KiPart Documentation

Release 0.1.30

XESS Corp.

Oct 05, 2017

Contents

Contents:

CHAPTER 1

KiPart

Generate multi-unit schematic symbols for KiCad from a CSV file.

- Free software: MIT license
- Documentation: https://xesscorp.github.io/KiPart.

Features

- Generates schematic part libraries for KiCad from CSV files.
- Converts lists of pins in a CSV file into a multi-unit schematic part symbol.
- Converts multiple CSV files stored in .zip archives.
- Each row of the CSV file lists the number, name, type, style, unit and side of a pin.
- Pins on a unit with the same name (e.g., GND) are placed at the same location so they can all be tied to the same net with a single connection.
- Also includes `kilib2csv` for converting schematic part libraries into CSV files suitable for input to KiPart.

CHAPTER 2

Installation

This is a Python package, so you'll need to have Python installed to use it. If you're using linux, you probably already have Python. If you're on Windows, you can download a Python installer from Anaconda , Active State , or even WinPython .

Once you have Python, you can install this package by opening a terminal window and typing the command:

```
$ easy_install kipart
```

Or:

```
$ pip install kipart
```

CHAPTER 3

Usage

KiPart

KiPart is mainly intended to be used as a script:

```
usage: kipart [-h] [-v]
              [-r [{generic,xilinxultra,xilinx7,xilinx6s,xilinx6v,psoc5lp,stm32cube,
↪lattice}]]
              [-s [{row,num,name}]] [--reverse]
              [--side [{left,right,top,bottom}]] [-o [file.lib]] [-f] [-b]
              [-a] [-w] [-d [LEVEL]]
              file1.[csv|zip] file2.[csv|zip] ... [file1.[csv|zip]
              file2.[csv|zip] ... ...]

Generate single & multi-unit schematic symbols for KiCad from a CSV file.

positional arguments:
  file1.[csv|zip] file2.[csv|zip] ...
                        Files for parts in CSV format or as CSV files in .zip
                        archives.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -r [{generic,xilinxultra,xilinx7,xilinx6s,xilinx6v,psoc5lp,stm32cube,lattice}], --
↪reader [{generic,xilinxultra,xilinx7,xilinx6s,xilinx6v,psoc5lp,stm32cube,lattice}]
                        Name of function for reading the CSV file.
  -s [{row,num,name}], --sort [{row,num,name}]
                        Sort the part pins by their entry order in the CSV
                        file, their pin number, or their pin name.
  --reverse             Sort pins in reverse order.
  --side [{left,right,top,bottom}]
                        Which side to place the pins by default.
  -o [file.lib], --output [file.lib]
                        Generated KiCad library for part.
```

```
  -f, --fuzzy_match     Use approximate string matching when looking-up the
                        pin type, style and orientation.
  -b, --bundle          Bundle multiple, identically-named power, ground and
                        no-connect pins each into a single schematic pin.
  -a, --append, --add   Add parts to an existing part library. Overwrite
                        existing parts only if used in conjunction with -w.
  -w, --overwrite       Allow overwriting of an existing part library.
  -d [LEVEL], --debug [LEVEL]
                        Print debugging info. (Larger LEVEL means more info.)
```

A generic part file is expected when the `-r generic` option is specified. It contains the following items:

1. The part name or number stands alone on row. The following three cells on the row can contain:
   (a) A reference prefix such as `R` (defaults to `U` if left blank),
   (b) A footprint such as `Diodes_SMD:D_0603` (defaults to blank),
   (c) A manufacturer's part number such as `MT48LC16M16A2F4-6A:GTR` (defaults to blank).
2. The next non-blank row contains the column headers. The required headers are 'Pin' and 'Name'. Optional columns are 'Unit', 'Side', 'Type', 'Style', and 'Hidden'. These can be placed in any order and in any column.
3. On each succeeding row, enter the pin number, name, unit identifier (if the schematic symbol will have multiple units), pin type and style. Each of these items should be entered in the column with the appropriate header.
   - Pin numbers can be either numeric (e.g., '69') if the part is a DIP or QFP, or they can be alphanumeric (e.g., 'C10') if a BGA or CSP is used. Placing a * at the start of a pin number creates a non-existent "gap" pin that can be used to divide the pins into groups. This only works when the `-s row` sorting option is selected.
   - Pin names can be any combination of letters, numbers and special characters (except a comma).
   - The unit identifier can be blank or any combination of letters, numbers and special characters (except a comma). A separate unit will be generated in the schematic symbol for each distinct unit identifier.
   - **The side column specifies the side of the symbol the pin will be placed on. The allowable values are:**
     - left
     - right
     - top
     - bottom
   - **The type column specifies the electrical type of the pin. The allowable values are:**
     - input, inp, in, clk
     - output, outp, out
     - bidirectional, bidir, bi, inout, io, iop
     - tristate, tri
     - passive, pass
     - unspecified, un, analog
     - power_in, pwr_in, pwrin, power, pwr, ground, gnd
     - power_out, pwr_out, pwrout, pwr_o
     - open_collector, opencollector, open_coll, opencoll, oc

    - open_emitter, openemitter, open_emit, openemit, oe
    - no_connect, noconnect, no_conn, noconn, nc
  - **The style column specifies the graphic representation of the pin. The allowable pin styles are:**
    - line, <blank>
    - inverted, inv, ~, #
    - clock, clk, rising_clk
    - inverted_clock, inv_clk, clk_b, clk_n, ~clk, #clk
    - input_low, inp_low, in_lw, in_b, in_n, ~in, #in
    - clock_low, clk_low, clk_lw, clk_b, clk_n, ~clk, #clk
    - output_low, outp_low, out_lw, out_b, out_n, ~out, #out
    - falling_edge_clock, falling_clk, fall_clk
    - non_logic, nl, analog
  - The hidden column specifies whether the pin is visible in Eeschema. This can be one of 'y', 'yes', 't', 'true', or '1' to make it invisible, anything else makes it visible.
4. A blank row ends the list of pins for the part.
5. Multiple parts (each consisting of name, column header and pin rows) separated by blank lines are allowed in a single CSV file. Each part will become a separate symbol in the KiCad library.

When the option `-r xilinx7` is used, the individual CSV pin files or entire .zip archives for the Xilinx 7-Series FPGAs can be processed.

When the option `-r psoc5lp` is used, the CSV pin file contains the pinout text extracted from a Cypress PSoC5LP datasheet.

When the option '-r stm32cube' is used, the input CSV file should be the pin layout file exported from the STM32CubeMx tool. To create this file; create a project with STM32CubeMx then from window menu select "Pinout -> Generate CSV pinout text file". If you select pin features or define labels for pins these will be reflected in the generated library symbol.

When the option `-r lattice` is used, the input CSV file should come from the Lattice website or from their Diamond tool. (The iCE40 FPGAs are not supported since they use a different format.)

The `-s` option specifies the arrangement of the pins in the schematic symbol:

- `-s row` places the pins in the order they were entered into the CSV file.
- `-s num` places the pins such that their pin numbers are in increasing order.
- `-s name` places the pins in increasing order of their names.

The `--reverse` option reverses the sort order for the pins.

Using the `--side` option you can set the default side for the pins. The option from the CSV file will override the command line option. Default choice is `left`.

Specifying the `-f` option enables *fuzzy matching* on the pin types, styles and sides used in the CSV file. So, for example, `ck` would match `clk` or `rgt` would match `right`.

Specifying the `-b` option will place multiple pins with the identical names at the same location such that they can all attach to the same net with a single connection. This is helpful for handling the multiple VCC and GND pins found on many high pin-count devices.

The `-w` option is used to overwrite an existing library with any new parts from the CSV file. The old contents of the library are lost.

The `-a` option is used to add parts to an existing library. If a part with the same name already exists, the new part will only overwrite it if the `-w` flag is also used. Any existing parts in the library that are not overwritten are retained.

