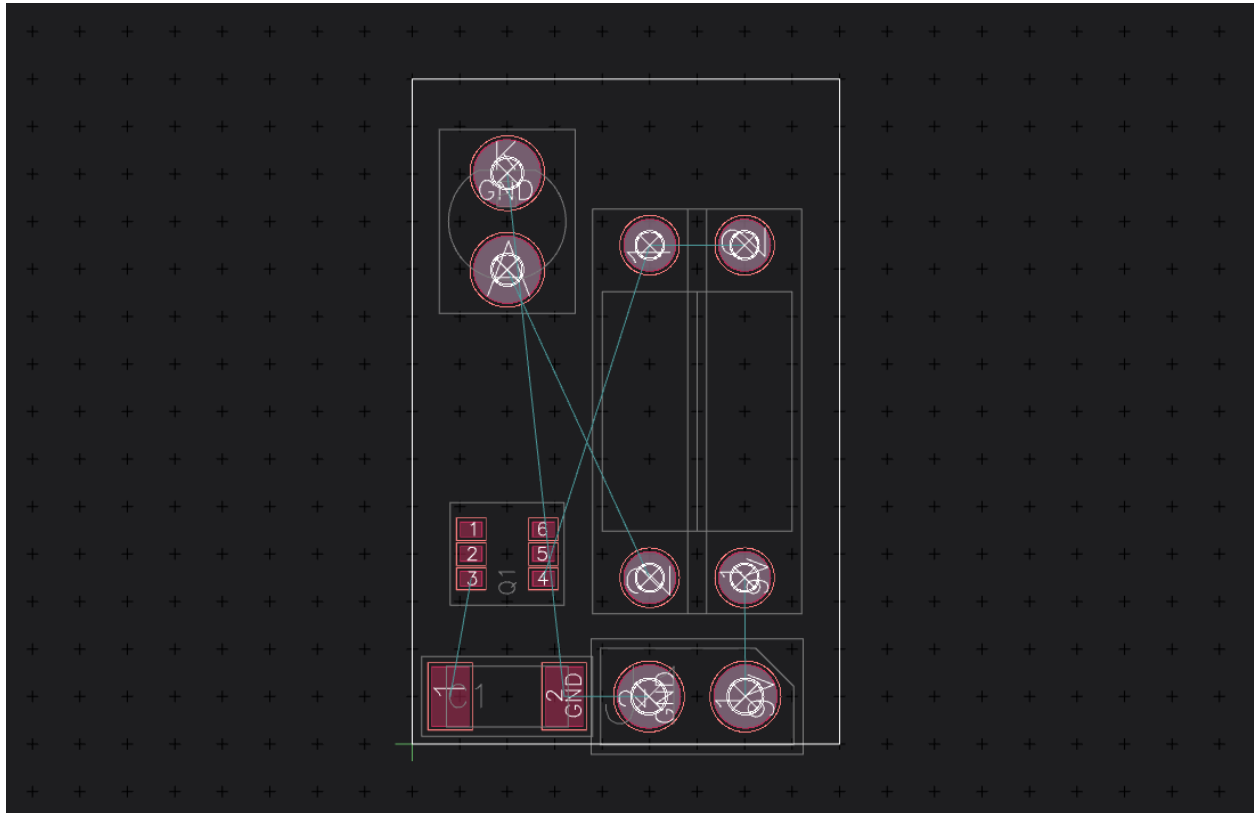
Draw a outline by first selecting the “Outline” Layer in the Layers Panel and then selecting the “Draw polygon rectangle” Tool by typing `d Y` (or use the Spacebar menu):



## 9.2 Place the Packages

Place packages on the board using by typing `p p` (this time it stands for “Place Package”). If you still have the Schematic Editor open you can also select a Symbol and use the “Place on Board” Tool by clicking it’s button in the top bar.

If you select Parts in the Schematic Editor they will also be highlighted in the Board Editor. If you change something in the Schematic Editor bring the changes from the schematic to the board, by saving the schematic and clicking ‘reload netlist’ in the board editor.



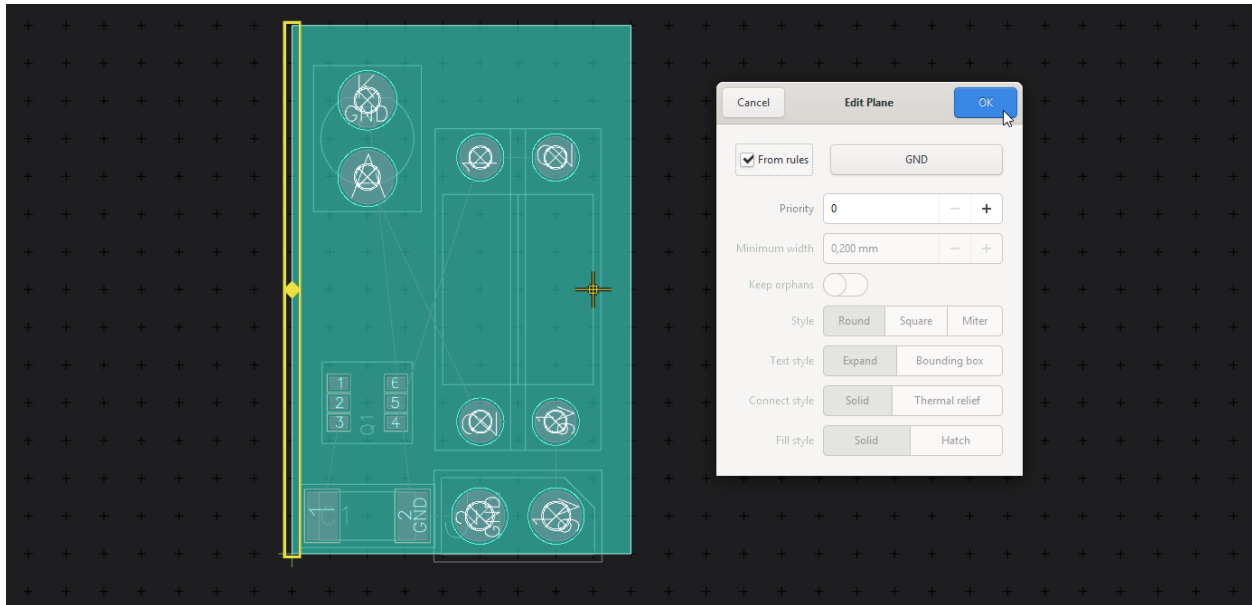
If you want to place a part on the bottom of the board, press `e` to flip it to the bottom side. If you are working a lot on the bottom you can use the “View bottom” tool to flip the whole board around.

## 9.3 Add Planes

If you want to use a Ground Plane now would be a good time to add it:

1. Select the layer you want to have the plane on (Top- or Bottom-Copper)
2. Draw a polygon rectangle by typing `d Y`
3. Right-Click one of the Edges of the newly added polygon and select “Add Plane”
4. In the Window that pops up select the GND net (if it is not there you didn’t add the power net in the Schematic)





If the solid filled color of the plane annoys you, you can switch the display style to something else by clicking onto the colored square left of the layer:

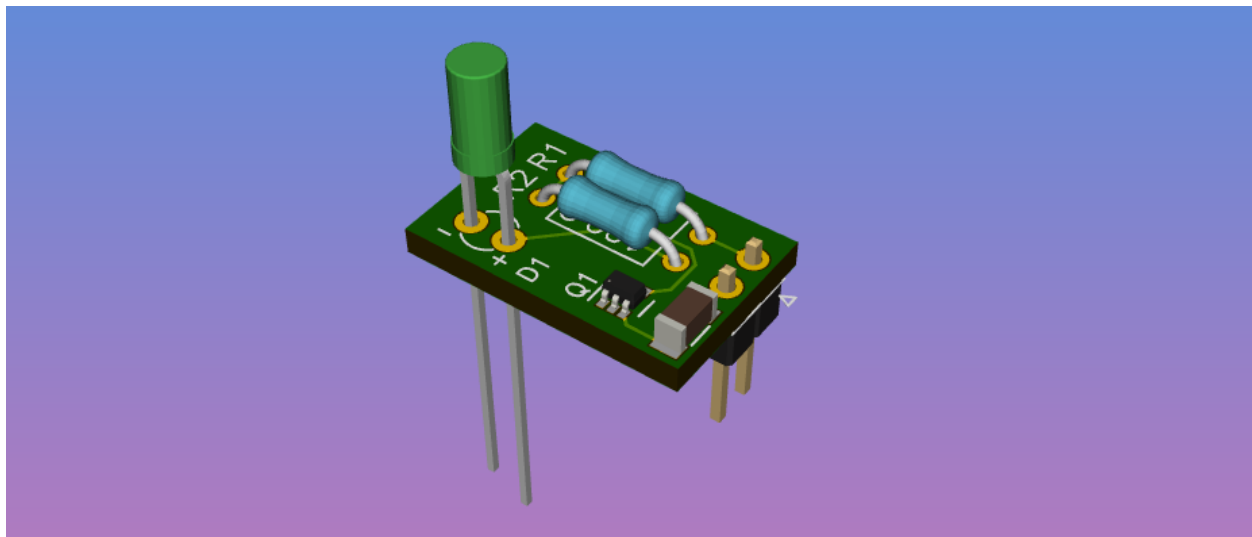
## 9.4 Route the Nets

To route tracks between the parts you connected in the Schematic, type **x** and drag from any pin that starts an airwire:

The track will always be routed on the layer you have selected, you can quickly select the Top Copper Layer by pressing **1** and the Bottom Copper Layer by pressing **2**.

If you want to change sides in the middle of a track you can place a via by pressing **v** and then the number of the layer you want to continue.

Once you are done, check out the 3D view of your part, and export Gerbers using the menu point “Fabrication Output” in the application menu.



See *Board Editor* for more details on the board editor.

Next: *Look at a Example Project*

## **EXAMPLE PROJECT**

Instead of starting your own project, you can also download the [design files for an X-Band transmitter](#). To open it, point the project manager to the `ddstx.hprj` file. Make sure that you extract all the files contained in that repository.

Next: *Basic Editor Usage: Tools*

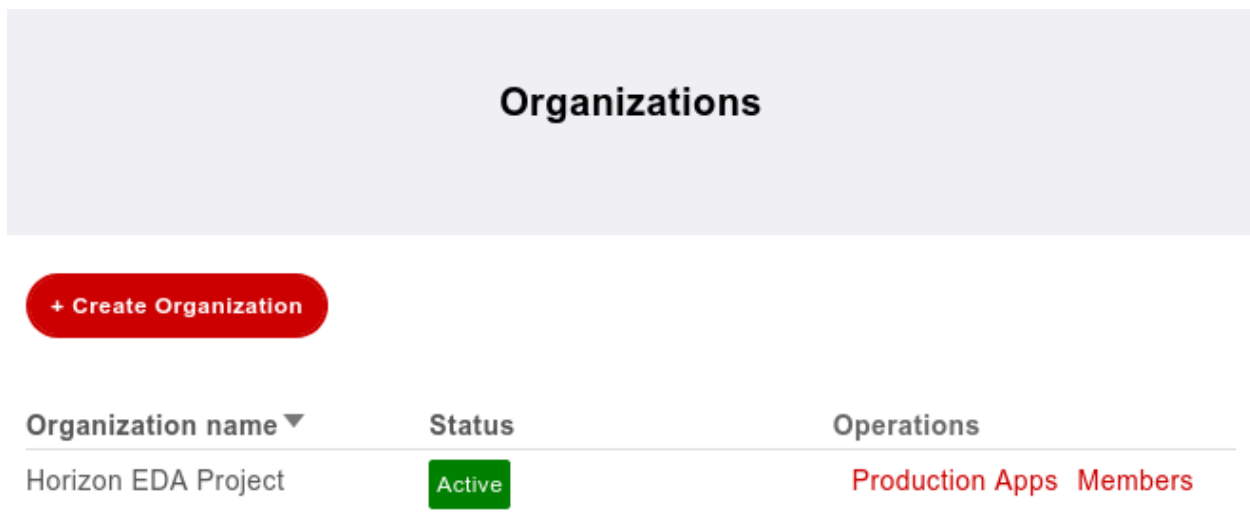


## DIGI-KEY API INTEGRATION

For displaying stock information in part browsers, Horizon EDA can make use of Digi-Key's API. To use it, you need to register an application with Digi-Key.

### 11.1 Registering

Go to [developer.digikey.com/teams](https://developer.digikey.com/teams) and create an organisation, the name doesn't matter.



**Organizations**

[+ Create Organization](#)

Organization name ▼	Status	Operations
Horizon EDA Project	Active	<a href="#">Production Apps</a> <a href="#">Members</a>

With the organisation in place, click on “Production Apps” in the organisations overview and create a new production app. Set <https://horizon-eda.org/oauth.html> as the OAuth Callback. Name and description don't matter. Select the “Product Information” product and create the app.