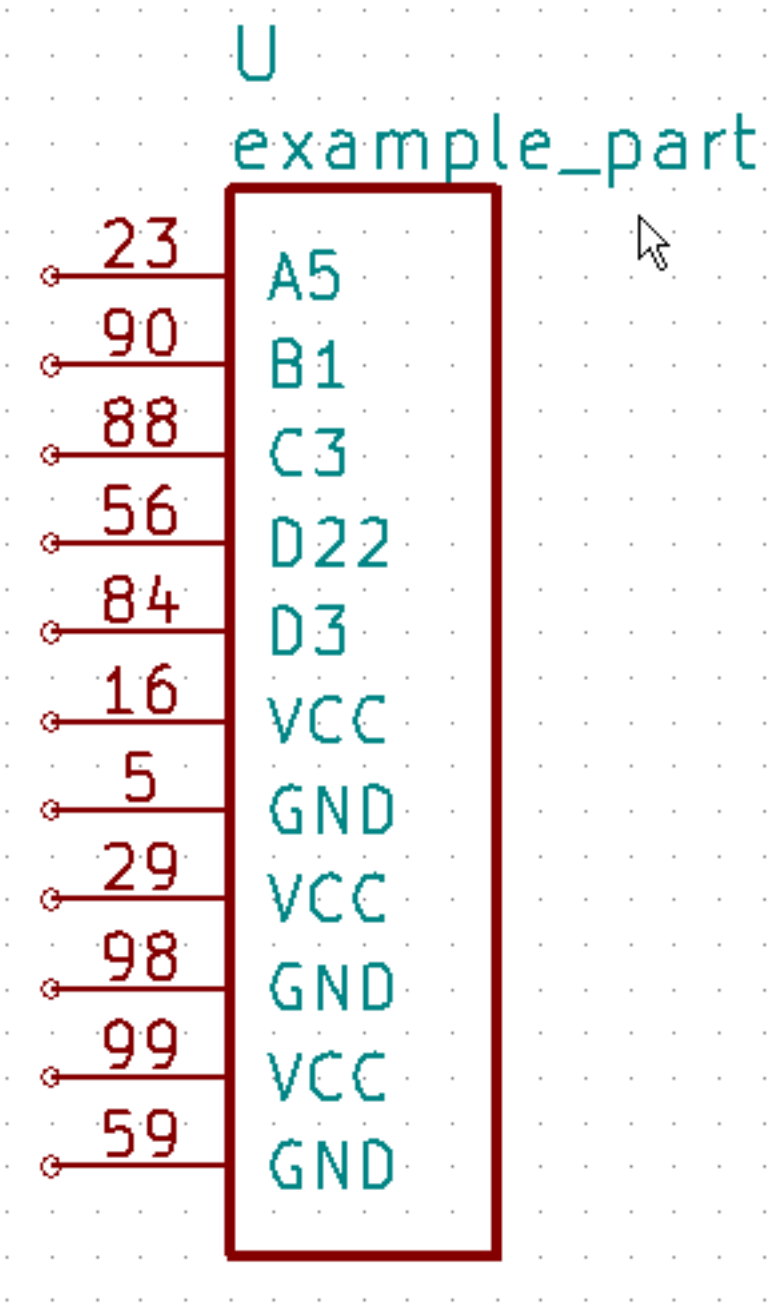
Examples

Assume the following data for a single-unit part is placed into the *example.csv* file:

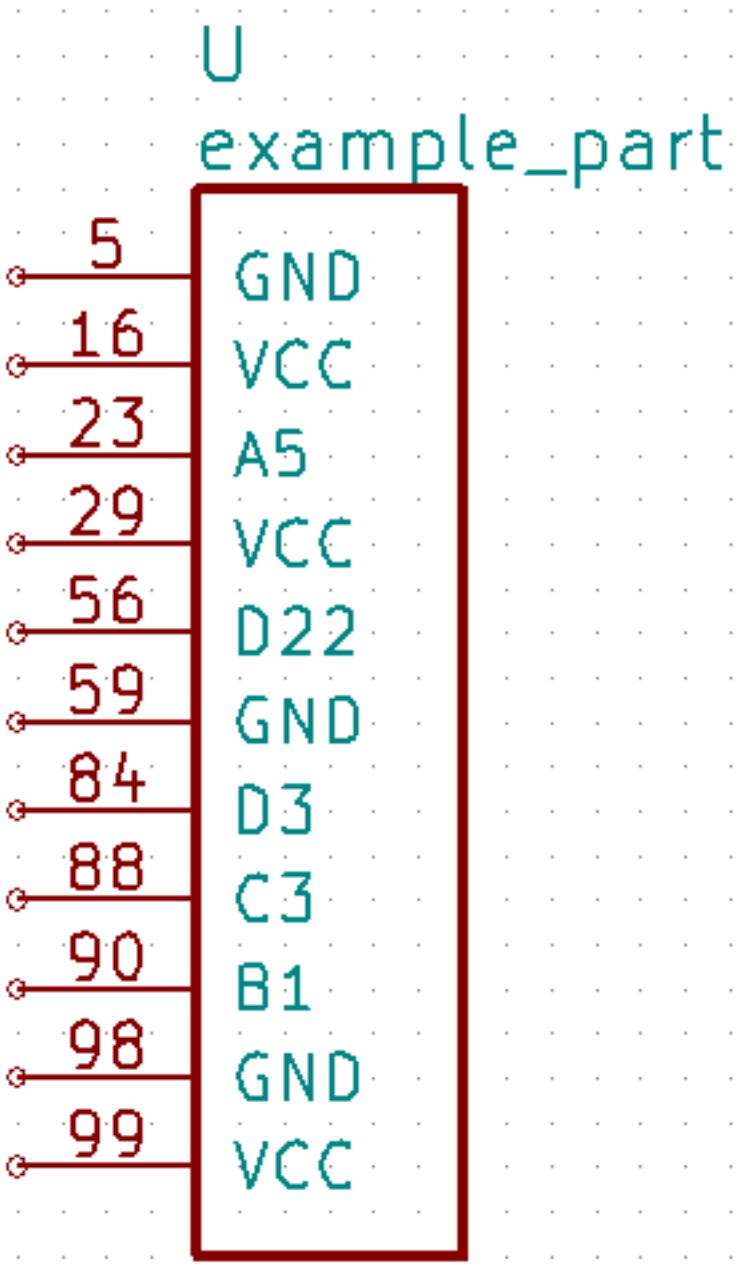
```
example_part

Pin,    Type,           Name
23,     input,          A5
90,     output,         B1
88,     bidirectional,  C3
56,     tristate,       D22
84,     tristate,       D3
16,     power_in,       VCC
5,      power_in,       GND
29,     power_in,       VCC
98,     power_in,       GND
99,     power_in,       VCC
59,     power_in,       GND
```