## Add Production App to Horizon EDA Project

Members of an Organization can see all Production Apps that are created for that Organization. Any API Products that are selected below grant production-level access to those API Products.

Production App name

OAuth Callback  [Learn more about OAuth 2.0](#)

Description

Select one or more Production products:

Product Information

☐ ☒

None Product Information

Click on the newly-created app and paste the Client ID and Secret into the preferences.

### Horizon EDA production app

[View](#) [Edit](#) [Delete](#)

Production App status

Approved

Callback URL

https://horizon-eda.org/oauth.html

Credential

Client ID

 [Hide key](#)

APIs

Product Information

Enabled

Client Secret

 [Hide key](#)

Issued

4 days 20 hours ago

Expires

Never

Key Status

Approved

Board appearance

Schematic appearance

Editor

Stock info

Keys

In-tool keys

Preferences

Provider

Digi-Key

See [the docs](#) for obtaining Client ID and secret.

Client ID

Client secret

Max. price breaks

3

-

+

Cache

5 days

-

+

Site

Germany (DE)

Currency

Euro (EUR)

Not every currency is supported on every site. Site and currency only apply to new queries.

Log in

No token, sign in to obtain one.

Clear cache

## 11.2 Logging in

Finally, you need to log in to get the tokens required to access the API. Do so by clicking the “Log in” button and following the instructions in the window that opens.





## TOOLS

In order to provide a unified user experience and enable code reuse, the editors for symbols, schematic, padstack, package and board are all based on the interactive manipulator.

### 12.1 How to select the right tool

To edit the things on screen, use the tools. Tools can be started in multiple ways:

- By typing in the tool's key sequence. Available key sequences are listed by clicking the help button in the top right corner or typing ?.
- By using the *Spacebar Menu*. Just start typing for the tool you're looking for or browse through the list.
- By right clicking an object. This will bring up context menu listing all the tools relevant to what's selected.
- By pressing one of the dedicated buttons in the top bar.

After having started a tool, it receives all keyboard and mouse input, till the tool is finished or you end it by pressing Esc. Look at the bar that appeared at the bottom of the canvas to see what pressing keys will to in this particular tool.

Next: *Basic Navigation: Spacebar Menu*



## SPACEBAR MENU

Horizon's Editors rely on fast and intuitive single key shortcuts and key sequences alike. While this is very powerful for those who use it daily, it could be cumbersome for those who just started out or have long periods passing between each of their electronics projects.

In order to make the transition between beginner and power user an easy one, Horizon EDA has a Spacebar Menu, that (hence the name) pops up after you pressed **Spacebar**. In this menu all Tools you can select within the Editor's Canvas are listed. To help you getting faster the shortcuts and key sequences are listed alongside the tools. So if you find yourself in the position of having to go to the space bar menu over and over again, you can easily speed up things, by using the short cuts.

You can easily customize these shortcuts in the preferences window.

Next: *Grid*



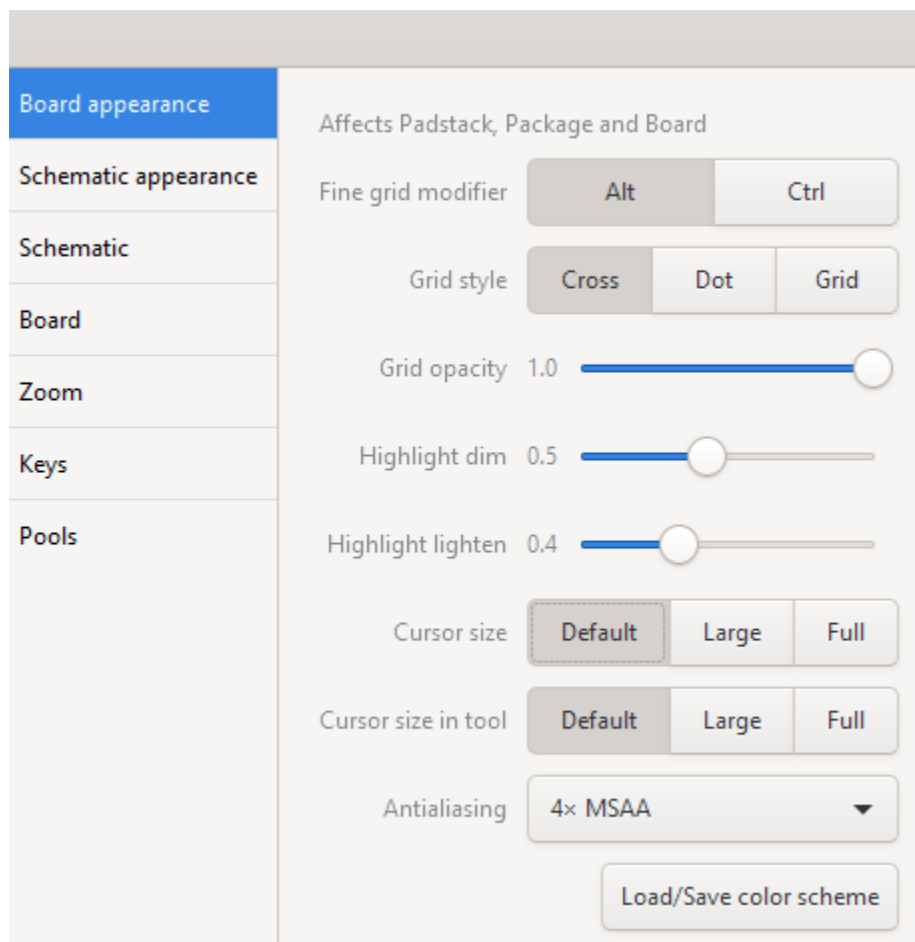
## GRID

All movements in Horizon happen on the grid. By holding **Alt** you get a finer grid (by a factor of 10).

You can change the size of the grid by changing the value in the numeric field on the top left of the window. Note that depending on a zoom level your grid may be resized by factors of two.

You can enter basic math operations into *any* numeric field in horizon, this makes it easy to divide a value by two, multiplying something by a certain factor, or adding a value to a coordinate to create a fixed offset. For more details, see [Numeric entries](#).

You can change the appearance of the grids and the cursor in the Preferences:

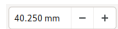


Next: *Drawing*

## NUMERIC ENTRIES

All numeric entries in horizon support basic two-operand math as well as some other goodies.

They look like these:

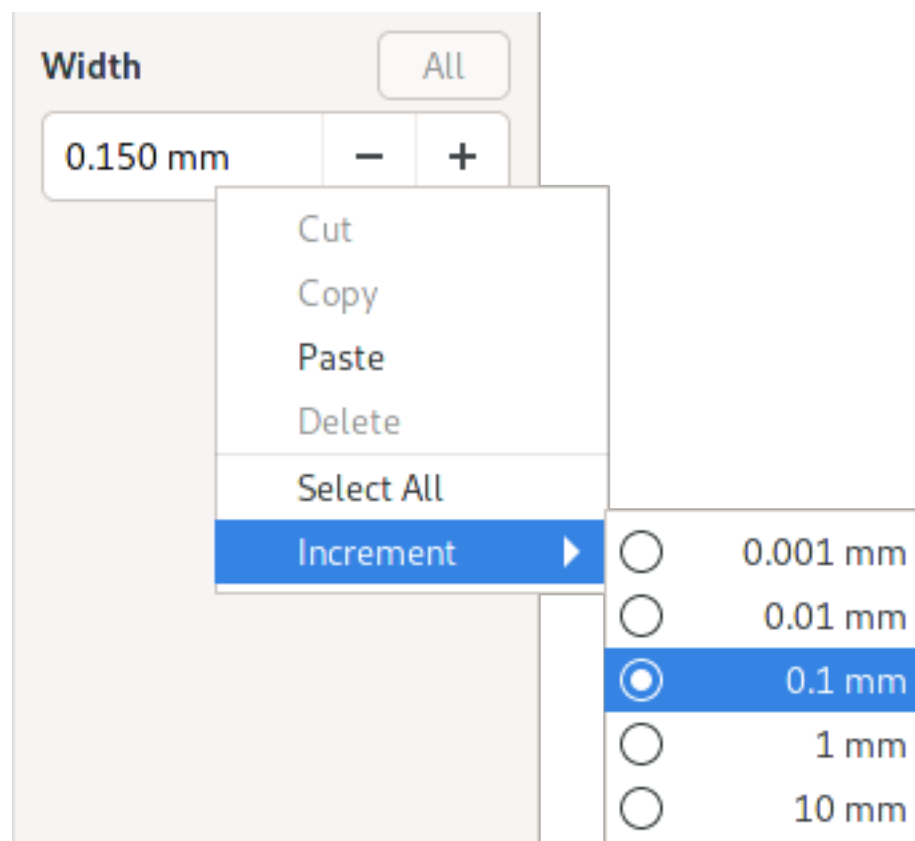


Both point and comma are recognized as a decimal separator regardless of locale settings. By default all numbers are treated as millimeters. Suffixing a number with an `i` or `in` will treat it as inches. Use `mi` or `mil` to convert from thous.

Additionally, two-operand infix math is supported, so you can to this:

- Addition: `1+2`
- Subtraction: `1-2`
- Multiplication: `3*2`
- Division: `3/2`
- Average  $\frac{a+b}{2}$ : `3|2`
- Adding mm and inch: `1in+2mm`

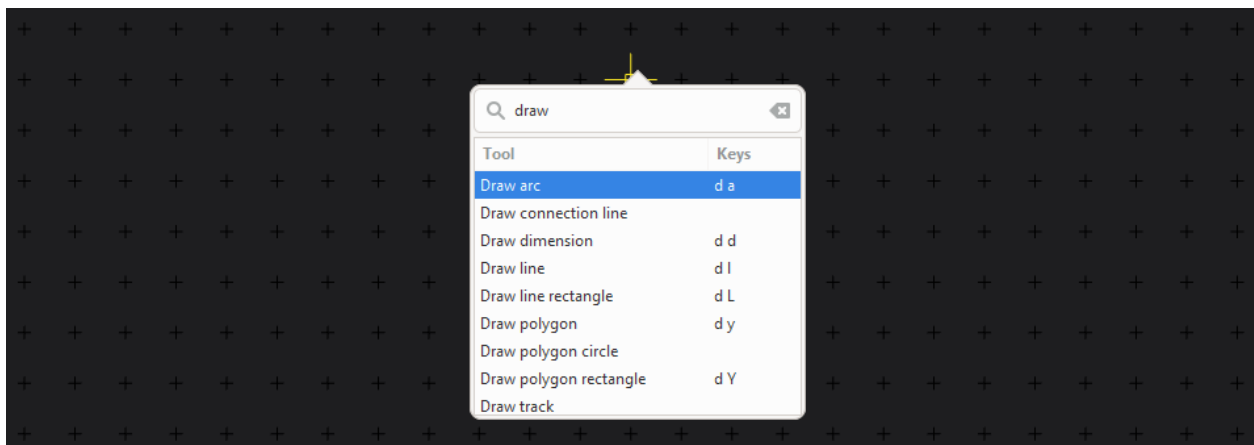
Right-click the entry to change the increment used by the +/- buttons:





## DRAWING

The different Editors in Horizon EDA share a set of different Drawing tools. Here you can see the ones available in the Board Editor:



The drawing tools are mainly split in three categories:

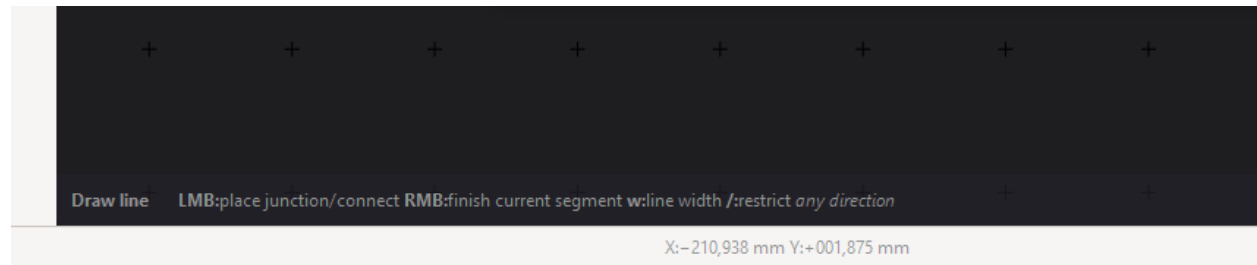
- lines
- polygons
- special (like draw track, draw dimension etc.)

Lines are usually used for everything visual (e.g. Silkscreens) while polygons are used for all things where it matters that the thing you draw results in a closed shape (pads, board outlines, package assembly and courtyard layers, etc.)

### 16.1 Draw Line

To draw a line simply select the “Draw line” Action in the spacebar menu or type the key sequence `d l` (think: “draw line”) – once you click anywhere, you start the first point of the line right at the place where your manipulator was. You might notice that the point snaps to the grid. If you want a finer grid hold the `Alt` key down while placing points. Left mouse button places more and more points, while the right mouse button (or pressing `Esc`) will finish the lines.

You might also have noticed the Action Bar at the left bottom of the editor window:



The Action bar will show additional keys you can press to change the behaviour of the tool. In this case you could press **w** to change the width of the stroke or **/** to restrict the movement of the manipulator to one direction.

## 16.2 Draw Line Rectangle

To save your time, there is also a “Draw line rectangle” Action, which can also be invoked by typing **d L**. Per default you first set the rectangles center point and then one of the corner points. By pressing **c** you can change this behaviour and set two diagonally opposed corner points instead.

## 16.3 Draw Arc

The “Draw arc” tool is straightforward it draws line arcs by setting three points (in this order): start point, end point and the center point. You can also use the key sequence **d a** to start the tool. If you want your arc to flip direction, press **e** before putting down the center point.

## 16.4 Draw Polygon

When drawing polygons with **d y** you can set a series of points by clicking, until you either press **Esc** or use the right mouse button. You can make the next edge of the polygon an arc by pressing **a**. Just like with the “Draw Arc” tool set the endpoint first and the center point after. Before setting the center point you can flip the arc direction with **e** and finally you set the end point of the arc.

## 16.5 Draw Polygon Rectangle

Similar to the “Draw line rectangle” Tool there is a “Draw polygon rectangle” tool. Invoke it by typing **d Y**. Just like with the according line Tool you can switch between the different draw modes (Center/Corner) by pressing **c**.

There are some differences though: you can set a corner radius by pressing **r** and entering a value and you can choose a decoration by pressing **d**. These decorations are used to mark the pin 1 on a Package’s assembly layer. You can cycle through different decoration positions by pressing **p** and set the size of the decoration by pressing **s** and entering a value.

## 16.6 Draw Polygon Circle

For ease of use there is also a “Draw polygon circle” Tool. With the first click you set the circles center point and by setting the second point you set the radius. You can also enter a radius by pressing `r` and entering a value.

## 16.7 Draw Dimension

Sometimes it can be useful to add dimension information to certain parts. You can do so by using the “Draw dimension” ActToolion. Start it by typing `d d`, selecting the first and the second point and dragging it out. If the numeric value is on the wrong side, you can fix it by selecting the dimension and flipping it with the `e` key.

Dimensions can also be set to specified length by selecting the end that’s supposed to move and activating the “Enter Datum” tool (press Enter). You can then snap other items to the end points of the dimension.

Next: [Selection](#)



## SELECTION

Selection throughout the Editors of Horizon EDA is very straightforward and there are no surprises here. No matter how basic it seems, the different modes and filters could save you quite some time.

### 17.1 Basics

Initially, the “hover select” mode is active. It simply selects the smallest object under the cursor. Leftclick or drag with the clicked button to select objects permanently. Hit Esc for returning to hover select mode.

If you want to select multiple things by clicking keep Hit Ctrl pressed while you click on things – this also works to deselect things you accidentally selected while dragging a selection box.

### 17.2 Selection Mode

You can change how dragging the left mouse button behaves in the lower left corner of the editor window. Available are three different modes (Box, Lasso, Paint) with four different selection characteristics (Auto, Include Origin, Touch Box, Include Box).

### 17.3 Selection Filter

If you have to select many things, it can sometimes be handy to only select certain classes of objects. This is what the selection filter is for. You can open it up via the spacebar menu or by pressing Ctrl+i

Double click on an item to select only that item. Click on the check mark button on the top left to select all items.

Next: *Moving and the interactive Manipulator*



## MOVING AND THE INTERACTIVE MANIPULATOR

### 18.1 Move with Mouse

You can move parts and nets by selecting them and typing **m**, press **Alt** to move on a fine grid. The movement of the selected object has its origin (or pivot) point right at the spot where the interactive manipulator was, when you started the move tool.

This can be used to our advantage, because it allows us to move objects that have been placed on a finer grid without snapping them back to the coarse grid.

### 18.2 Move with Keyboard

By using the **←/↑/↓/→** arrow keys you can move the selected object one grid step into the desired direction. If you press **Alt** at the same time you will move the part on the fine grid (1/10 of the original grid). Because this is also a tool, you can cancel it by pressing **Esc** and finish it by pressing **Enter** or clicking onto a empty spot.

### 18.3 Move Exactly

The “Move exactly” tool allows you to enter numbers to move an object by an exact number of units (e.g. 1 mm) into the desired direction. You can start the tool by pressing **M**

### 18.4 Flip

You can flip symbols or parts by pressing **e**. If you use this on the board editor, it is the same as flipping the part onto the other side

### 18.5 Rotate

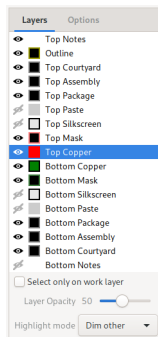
You can rotate objects by 90° if you press **r** or select the “rotate” tool. If you like to rotate a object by something else (e.g. 30° or 45°) use the “Rotate arbitrary” tool. The pivot point of the rotation is always the location of the interactive manipulator, so make sure it is at the right spot





## LAYERS

Board, package and padstack editor use the widget shown below to specify how layers are displayed.



The selected layer is called the “work layer” and is always visible and drawn on top of all other layers. Use Page up/down to move the work layer to the next or previous layer. Pressing 1 selects the “Top Copper” layer, 2 selects the “Bottom Copper” layer, 3 . . . 0 select inner layers if present.

Clicking on the eye toggles a layer’s visibility. Keep in mind that the work layer is always visible even if it’s set to be invisible.

Clicking on the colored box cycles through a layer’s display modes:

- Solid color: Layer is drawn with outlines and fill, overlapping filled objects on the same layer will appear brighter.
- Solid color with black border: Layer is drawn just filled, overlapping objects look the same as non-overlapping ones.
- Black with colored border: Layer is drawn just with outlines.
- Striped: Same as first, but areas are striped rather than filled.



## TIPS AND TRICKS

There are quite a few features that make day-to-day usage easier but might not be immediately obvious.

### 20.1 Quickly start tracks and net lines

Drag away from a pad or pin to start drawing a track or a net line.

### 20.2 Duplicating objects

Hold down **Ctrl** and drag selected objects to duplicate them.

### 20.3 Panning without a middle mouse button

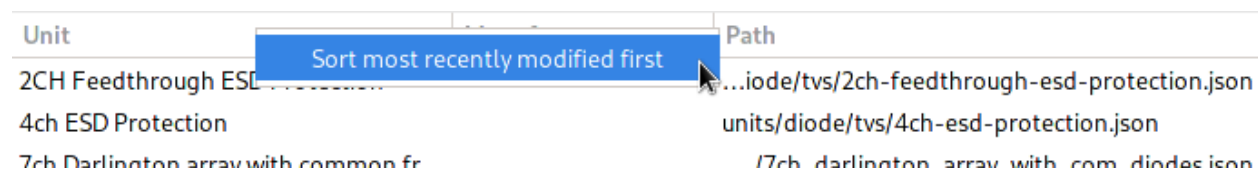
Drag with the left mouse button while holding down **shift** to pan the viewport.

### 20.4 Versatile length input

All input fields for lengths support more than just typing in millimeters. See *Numeric Entries* for what else they can do.

### 20.5 Sorting pool items by modification time

Right click the column headers in any pool browser to sort most recently modified items first:



The screenshot shows a table with two columns: 'Unit' and 'Path'. A right-click context menu is open over the 'Unit' header, displaying the option 'Sort most recently modified first'. The table contains three rows of data.

Unit	Path
2CH Feedthrough ESD protection	...iode/tvs/2ch-feedthrough-esd-protection.json
4ch ESD Protection	units/diode/tvs/4ch-esd-protection.json
7ch Darlington array with common fr	/7ch_darlington_array_with_com_diodes.json



## SCHEMATIC EDITOR

To launch the schematic editor click on “Top Schematic” in the project manager.

### 21.1 Placing parts

To place parts, open the part browser either by typing `p p` (place part) or clicking on the corresponding icon in the project manager. Once a part has been placed, it can be replaced by a part of the same entity by selecting it and using the “Assign part” button in the part browser window.

See *Project pool* for how to get updated items from pools.

### 21.2 Nets and net segments

Unlike some other schematic entry tools, horizon’s actually knows about nets and isn’t just about drawing lines that will eventually be transformed into nets when generating the netlist. A net may be represented by one or more net segment. A net segment is a set of net lines, junctions, pins, etc. all connected by net lines. Since the editor tracks which net segments belongs to which net, it provides feedback when an operation is about to merge two nets.

So when you see a net in the property editor after selecting a net line, the net in the property editors is the “whole” net and not just the net segment. That’s why renaming a net doesn’t change connectivity. To connect the pins on a net segment to a different net, use the “Move net segment to other/new net” tool.

A net label just displays the name of the net it is connected to and doesn’t set net names. To alert you about inconsistencies in the schematic that could result in unexpected connectivity, the schematic editor places warnings on the offending items.

### 21.3 Power symbols

The easiest way of creating a power net is using the “Manage Power Nets” Tool available from the hamburger menu. Then, use the tool “place power symbol”, to place a power symbol for this net. Power symbols force their net on the connected net segment. You can select from three styles of Power symbols in the aforementioned tool. The Antenna and Dot symbols can be placed either pointing up or down. The GND symbol can only point downwards.

## 21.4 Buses

To group related nets, use Buses. After creating a bus, add members to it. You can either assign existing nets or automatically name the newly created net by clicking on the arrow button next to it.

## 21.5 Diffpairs

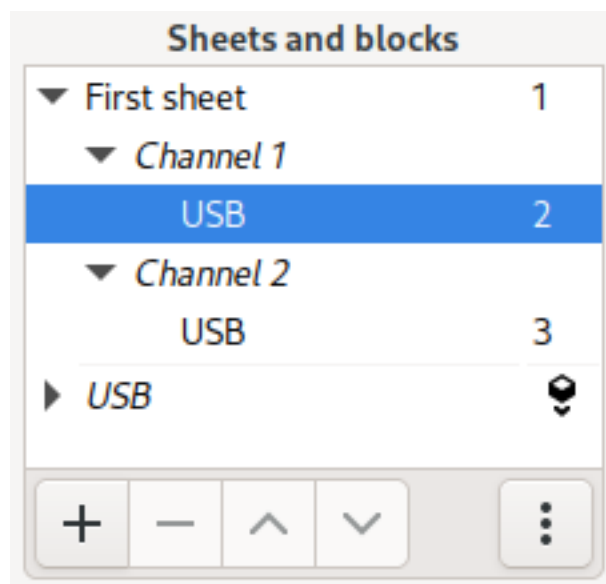
To create a differential pair, select the two nets you want to become a pair and run the “Set diff. pair” tool. You can also select one net and you’ll be asked for the other net. To decouple the nets, use the “Clear diff. pair” tool. It’s recommended to assign both nets a netclass such as “100diff” so you can match them in the rules.

## 21.6 To board

To facilitate placing packages on the board, simply select the corresponding symbols and activate the “place on board” action by pressing **p b**. This will switch to the board editor and launch the “place package” tool with the packages for the selected symbols. Note that you may need to reload the netlist in the board editor before doing so to make the board editor pick up new components.

## 21.7 Hierarchy

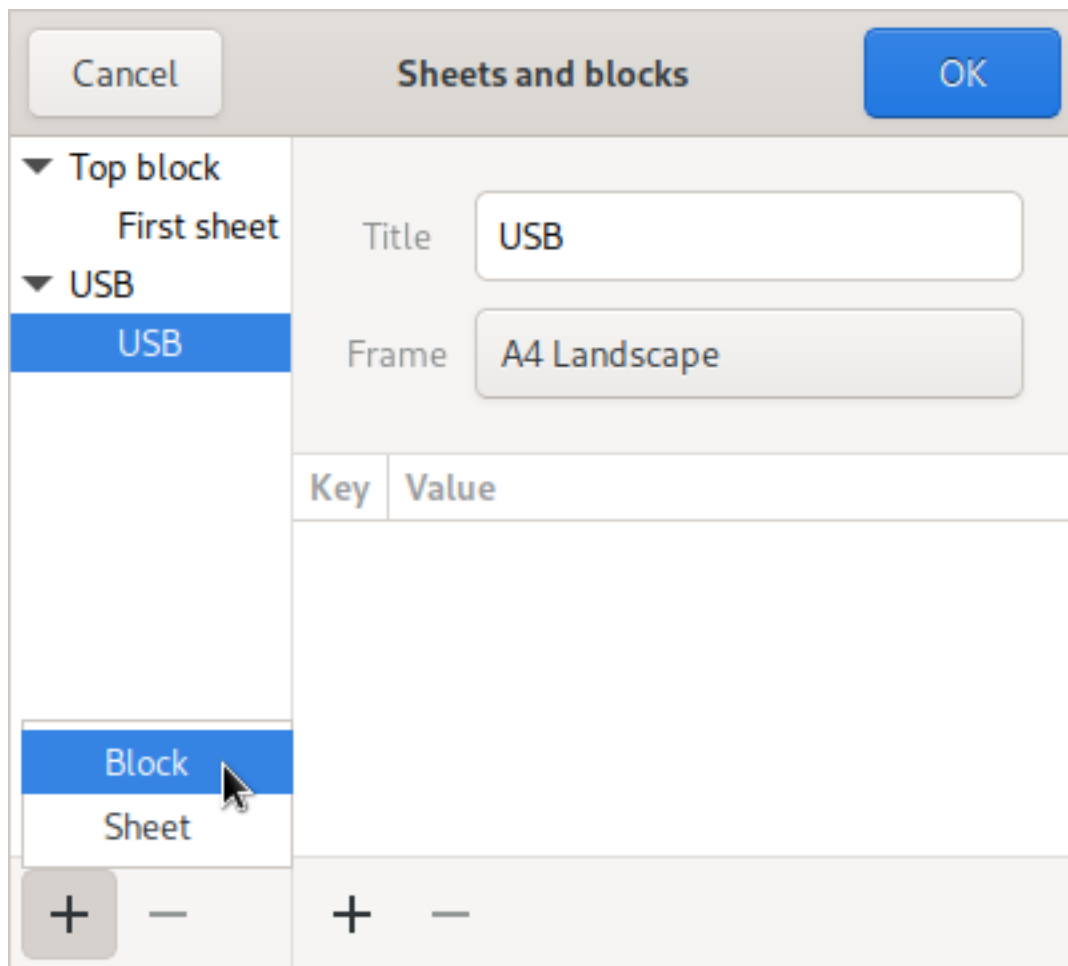
Similar to other schematic entry tools, hierarchical schematics are made up of blocks that can be instantiated. To create a new block, open the sheets and blocks dialog by clicking on the three dots in the sheet list.



In that dialog, click on the plus icon in the block list to create a new one.

All blocks, regardless of whether they’re instantiated or not, appear below the sheets in the sheet list. Select a block to edit its symbol. Clicking on the insert icon next to a block instantiates it on the current sheet.

Connectivity to the rest of the schematic can be established either through ports or power nets. Power nets are global to the entire schematic. Since it’s good practice to name ports identical to nets, ports aren’t separate objects in Horizon



EDA. Instead, a port is defined by setting the “is port” property on an existing net. This can be done either by toggling the appropriate switch in the properties of a net in the right sidebar or through the port nets dialog.

Blocks can be edited either inside or outside of the hierarchy. To edit a block within the hierarchy, double-click any of its placed block symbols or select one of its sheets in the top part of the sheet and block list. Changes to reference designators or “do not populate” flags are specific to the selected instance.

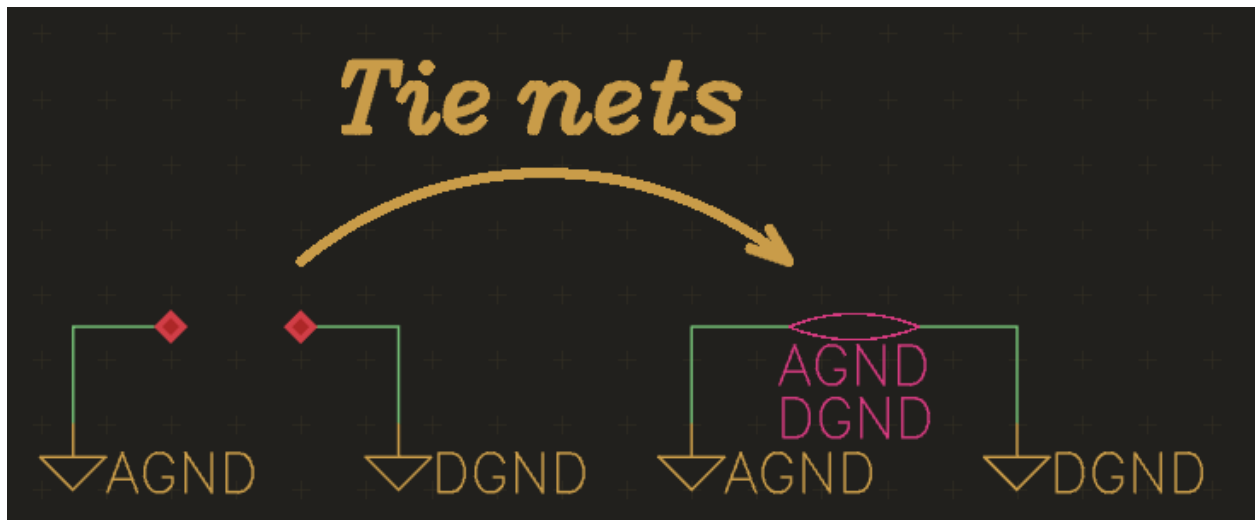
When selecting a sheet of a block directly from the list of blocks in the lower part of the sheet and block list, reference designators and “do not populate” can’t be modified as they’re instance-specific. Aside from that, there’s no difference between editing a block inside or outside of the hierarchy. The current mode is shown in the bar that sits above the schematic viewport.

Deleting a block won’t delete its schematic and block from disk. Right now, there isn’t any way to do so, so you’ll manually need to delete the block from the *blocks* directory in the project if you want it to be gone.

## 21.8 Net ties

Use net ties to electrically connect two nets. One use case could be connecting analog and digital ground nets.

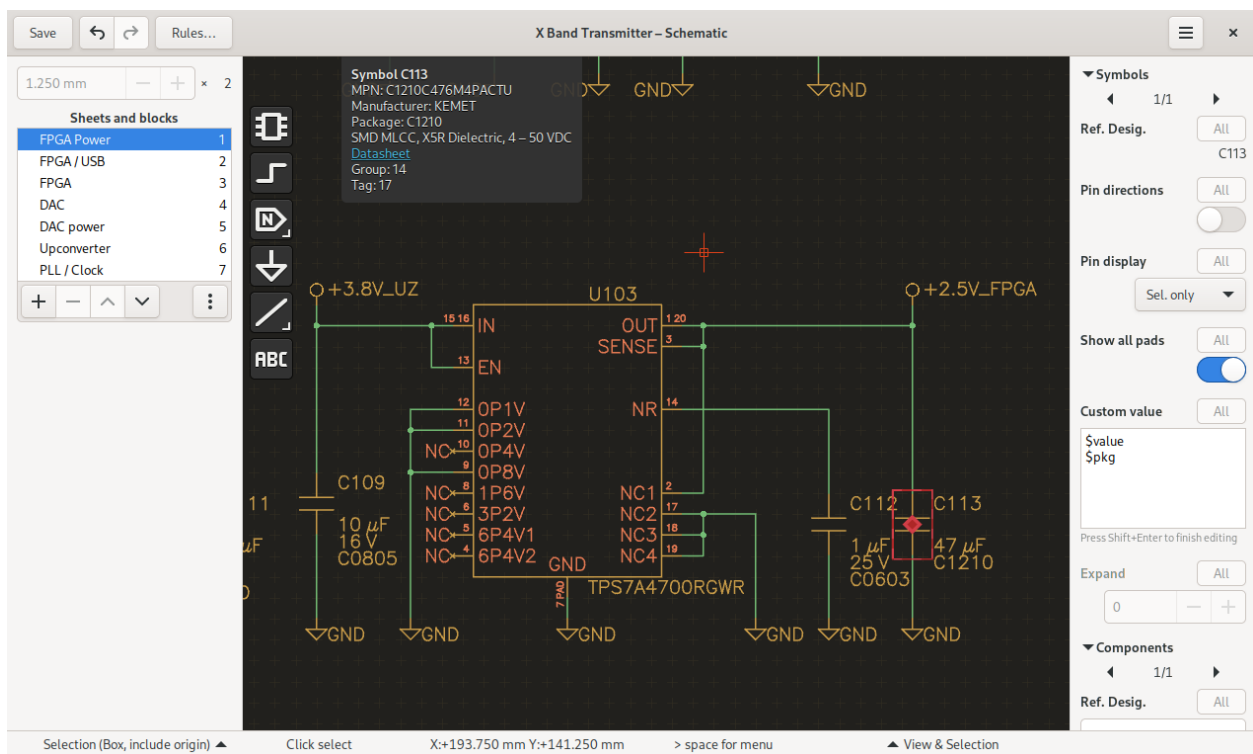
To create a net tie, select two junctions, one of each net, that should be tied and invoke the “tie nets” tool:



The net that’s listed first (AGND in this case) is the primary net of a net tie. See [Board Net ties](#) for what that means and how to represent net ties on the board. The “flip net tie” tool swaps primary and secondary net of a net tie.

## 21.9 Screenshots







## BOARD EDITOR

To launch the schematic editor click on “Board” in the project manager.

The board editor keeps an internal copy of the netlist. To update the netlist, click “Save” in the schematic editor for writing the netlist to disk, then click on ‘reload netlist’ or re-open the board editor or use the action “Save and reload netlist” in the the schematic editor.

### 22.1 Planes

For adding planes, first draw a polygon of the desired shape in a copper layer. Then use the “Add plane” tool to assign it a plane. Planes with lower fill order will get filled first. To override the connection style (solid or thermal relief), create a Thermal rule.

### 22.2 Vias

To place a via while routing, press v. The “Layer pairs” rule defines which layer the router switches to. To change the size of vias, define a matching via rule.

You can also place vias not connected to any track with the “Place via” tool.

### 22.3 Importing logos

The preferred way to import logos and other artwork into the board editor by means of the DXF import tool. Since Horizon EDA doesn’t support bezier curves, convert these to lines first. Inkscape’s “Modify Path/Flatten Beziers” extension works great for this. If you require filled polygons, use the “Line loop to polygon tool” to convert line loops to polygons. Use the “Scale” tool to adjust the imported logo to the size you need.

### 22.4 Board outline

Define the board outline by drawing polygons on the Outline layer. The board must be representable by a single polygon with zero or more holes, that means:

- There must one polygon that encloses all others
- These other polygons are holes
- Hole polygons must not intersect or touch each other

The “Outline” rule checks that these requirements are met. Do not use arcs or lines on the outline layer. Put any other outline-related information such as indications for v-scores on the “Outline Notes” layer. The Gerber export merges the “Outline” and “Outline Notes” layers.

## 22.5 Reconnecting tracks without a net

For a track to be assigned a net, it has to be connected to a pad or a via either directly or indirectly. If that’s not the case, tracks are assigned to no net and turn orange. Use the “draw track” tool to reconnect the track to a net.

## 22.6 Diffpairs

To create a diffpair see [Schematic Diffpairs](#). Before routing a diffpair, create a diffpair rule specifying track width and gap. To route a diffpair, use the “Route diff. pair” tool.

## 22.7 Nets window

You can open the nets window from the “View & Selection” menu in the bottom bar or by searching for it in the spacebar menu.

### 22.7.1 Hide Airwires

Uncheck the checkbox in the airwires column to hide the airwires of a net. You can also select more than one net and use the context menu to turn off or on airwires for multiple nets at once.

Nets		
<div> <input type="text" value="PLL"/> <input type="button" value="x"/> </div>		
<div> <input type="button" value="All on"/> <input type="button" value="All off"/> <input type="checkbox"/> Airwires only         </div>		
Net	Net class	Airwires
+3.3V_PLL	default	<input checked="" type="checkbox"/> 0
+3.3V_PLL_F	default	<input checked="" type="checkbox"/> 0
+3.3V_PLL_OUT	default	<input checked="" type="checkbox"/> 0
PLL_CE	default	<input checked="" type="checkbox"/> 1
PLL_CS	default	<input checked="" type="checkbox"/> 0
PLL_MOSI	default	<input checked="" type="checkbox"/> 0
PLL_MUXOUT	default	<input checked="" type="checkbox"/> 0
PLL_REFCLK+	100diff	<input checked="" type="checkbox"/> 0
PLL_REFCLK-	100diff	<input checked="" type="checkbox"/> 0
PLL_SCK	default	<input checked="" type="checkbox"/> 0

### 22.7.2 Recolor nets

Right click on one more selected nets to open the context and assign a specific color to nets.

## 22.8 Panelisation

For making best use of the available board space, Horizon EDA supports panelisation. Rather than copy/pasting a board into another one, which complicates last-minute changes, a board can reference other boards.

Panelising an existing board in Horizon EDA is a simple multi-step process:

1. Create a new empty project and open the board.
2. Use the “Manage included boards” tool to load the board(s) to be placed on the panel.
3. Place the included boards using the “Place board panel” tool.

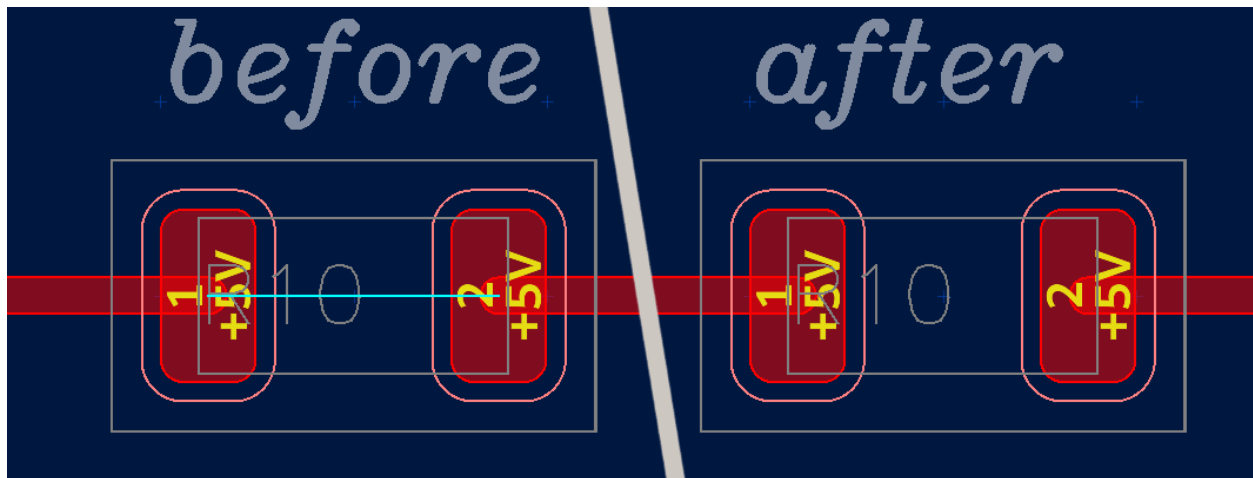
The boards can then be arranged as one object each without the possibility of unintentionally modifying it. To update the included boards, either use the reload button in the “Manage included boards” tool or reopen the project.

For making it easier to draw the panel outline, the outline of included boards can be extracted using the “Smash panel outline” tool.

### 22.9 Shorted pads

When using zero-ohm resistors or other components for connectivity, there’s still an airwire across the component since Horizon EDA isn’t aware that the part is a short circuit.

Create a “Shorted pads” rule to specify that the pads of a part are electrically connected.



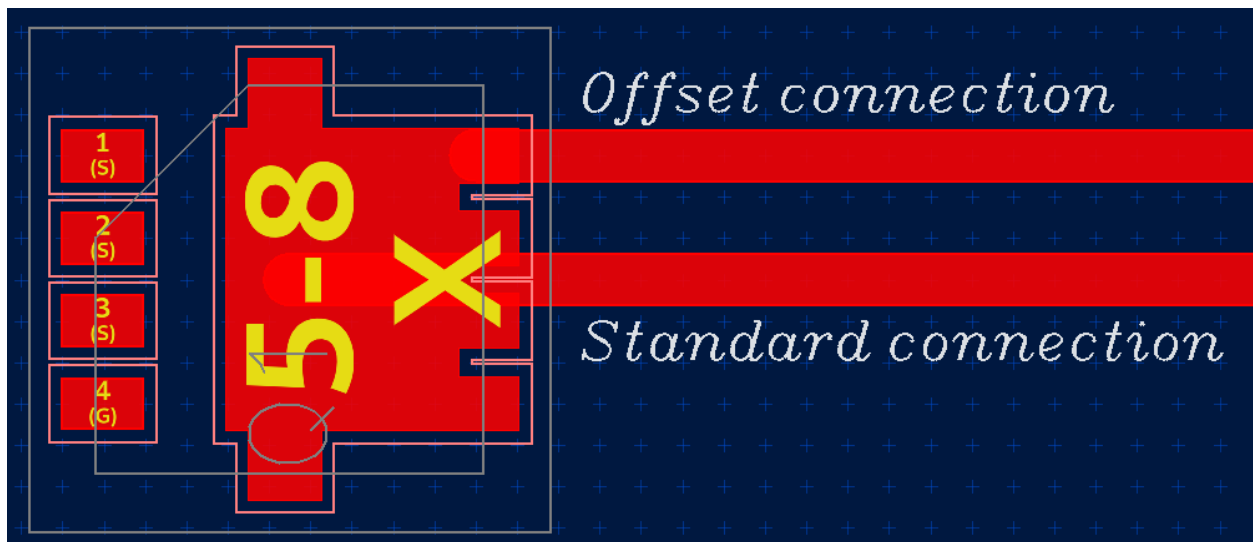
## 22.10 Net ties

Once a net tie has been defined on the *Schematic*, it can be added to the board using the “Draw net tie tool”. A net tie on the board behaves similar to a track of the primary net and are treated as such by track width, copper clearance and other rules. Since net ties go from junction to junction, they can be joined with other tracks.

The “Net ties” rule checks that all net ties that are defined on the schematic are drawn on the board and connect to the correct nets. The copper clearance checks ignore the clearance violation caused by the net tie, but still flag clearance violations caused by other objects of the net.

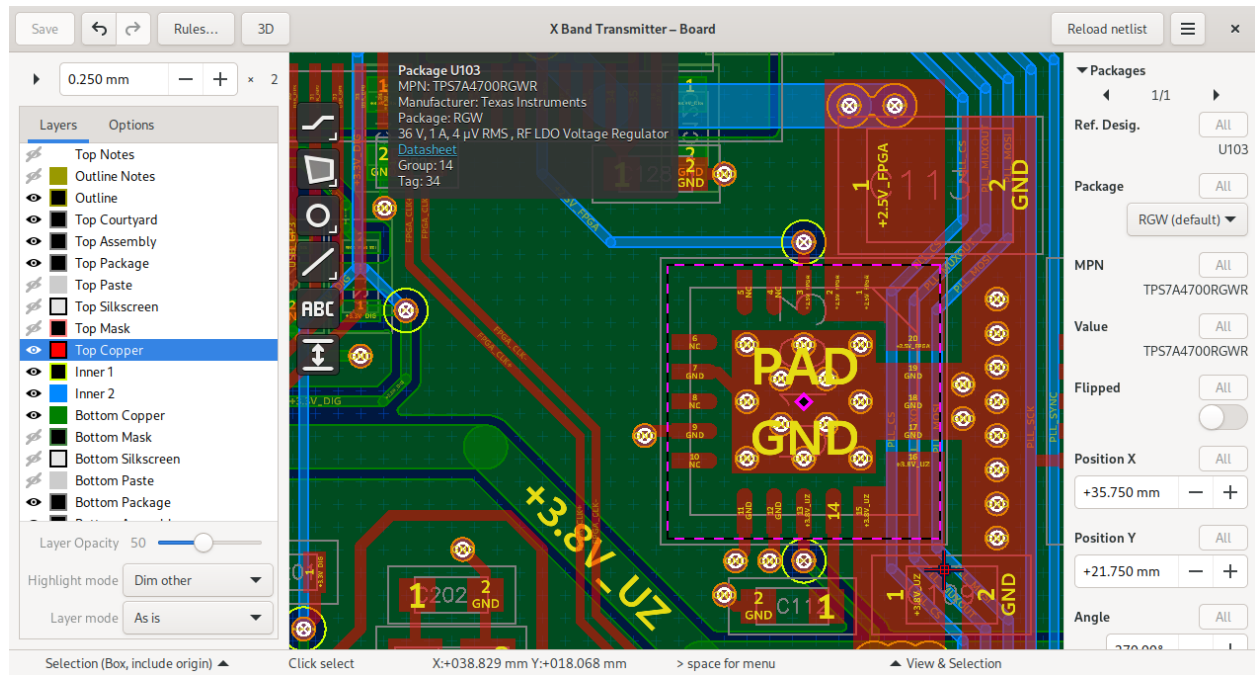
## 22.11 Offset pad connection

For complex-shaped pads, one might want to connect a track to a specific position in the pad rather than to its origin. Use the “Move track connection” tool to move the end of a track that’s connected to a pad.



Keep in mind that tools based on the KiCad router such as “Route track” or “Drag track” aren’t aware of this and will break the pad-track connection.

## 22.12 Screenshot







## COPYING LAYOUT AND PLACEMENT

### 23.1 Motivation

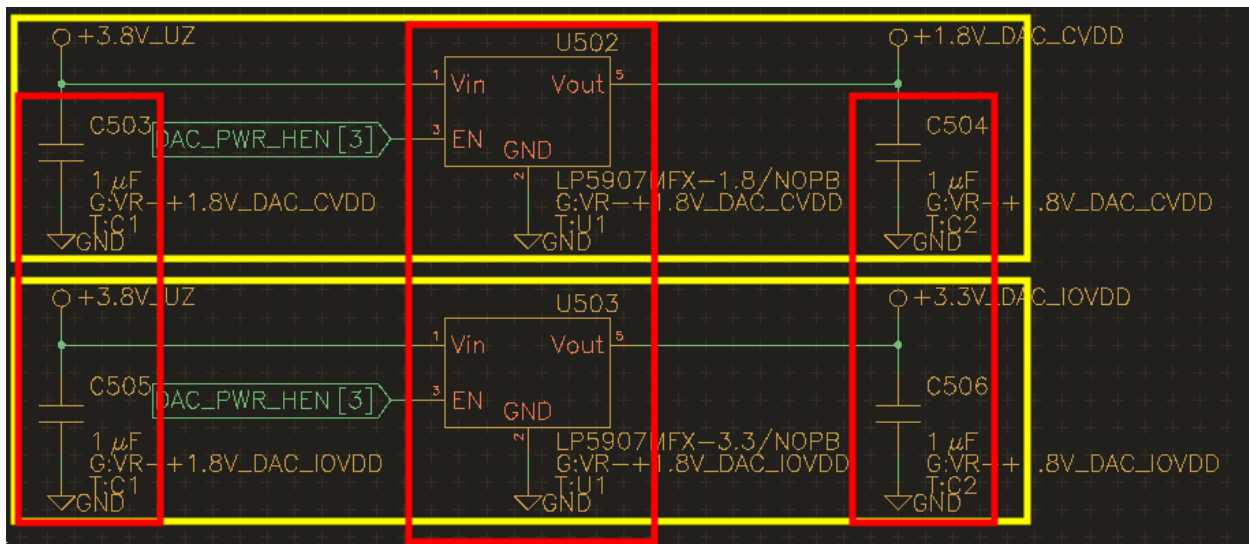
Often times, your design includes similar-but-not-identical sections such as voltage regulators. Wouldn't it be nice if you'd only had to do layout and placement once and then copy it to the other instances? Horizon EDA lets you do just that in a simple two-step process.

### 23.2 Groups & Tags

For this feature to work, you'll first need to tell horizon EDA how the components fit together. This is accomplished by assigning groups and tags to components. Each section, i.e. all components associated with one voltage regulator, get assigned one group. To do so, select all symbols of one section and use the tool "Set new group" to assign all of them to a new group. For making groups and tags visible on the schematic, use the "Toggle group & tag visibility". A components group and tag will then show up below the reference designator.

To tell horizon EDA the matching components in each group, these get assigned identical tags. Since a newly-placed component will already be assigned a unique tag and groups and tags get preserved on copy/paste other instances of the same circuit will likely have the appropriate tags already set. To change the tag on a component, use the "Set tag" tool.

When you're done, the schematic should roughly look like this (with the boxes added for clarification). All components inside a yellow box belong to the same group, all inside a red box belong to the same tag.



You may use the “Highlight group/tag” action to make sure that you got the assignments right.

## 23.3 Board

Place and route any group as usual.

### 23.3.1 Copy placement

For each group, place the package you’d like the other packages to be referenced to at the desired position and place all other packages anywhere. Then, select all packages of the group that you’d like to apply the placement to and start the “Copy placement” tool. Click on the reference package (any pad or centroid) in the already placed group and all selected packages will be placed accordingly.

### 23.3.2 Copy tracks

Select all tracks (other objects will be ignored) you want to copy in the routed group, start the “Copy tracks” tool and click on any package (any pad or centroid) in the destination group.

## BACKANNOTATING CONNECTIONS

Sometimes, you may want to connect component pins based on their location on the board, such as connectors or FPGA IO pins. Wouldn't it be nice if you could define these connections directly in the board editor without going back and forth between board and schematic for every connection? With horizon EDA, you can!

### 24.1 How To

Use the tool "Draw connection line" to connect pads or junctions as desired. Then, use "Backannotate connection lines" to send these connections over to the schematic editor. The newly created connections will appear as net stubs. After saving the schematic and reloading the netlist in the board editor, the connection lines will automatically be replaced by airwires.

### 24.2 Limitations

Since this feature hasn't been there for very long, some things are still unsupported:

- Can't connect two existing nets
- Chaining connection lines (connecting two lines to one pad/junction) will not yield the expected result



## **RULES**

Horizon uses rules for specifying constraints for DRC as well as input for various tools such as the interactive router. Rules are evaluated top to bottom and the first rule matching all criteria will be applied. So it's up to you to make sure that rules are ordered from more specific to less specific. It's always a good idea to have a catchall rule at the very bottom.



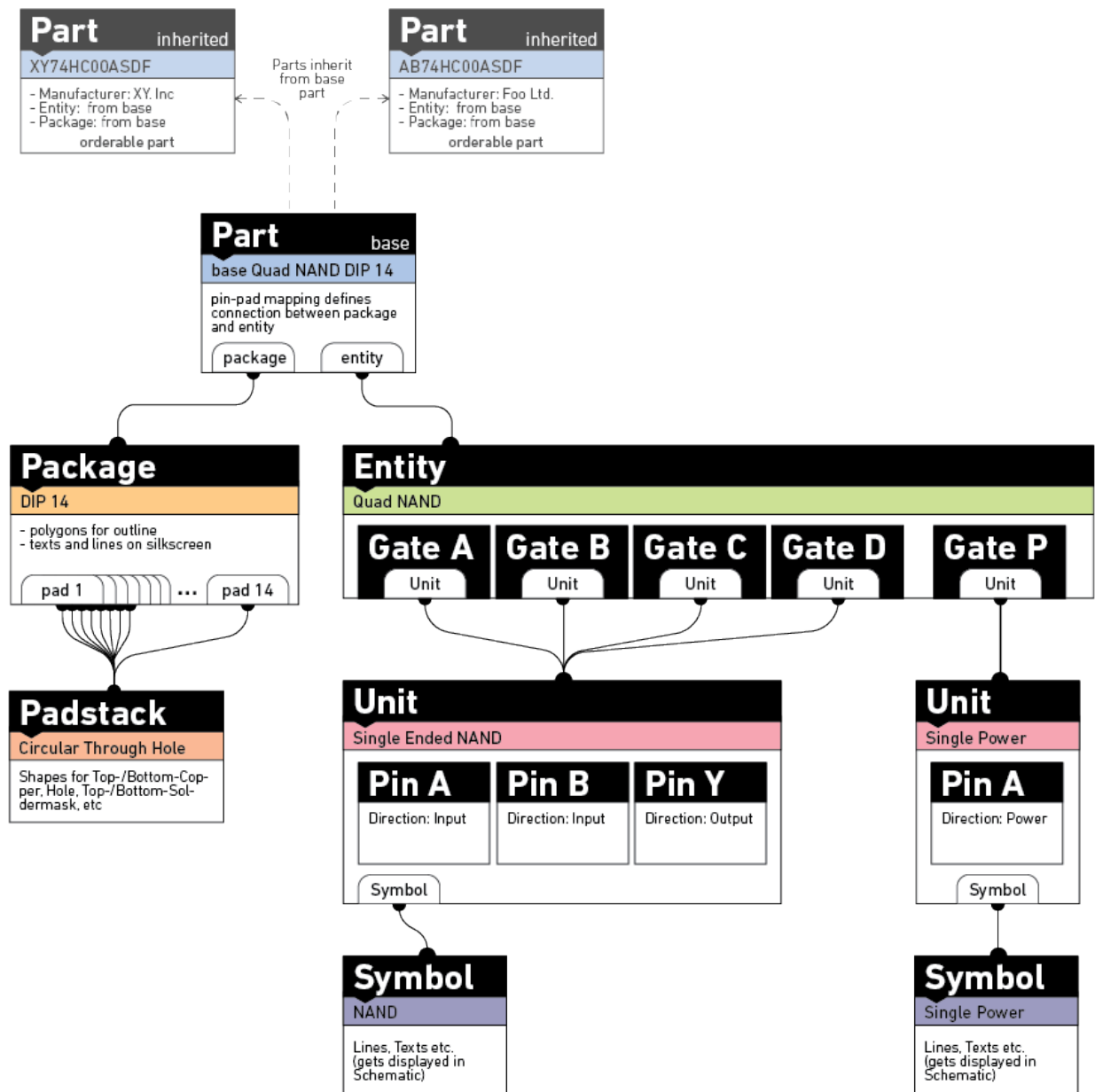
## WHY A POOL?

### 26.1 What's a pool

So what's all this Pool stuff anyhow? Many EDA packages organize packages, symbols and the like in libraries. These are often messy and version-controlling these is difficult since many independent parts are put in a single file. Especially the latter often makes collaboration difficult.

With horizon, there are no libraries. Instead all the non-project elements (symbols, etc.) are stored in a pool. Similar to the “central library” approach common among the more enterprisey EDA packages.

However the new thing here is, that a Part within this pool is *composed* of multiple other elements, that handle different aspects of the Parts nature:



For example you can define one “Quad NAND” Entity and reuse it for each new NAND Part, without having to redefine the Gates time and time again. The Quad NAND Entity in turn is composed of multiple Single Opamp NAND Units and one Power Unit. If you now want to make a Dual NAND Entity, you can just reuse the already existing Units and this guarantees you consistency with the other NAND parts in your pool.



## 26.2 Organisation in the file system

Each of these elements that make up a Part is stored in a single json file in the respective directory, i.e. /symbols, /entities, /units, /parts, etc. The exact location within these directories is irrelevant, as long the json file is stored in the correct directory: Symbols in /symbols, Units in /units and so on. Additionally it is important for the files to end in “.json” so they can be picked up by the pool updater. To make searching for parts more convenient, the metadata of all json files is aggregated into a sqlite database. This is what the ‘Update pool’ button in the Pool Manager is for.

Naturally a pool with a focus on composition is organized using tags instead of a hierarchical system since these often lead to (unnecessary) confusion over aspects like whether to group parts by manufacturer or other attributes.

## 26.3 Including other pools

Apart from containing items, a pool can also include other pools to get access to their items. To include a pool, first add it in the “Pools” window available from the menu on the top right in the pool/project manager and then move it to the “Pools included” list in the “Settings” tab in the pool manager. The order of included pools becomes relevant when an item of identical type and UUID exists in multiple included pools. Items from pools that are higher up in the list take precedence over items from pools that are lower down in the list.

When a pool includes at least one pool, pool browsers show a colored box in front of items to indicate their source. Check the tooltip for the meaning of each color.

## 26.4 Contributing

Although you can create your own pool, you are strongly encouraged to use the pool over at <https://github.com/horizon-eda/horizon-pool/>. To add new parts to it, simply submit a merge request. See also: *Contribute to the Pool*



## ELEMENTS OF A PART

A final Part is composed of multiple elements, that are stored independently from each other in the Pool.

### 27.1 Parts

On top of pool's structure there is the part. To avoid redundancy and allow faster changes, a part can inherit its definition (or parts of it) from another part. This is intended to be used for groups of parts that only differ in some property like resistance or output voltage for fixed voltage regulators. Each part can be accompanied with parametric data to make it easier to search for. Right now, this feature is only implemented for resistors and capacitors.

In addition to its data the part is connected to both a Entity and a Package. The Part stores how the pins of each Gate in the Entity map to the pads in the Package (e.g. in the case of the NAND it would map Input A to the corresponding Pin on a DIP 14 package).

### 27.2 Packages

A Package defines the footprint of a part. If the part's manufacturer provides a reasonable footprint recommendation, use this one. Only use generic packages if there isn't any. For details on packages see [Creating a Package](#).

### 27.3 Entities

An Entity is a Part's netlist representation and consists of one or more Units. Parts that are logically the same like different shapes of USB connectors therefore can all share the same Entity, e.g. "USB connector with shield and ID". The Symbol for each Unit in the Entity can be placed independently on the schematic.

### 27.4 Units

A Unit actually defines a part's logical pins. For parts that only consist of one "gate" like a simple resistor, their entity simply references one unit. For parts consisting of multiple "gates" like a dual operational amplifier or a big MCU, each gate references one unit. Having units separate from entities allows multiple entities to share the same units. The entity for a dual logic gate thus is supposed to reference the same unit as a quad one. Apart from a name, a pin has a direction (for ERC) and optionally alternate pin names to deal with pins having multiple functions as it's common among MCUs.

## 27.5 Symbols

A symbol is used in the schematic to represent a unit. Contrary to other EDA applications, a symbol just displays the pins from its unit and doesn't define these.

## 27.6 Padstacks

In horizon a pad references a padstack to determine its shape. There are two kinds of padstacks in horizon: Global padstacks are available for use to all packages and should cover most uses cases. In case a package requires a specialized padstack, you can create one using the “Create padstack for package” button in the “Packages” tab in the pool manager.

A padstack consists of these items:

- Copper
- Solder mask opening
- Paste mask
- Hole (optional)

The preferred way of defining geometry in padstacks is to use shapes as these get translated to efficient primitives when exporting the board as a gerber file. To suit more than one package parameters can be applied to padstacks that alter their size. Parameters are also used to apply application-specific global parameters like solder mask expansion and paste mask contraction. To get an idea on how this works out in reality take a look at the global padstacks in the pool.

To apply the parameters to the padstack's geometry each padstack is accompanied by a *Parameter Program* that takes care of applying the parameters to the shapes.

## **PROJECT POOL**

Since version 2.0, each project is accompanied by a project pool. When using a part or any other pool item for the first time, it gets copied into the project pool along with its dependencies and is kept there until explicitly updated. Same as regular pools, project pools can include other pools. In fact, that is the way to use parts from pools in the first place.

To get an overview of items that have been copied into the project pool, go to the “Cache” tab in the project pool manager. If an item has been modified in an upstream pool, select it in the list and click on “Update from pool”. After that, reload the pool in the board/schematic editor or reopen them to get the updated items.

Over time, project pools may accumulate items that are no longer needed in the project. To get rid of them, click on “Remove unused” in the “Cache” tab in the project pool manager.



## POOL MANAGER

The Pool Manager and Part Wizard help with managing things in the Pool like symbols, entities and parts. You're most likely to use the Pool Manager for creating new parts. To open the pool manager, launch Horizon EDA and click on the recently opened pool you like to edit. If you never opened a pool before, you can manually select a pool.json you like to open or create a new one. Depending on what kind of part you want to create, several workflows are available:

### 29.1 Inheriting from an existing part

When the part you're about to create already exists in a different variant (different value or different temperature range) but is otherwise all identical, the new part should inherit from the existing part. To do so, select the desired base part in the "Parts" tab and click "Create Part from Part". After having specified the new part's location, you'll be shown the Part Editor. Uncheck the "inherit" option for the attributes you'd like to change and save the new part.

### 29.2 Create part from existing Entity

This workflow is appropriate when the Entity for the new part already exists. Resistors or LEDs in nonstandard packages are of this kind, for example. A Part is made up from a Entity and a Package, if the Package for the Part you want to make doesn't exist you will have to create it (see [Creating a Package](#)).

If you have a fitting Entity and Package you can go ahead and create a Part. In the "Parts" tab, click on "Create Part" for creating the new part. Then after specifying both Entity and Package and the Part's location, the Part Editor opens and you can edit the part and map the Entity pins to the Package pads.

### 29.3 Create all-new part

Many parts such as MCUs, FPGAs, ADCs and other miracles of today's world require creating new units and entities. Doing so manually would be very tedious, that's why there's the Part Wizard to assist you. After having selected the part's package (for creating packages, see [Creating a Package](#)) in the "Packages" tab, click on "Part Wizard..." to launch it. You'll be greeted with a list of all Pads of the package.

Fill in the pin names according to the datasheet. Only put the pins primary name (like PB5) on an MCU in the leftmost entry and put all other names (like UART0\_TX, TA0) space-separated in the "Alt. names" Entry. If your part is *really* big (such as an FPGA or large MCU) you may want it to appear as more than one symbol in the schematic. To do so, select all the pads you'd like to have in the same symbol and type in the new Gate name. In case many Pads are electrically identical (such as many GND pads) you may group them by selecting them and clicking the "Link Pads" button in the bottom toolbar. This way, only one Pin will be generated for these Pads.

For really large parts with 100+ pins, putting them all in manually can become too tedious. To get around is, the part wizard can import pin names from a CSV or json file.

A CSV file for pin import can look like this:

```
1,PB0,bidirectional,Main,TXD,SDA
2,PB1,bidirectional,Main,RXD,SCL
3,TDI,input,Main
4,TDO,output,Main
5,GND,power_input,Main
```

The full CSV format is `pad,pin,direction,gate,alt1,alt2,...`

Defining the same pins in json format can be done with:

```
{
  "1": {"pin": "PB0", "alt": ["TXD", "SDA"], "gate": "Main"},
  "2": {"pin": "PB1", "alt": ["RXD", "SCL"], "gate": "Main"},
  "3": {"pin": "TDI", "direction": "input", "gate": "Main"},
  "4": {"pin": "TDO", "direction": "output", "gate": "Main"},
  "5": {"pin": "GND", "direction": "power_input", "gate": "Main"}
}
```

For both CSV and json, entries with the same `pin-gate` will get their pads merged. Additional valid values for `direction` are `open_collector`, `passive` and `not_connected`.

Reasonable sources for pin/pad name data are:

- IBIS models
- BSDL files
- PDF datasheets

Once you're done filling in the pin names, click "Next" in the top left corner for advancing to the next screen. Fill in the entries according to your part. In case you're unsure on what to put in, take a look at existing parts in the pool. If your part is available in multiple almost-identical variants that only differ in aspects such as temperature range or packing option (Tape/Reel, Tube, etc.) create the part you're about to use. For creating the other variants, follow the instructions in the topmost section. Take care of specifying the correct location for units/symbols/entity and parts, so that they end in a subdirectory of their respective directory in the pool.

For each gate, click on "Edit Symbol" for launching an interactive manipulator to create the symbol for that unit. Use the "Map pin" tool to place the pins in the symbol and "Draw line rectangle"/"Edit line rectangle" for drawing the symbols body. Don't forget to give the symbol a meaningful name and place the "\$REFDES" and "\$VALUE" texts.

When you've drawn all the symbols and filled in all of the metadata, click "Finish" to finally insert the part into the pool.

## 29.4 Where to save things

When creating new symbols, parts and the like the pool manager/part wizard will sooner or later ask you for a file name or a directory (in case of packages) to save the new part. Technically, the path you specify only needs to meet two requirements:

- It needs to be in the correct top-level directory, i.e. every symbol has to be somewhere in `/symbols` and so on. Padstacks specific to a package must be placed in the package's padstack directory.
- The file has to end in `.json`

To get an idea of how all this looks in practice, take a look at [the pool](#)



## 29.5 The Pool database

The Pool keeps the metadata (filenames, UUIDs, names, etc.) in a SQLite database to facilitate searching. Normally, the pool manager updates the database every time a file in the pool is saved. However, if you externally manipulate/remove files, you'll have to click "Update Pool" for the database to include the changes you made.



## CREATING A PACKAGE

A PCB layout can only be as good as the footprints it uses, that's why it's important to create high-quality footprints (called packages).

As far as horizon is concerned a package consists of these things:

- The pads the part gets soldered onto
  - Copper layers (top/bottom/inner)
  - Holes (for TH parts)
  - Solder mask opening
  - Paste mask
- Package outline
- Assembly outline and reference designator
- Silkscreen graphics and text
- Courtyard outline

### 30.1 Pads

Each pad is defined by its padstack and parameters applied to the padstack. For details on padstacks see [here](#). Since the pads are probably a package's most important feature, it's makes sense to start with these. You can either place pads manually using the "Place pad" tool or have them placed automatically according to commonly used patterns using the "Footprint generator" dialog available from the magic wand button. After having placed the pads, they still have their default size, which is very likely not what you want. To fix this, select the pads you'd like to modify and use the "Edit pad" tool to add parameters to a pad. Depending on the selected padstack, certain parameters are understood. The most commonly used are obviously pad width and height. Use the checkmark button next to a parameter to apply it to all selected pads.

## 30.2 Package outline

The package outline is used for visualizing what the part's outline looks like, hence it should follow the part's nominal dimensions. You may use the “import DXF” tool for importing a DXF drawing obtained from a STEP model or otherwise. Since the package outline's purpose is purely visual, you can use either lines or polygons. Only include pins if they significantly contribute to the part's appearance.

## 30.3 Assembly outline

The assembly outline ends up on the assembly drawing (not yet implemented) and is intended to aid assembling and inspecting the PCA. The assembly outline layer thus contains only these items: A polygon indicating the part's outline, optionally pins if they significantly contribute to the part's appearance and the part's reference designator. Opposed to the package outline, the assembly is just a rough approximation of the part's shape. It has to include some sort of visual indication of the part's pin 1 location. Use the “Draw polygon rectangle” tool and its decoration options for drawing such outline. For the reference designator, place a text containing “\$RD” with such size, that it fits withing the assembly outline even when being prefix + 4 digits long.

## 30.4 Silkscreen

The silkscreen graphics purpose is to clarify a parts location and orientation during manual assembly and visual inspection, so it should include some sort of pin 1 marker if the part is orientation-sensitive. Don't place a dot at pin 1, instead shorten/elongate the silkscreen graphics. The recommended line with is 0.15mm. Also place a text “\$RD”, 0.15mm in with on the silkscreen layer.

## 30.5 Courtyard

The courtyard denotes the space needed by the part that mustn't be occupied by other parts in order to leave enough room for assembly. Since the size of the courtyard needs to be adjusted depending on the users manufacturing requirements, it has to be set using a parameter program. At 0mm courtyard expansion the courtyard outline is the (rectangular) hull around copper pads and package outline. To create a rectangular courtyard outline that can be parametrized, do this:

Use the “Generate courtyard” tool to generate the initial courtyard at 0mm expansion. If that doesn't result in the desired polygon, use the “Draw polygon rectangle” tool for drawing the initial courtyard and set its parameter class to “courtyard” using the property editor on the right side of the window.

Open the “Parameters” Window and click on “Insert courtyard program”. If all goes well, this should add the courtyard program and as the parameter “Courtyard expansion” set to 0.25mm.

## CONTRIBUTE TO THE POOL

The official pool at <https://github.com/horizon-eda/horizon-pool/> lives from it's user contributions. There are multiple ways you can help. The most obvious one is by submitting parts you made.

To keep the pool nice an clean, only add parts you can actually buy with their corresponding symbols, entities, etc. So don't add some part called 7805, instead add a MC7805BDTRKG manufactured by ON Semiconductor. Once you created something you'd like to share, you can use the *Pool Manager* to upload your creation to the offical pool:

### 31.1 Using the GitHub integration.

For the GitHub integration to work, the pool has to be downloaded using the “Download...” button on the start Page of the pool manager. The pool manager will clone the global pool into the `.remote` directory in your local pool. If all goes right, you should never need to touch that directory. Two operations are available for keeping your local copy up-to-date and merging your parts into the global pool

#### 31.1.1 Upgrade pool

This will update your copy of the global pool in the `.remote` directory to the latest commit and ask you which changes you'd like to have applied to your local pool.

#### 31.1.2 Create pull request

First, add the Parts/Entities/etc. to the “items to be merged” list, then fill in Pull Request title and body. The pool manager will automatically add items that are needed to not break references. So if you create an all-new Part with new Unit, Entity and Package, these will get added to the list when you add the Part. Don't forget to add the new symbols. After making sure that this is what you want, click the “Create pull request” button. You'll be prompted for your GitHub credentials as well as your name and email address for the commit author information.

#### 31.1.3 Helping by reviewing

Adding parts is a great thing, but checking parts other people made could be a good thing as well. More eyes that crosscheck a part against its datasheet will decrease the chance of something that works. If you successfully produced a PCB with certain parts on it, you can say something about solderability as well and this is a stronger indicator, that the part has no critical mistakes.



## BUILDING ON WINDOWS

### 32.1 Install MSYS2

Download and run the msys2 installer from <http://msys2.github.io/> I've only tested with the 64bit version, 32bit should work as well (but come on, it's 2017...) Make sure that the path you select for installation doesn't contain any spaces. (Don't blame me for that one)

### 32.2 Start MSYS console

Launch the Start Menu item "MSYS2 mingw 64 bit" you should be greeted with a console window. All steps below refer to what you should type into that window.

### 32.3 Install updates

Type

```
pacman -Syu
```

if it tells you to close restart msys, close the console window and start it again. Then run `pacman -Syu` again.

### 32.4 Install dependencies

Type/paste

```
pacman -S mingw-w64-x86_64-gtkmm3 git base-devel \  
mingw-w64-x86_64-boost mingw-w64-x86_64-curl \  
mingw-w64-x86_64-sqlite3 mingw-w64-x86_64-toolchain \  
mingw-w64-x86_64-zeromq mingw-w64-x86_64-glm zip \  
mingw-w64-x86_64-libgit2 mingw-w64-x86_64-ocaml \  
mingw-w64-x86_64-podofu mingw-w64-x86_64-libarchive --needed
```

When prompted, just hit return. Sit back and wait for it to install what's almost a complete linux environment.

Before continuing you may change to another directory. It easiest to type `cd` followed by a space and drop the folder you want to change to on the window.

## 32.5 Clone horizon

```
git clone http://github.com/horizon-eda/horizon
cd horizon
```

## 32.6 Build it

```
make -j 4
```

You may adjust the number to the number of CPUs in your system to speed up compilation. Expect 100% CPU load for several minutes. Due to debug symbols the resulting executables are of considerable size.

## 32.7 Running

You won't be able to double-click the resulting executables since all the required DLLs are in a directory unknown to windows. You'll have to launch them from the mingw shell using `./horizon-eda` for example. For the pool download to work, you'll have to copy the file `/mingw64/ssl/certs/ca-bundle.crt` to the directory the directory `horizon-eda.exe` resides in.

## 32.8 Packaging

To create the zip archive as it's available from the CI, run `./make_bindist.sh`.



## BUILDING ON LINUX

Building horizon on Linux is as simple as `make` after you've cloned this repo.

### 33.1 Install dependencies

Make sure you got these dependencies installed:

- Gtkmm3 3.20
- cairomm-pdf
- librsvg
- util-linux
- sqlite
- boost
- zeromq
- glm
- libgit2
- curl
- opencascade / opencascade community edition
- zeromq with C++ bindings: <https://github.com/zeromq/cppzmq>
- podofo
- libarchive
- libspnav

The C++ compiler needs to support `std::filesystem`, for GCC this requires version 8 or newer.

On Ubuntu 18.04 run:

```
sudo apt install libsqlite3-dev util-linux librsvg2-dev \  
libcairomm-1.0-dev libepoxy-dev libgtkmm-3.0-dev uuid-dev libboost-dev \  
libzmq5 libzmq3-dev libglm-dev libgit2-dev libcurl4-gnutls-dev liboce-ocaf-dev \  
libpodofo-dev libarchive-dev libspnav-dev
```

On Arch Linux:

```
sudo pacman -S zeromq gtkmm3 cairomm librsvg sqlite3 libgit2 curl \
  opencascade boost glm podofo libarchive libspnav
```

On Fedora 25/26/27:

```
sudo dnf install git make gcc gcc-c++ pkg-config cppzmq-devel OCE-devel\
  gtkmm30-devel libgit2-devel libuuid-devel sqlite-devel librsvg2-devel\
  cairomm-devel glm-devel boost-devel libcurl-devel podofo-devel libarchive-devel\
  libspnav-devel
```

On openSUSE Tumbleweed:

```
sudo zypper in git make gcc gcc-c++ pkg-config cppzmq-devel oce-devel\
  gtkmm3-devel libgit2-devel libuuid-devel sqlite3-devel librsvg-devel\
  cairomm-devel glm-devel boost-devel libcurl-devel libpodofo-devel binutils-gold\
  ↪libarchive-devel\
  libspnav-devel
```

On Solus:

```
sudo eopkg it -c system.devel
```

```
sudo eopkg it binutils-gold git glibc curl-devel libgtkmm-3-devel librsvg-devel \
  util-linux-devel sqlite3-devel libboost-devel zeromq-devel glm libgit2-devel \
  opencascade-ce-devel podofo-devel libarchive-devel cppzmq-devel
```

## 33.2 Build it

```
make -j 4 #adjust this to the number of CPU cores
```

Add `WITH_SPNAV=0` to disable space navigator support and remove the dependency on libspnav

## 33.3 Running

The resulting binaries are self-contained and don't require any external data files like icons or so. `horizon-eda` is the main program executable. Run it from the build directory:

```
build/horizon-eda
```

## BUILDING ON FREEBSD

Building horizon on FreeBSD is as simple as `gmake` after you've cloned this repo.

### 34.1 Install dependencies

```
sudo pkg install git gmake pkgconf e2fsprogs-libuuid sqlite3 \  
  gtkmm30 cppzmq libgit2 boost-libs glm opencascade podofo libzip
```

### 34.2 Build it

```
gmake -j 4 #adjust this to the number of CPU cores
```

### 34.3 Running

The resulting binaries are self-contained and don't require any external data files like icons or so. `horizon-eda` is the main program executable. Run it from the build directory:

```
build/horizon-eda
```



## PARAMETER PROGRAMS

As explained on other pages, horizon supports parametrizable padstacks and (to a limited extent) packages. To apply the given parameters to the existing geometry, each padstack and the like is accompanied by a small program.

These programs are written in a custom stack-based language. Users of HP calculators should feel familiar. Since there aren't any loops, these programs will terminate in finite time. The stack holds signed 64bit integers. Conceptually, it grows from top to bottom.

### 35.1 Syntax

On the top level, a program is made up of tokens. Tokens are separated by any amount of whitespace.

Token types:

- Integers: a number, optionally prefixed by a sign
- Dimension: a number with optional fractional part, suffixed by “mm”. The float before mm will get multiplied by 1e6, since horizon's internal unit of measurement is 1nm
- Mathematical operators such as: + - \* /
- Strings
- Argument start [ and end ] any token between these two will get appended to the last command's arguments

### 35.2 Generic Commands

#### 35.2.1 Zero-operand

`get-parameter [ <parameter> ]` gets parameter and pushes it onto the stack

#### 35.2.2 One-operand

Before the operation, the stack looks like this:

```
. .  
. .  
+---+  
| a |  
+---+
```

(continues on next page)

(continued from previous page)

```
Operators:
    | pushes
dup | a a
chs | -a
```

### 35.2.3 Two-operand

Before the operation, the stack looks like this:

```
  .  .
  .  .
+---+
| a |
+---+
| b |
+---+
```

```
Operators:
    | pushes
+   | a+b
-   | a-b
*   | a*b
/   | a/b
dupc | a b a b (Duplicate coordinate)
```

### 35.2.4 Three-operand

Before the operation, the stack looks like this:

```
  .  .
  .  .
+---+
| a |
+---+
| b |
+---+
| c |
+---+
```

```
Operators:
    | pushes
+xy | a+c b+c
-xy | a-c b-c
```

## 35.3 Padstack commands

In order for an object (shape, etc.) to be manipulated by the program, it needs to be assigned a parameter class. `## set-shape` `set-shape` [ `<parameter class>` `<form>` ] Sets a shape to the specified form or moves it to the specified position Valid forms:

- `rectangle`, pops height, width
- `circle`, pops diameter
- `obround`, pops height, width
- `position`, pops y, x

### 35.3.1 set-hole

`set-hole` [ `<parameter class>` `<shape>` ] Sets a hole to the specified shape Valid shapes:

- `round`, pops diameter
- `slot`, pops length, diameter
- `position`, pops y, x

## 35.4 Polygon commands (padstack and package)

### 35.4.1 set-polygon

`set-polygon` [ `<parameter class>` `<shape>` `<x0>` `<y0>` ] Sets a polygon to the specified shape with center at (x0,y0) Valid shapes:

- `rectangle`, pops height, width
- `circle`, pops diameter

### 35.4.2 set-polygon-vertices

`set-polygon-vertices` [ `<parameter class>` `<n_vertices>` ] Pops `n_vertices` coordinates from the stack and replaces the polygon's vertices with them.

### 35.4.3 expand-polygon

`expand-polygon` [ `<parameter class>` `<x0>` `<y0>` `<x1>` `<y1>` ... `<xn>` `<yn>` ] Pops expansion. Expands the polygon specified by the coordinates in the argument by the dimension popped from the stack.

## 35.5 Example program (from SMD rectangular padstack)

```
get-parameter [ pad_width ]
get-parameter [ pad_height ]
dupc dupc
set-shape [ pad rectangle ]
get-parameter [ solder_mask_expansion ]
2 *
+xy
set-shape [ mask rectangle ]

get-parameter [ paste_mask_contraction ]
2 *
-xy
set-shape [ paste rectangle ]
```



## THEORY OF OPERATION

### 36.1 Interactive manipulator

Horizon's primary interface is the so-called "Interactive manipulator" (imp). It's the unified editor for symbols, schematics, padstacks, packages and boards.

#### 36.1.1 Canvas

The canvas renders objects such as symbols, packages or tracks. The output of the rendering are line segments and triangles that get uploaded to the GPU for drawing. For rendering to non-OpenGL targets, the canvas provides hooks to get more information on what's being rendered. So far, this is used by the gerber exporter, 3D preview and DRC.

#### 36.1.2 Core

Since some documents such as symbols and schematics contain the same type of object (e.g. texts) and schematic and netlist need to be modified in-sync, some encapsulation has to take place. The core can be considered the glue between the document, the canvas and the tools.

#### 36.1.3 Tools

For each action the user can do, there's a tool. Once started, a tool receives keyboard and mouse input and modifies the document accordingly by means of the core. When needed, a tool may bring up additional dialogs for requesting information from the user.

#### 36.1.4 Property editor

Low-complexity adjustments such as line width don't warrant their own tool, that's why the core provides a property interface. The property editor's widgets are automatically generated from the object's description.



## CLI USAGE

The project and the pool manager have pretty much eliminated the need to run the interactive manipulator and other tools directly from a shell, but it's still useful for development.

All of the commands below require the environment variable `HORIZON_POOL` to point to the pool's directory (the one with the `pool.json` and `pool.db` in it)

### 37.1 horizon-imp

Symbol mode:

```
horizon-imp -y <symbol file>
```

Schematic mode:

```
horizon-imp -c <schematic file> <block file>
```

Padstack mode:

```
horizon-imp -a <padstack file>
```

Package mode:

```
horizon-imp -k <package file>
```

Board mode:

```
horizon-imp -b <board file> <block file> <via directory>
```

### 37.2 horizon-pool

Most of the `-edit` and `-create` commands will spawn `$EDITOR` with the file to be edited serialized as YAML.

```
horizon-pool create-unit <unit file>
horizon-pool edit-unit <unit file>
horizon-pool create-symbol <symbol file> <unit file>

horizon-pool create-entity <entity file> [<unit file> ...]
```

(continues on next page)

(continued from previous page)

```
horizon-pool edit-entity <entity file>

horizon-pool create-package <package file>
horizon-pool create-padstack <padstack file>

horizon-pool update #Recreates the pool's SQLite database.
```

Remember to run `horizon-pool update` after creating things

## 37.3 horizon-prj

Use these to create empty blocks, schematics, etc.

```
horizon-prj create-block <block filename>

horizon-prj create-schematic <schematic filename> <block filename>

horizon-prj create-board <schematic filename> <block filename>
```

## PYTHON MODULE

Parts of Horzion EDA are available as a python module for use in scripts.

### 38.1 Installation

The python module isn't included in the all target. To build it, run `make build/horizon.so`. This requires the python 3 headers to be installed. You can then place it in python's `sys.path` and import it using `import horizon`.

### 38.2 Usage

```
import horizon

#open project
p=horizon.Project("/path/to/project.hprj")

#open schematic
sch = p.open_top_schematic()

#export PDF
pdf_settings = sch.get_pdf_export_settings()
pdf_settings['output_filename'] = '/tmp/sch.pdf'
sch.export_pdf(pdf_settings)

#export BOM
bom_settings = sch.get_bom_export_settings()
bom_settings['output_filename'] = '/tmp/bom.csv'
sch.export_bom(bom_settings)

#open board
brd = p.open_board()

#export gerber
gerber_settings = brd.get_gerber_export_settings()
gerber_settings["output_directory"] = "/tmp/gerber"
brd.export_gerber(gerber_settings)

#export pick&place
pnp_settings = brd.get_pnp_export_settings()
```

(continues on next page)

(continued from previous page)

```
pnnp_settings["output_directory"] = "/tmp/pnp"
brd.export_pnp(pnp_settings)

#export STEP
step_settings = brd.get_step_export_settings()
step_settings["filename"] = "/tmp/pca.step"
brd.export_step(step_settings)

#run DRC
rules=brd.get_rules()
#modify rules if needed
rule_ids = brd.get_rule_ids()
#if needed, remove unneeded checks from rule_ids
result = brd.run_checks(rules, ids)

#export 3D rendering (see next section)
exporter = brd.export_3d(1920, 1080) #width, height
exporter.view_all()
exporter.load_3d_models() #optional
exporter.render_to_png("brd.png")
```

To further adjust the export settings, have a look at the dicts returned by the `get_*_export_settings` methods.

## 38.3 3D rendering usage

use `brd.export_3d(1920, 1080)` or similar to get an `Image3DExporter` object

**class** `Image3DExporter`

**render\_to\_png**(*filename*)

Render to png image

**render\_to\_surface**()

Render to pycairo surface

**Return type** `cairo.Surface`

**load\_3d\_models**()

Loads 3D models if available

**view\_all**()

Resets view to top side

**cam\_azimuth: float**

Camera azimuth angle in degrees

**cam\_elevation: float**

Camera elevation angle in degrees

**cam\_fov: float**

Camera field of view in degrees

**cam\_distance: float**

Camera distance in millimeters

**center\_x: float**

Where the camera looks at (millimeter)

**center\_y: float**

Where the camera looks at (millimeter)

**background\_top\_color: 3-tuple of float**

Background color at the top, components range from 0 to 1

**background\_bottom\_color: 3-tuple of float**

Background color at the bottom, components range from 0 to 1

**solder\_mask\_color: 3-tuple of float**

Solder mask color, components range from 0 to 1

**substrate\_mask\_color: 3-tuple of float**

Color of the PCB body, components range from 0 to 1

**ortho: bool**

Use orthographic projection

**show\_models: bool**

Show 3D models

**show\_silkscreen: bool**

**show\_solder\_mask: bool**

**show\_solder\_paste: bool**

**show\_substrate: bool**

**use\_layer\_colors: bool**

Use layer colors from 2D view for copper layers





## FILE VERSIONS

To protect against loss of fidelity when opening files in an older version of Horizon EDA than they were created with, version 1.3.0 introduces the concept of file versions.

Rather than storing the application version in design files and pool items, each file type has its own version number that'll get incremented if the file format changes in a way that's incompatible with older versions. That way, warnings about upgrading files are only shown if needed. Forward compatibility, as in being able to open files that were created in an earlier version, is always given.

### 39.1 By Application version

Type	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0
Unit	0	0	0	0	0	<b>1</b>
Symbol	0	0	0	0	<b>1</b>	1
Entity	0	0	0	0	0	0
Padstack	0	0	0	0	0	0
Package	0	0	0	0	0	0
Part	0	0	<b>1</b>	1	<b>2</b>	2
Frame	0	0	0	0	0	0
Decal	0	0	0	0	0	0
Schematic	0	0	<b>1</b>	1	<b>3</b>	<b>6</b>
Board	0	<b>2</b>	<b>4</b>	4	<b>7</b>	<b>14</b>
Project	0	0	<b>1</b>	1	<b>2</b>	2
Pool	N/A	N/A	N/A	N/A	0	<b>1</b>

### 39.2 Changelog

As of Horizon EDA Version 1.3.0, all object types are at version 0. Any changes will be listed here once they happen.

#### Board:

- 1: Add holes to PDF export
- 2: Support pick & place export format customisation
- 3: Add silkscreen color
- 4: Add rule net class regex matching
- 5: Add shorted pads rule

- 6: Add pads only flag to silkscreen exposed copper rule
- 7: Actually serialize from rules option for planes
- 8: Add matching multiple nets in rules
- 9: Add matching multiple components in rules
- 10: Add thermal rules
- 11: Add thermal spoke customisation
- 12: Add net ties
- 13: Add ODB++ export
- 14: Add track connection offset

**Schematic:**

- 1: Add custom values on symbols
- 2: Add hierarchy
- 3: Add name orientation to block symbol ports
- 4: Add connectivity checks
- 5: Add support for UUID-based alternate pin names with direction
- 6: Add net ties

**Project:**

- 1: Replace pool cache with project pool
- 2: Add hierarchy

**Part:**

- 1: Add flags
- 2: Add prefix override

**Symbol:**

- 1: Fix orientation-specific text placement

**Pool:**

- 1: Add default frame

**Unit:**

- 1: Use UUIDs for alternate pin names and support directions

## B

background\_bottom\_color (*Image3DExporter attribute*), 123

background\_top\_color (*Image3DExporter attribute*), 123

## C

cam\_azimuth (*Image3DExporter attribute*), 122

cam\_distance (*Image3DExporter attribute*), 122

cam\_elevation (*Image3DExporter attribute*), 122

cam\_fov (*Image3DExporter attribute*), 122

center\_x (*Image3DExporter attribute*), 122

center\_y (*Image3DExporter attribute*), 123

## I

Image3DExporter (*built-in class*), 122

## L

load\_3d\_models() (*Image3DExporter method*), 122

## O

ortho (*Image3DExporter attribute*), 123

## R

render\_to\_png() (*Image3DExporter method*), 122

render\_to\_surface() (*Image3DExporter method*), 122

## S

show\_models (*Image3DExporter attribute*), 123

show\_silkscreen (*Image3DExporter attribute*), 123

show\_solder\_mask (*Image3DExporter attribute*), 123

show\_solder\_paste (*Image3DExporter attribute*), 123

show\_substrate (*Image3DExporter attribute*), 123

solder\_mask\_color (*Image3DExporter attribute*), 123

substrate\_mask\_color (*Image3DExporter attribute*), 123

## U

use\_layer\_colors (*Image3DExporter attribute*), 123

## V

view\_all() (*Image3DExporter method*), 122