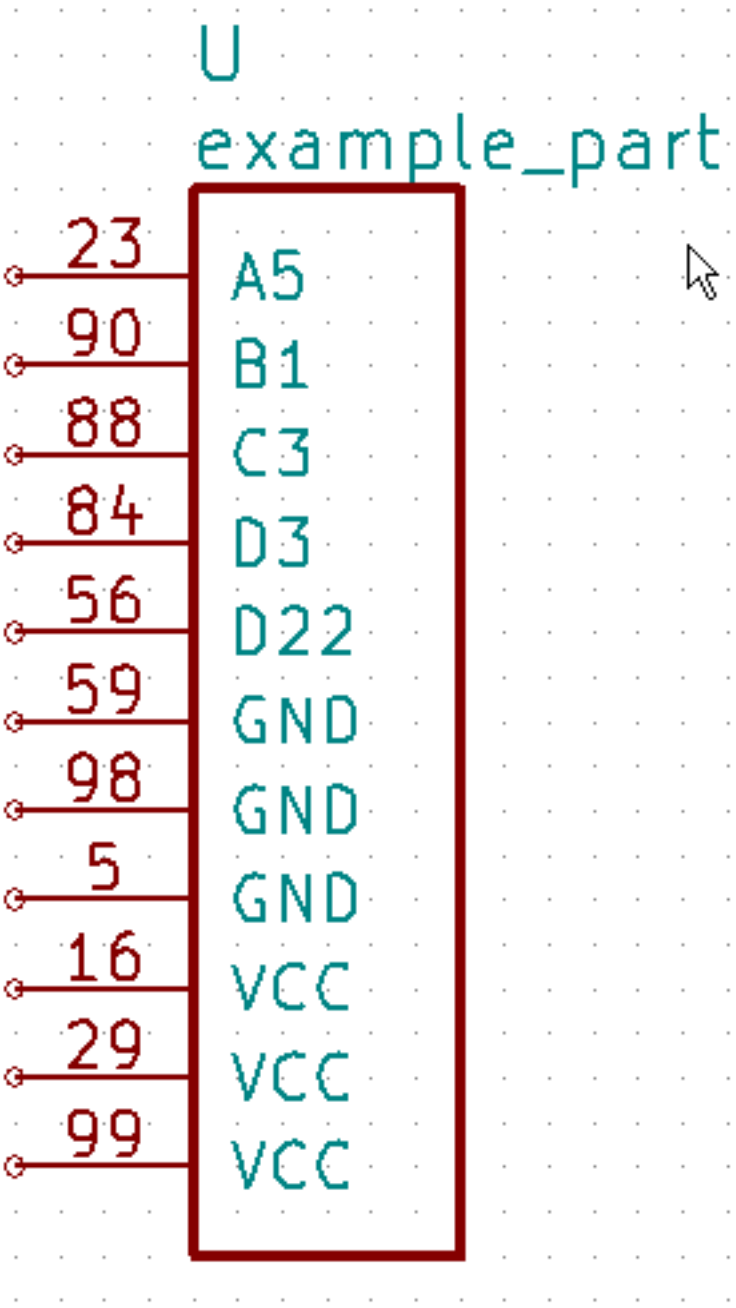
Then the command `kipart example.csv -o example1.lib` will create a schematic symbol where the pins are arranged in the order of the rows in the CSV file they are on:



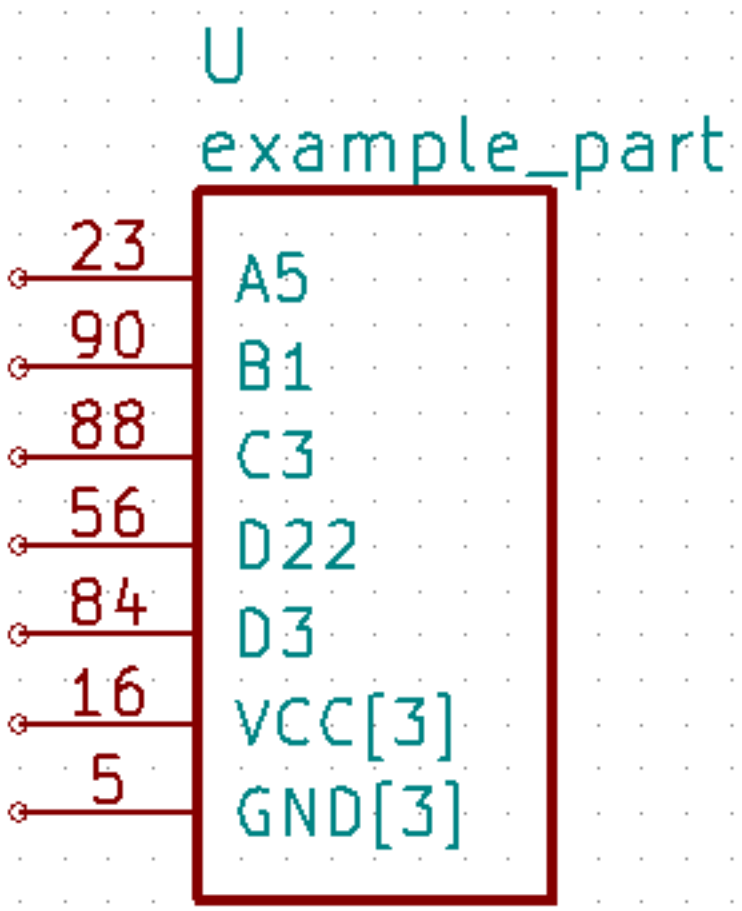
The command `kipart -s num example.csv -o example2.lib` will create a schematic symbol where the pins are arranged by their pin numbers:



The command `kipart -s name example.csv -o example3.lib` will create a schematic symbol where the pins are arranged by their names:



The command `kipart -b example.csv -o example4.lib` will bundle power and no-connect pins with identical names (like `GND`, `VCC`, and `NC`) into single pins like so:

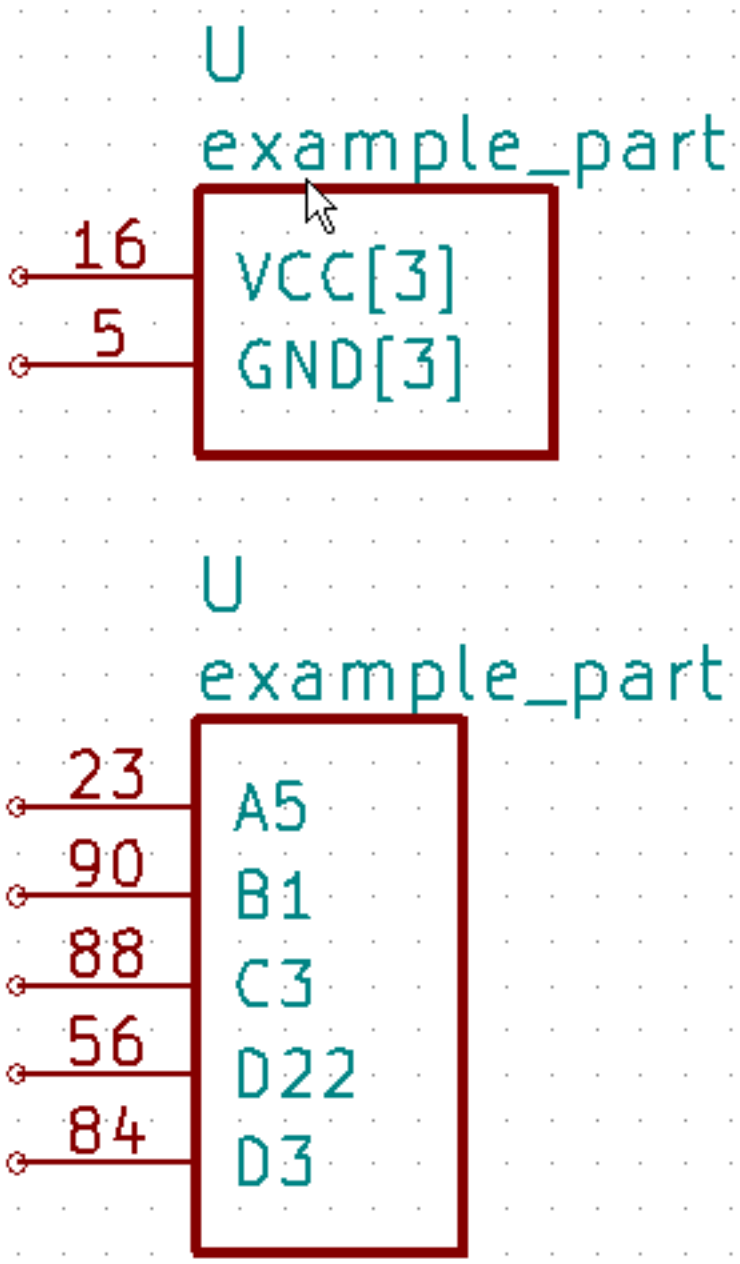


Or you could divide the part into two units: one for I/O pins and the other for power pins by adding a `Unit` column like this:

```
example_part

Pin,     Unit,    Type,           Name
23,      IO,      input,          A5
90,      IO,      output,         B1
88,      IO,      bidirectional,  C3
56,      IO,      tristate,       D22
84,      IO,      tristate,       D3
16,      PWR,     power_in,       VCC
5,       PWR,     power_in,       GND
29,      PWR,     power_in,       VCC
98,      PWR,     power_in,       GND
99,      PWR,     power_in,       VCC
59,      PWR,     power_in,       GND
```

Then the command `kipart -b example.csv -o example5.lib` results in a part symbol having two separate units:



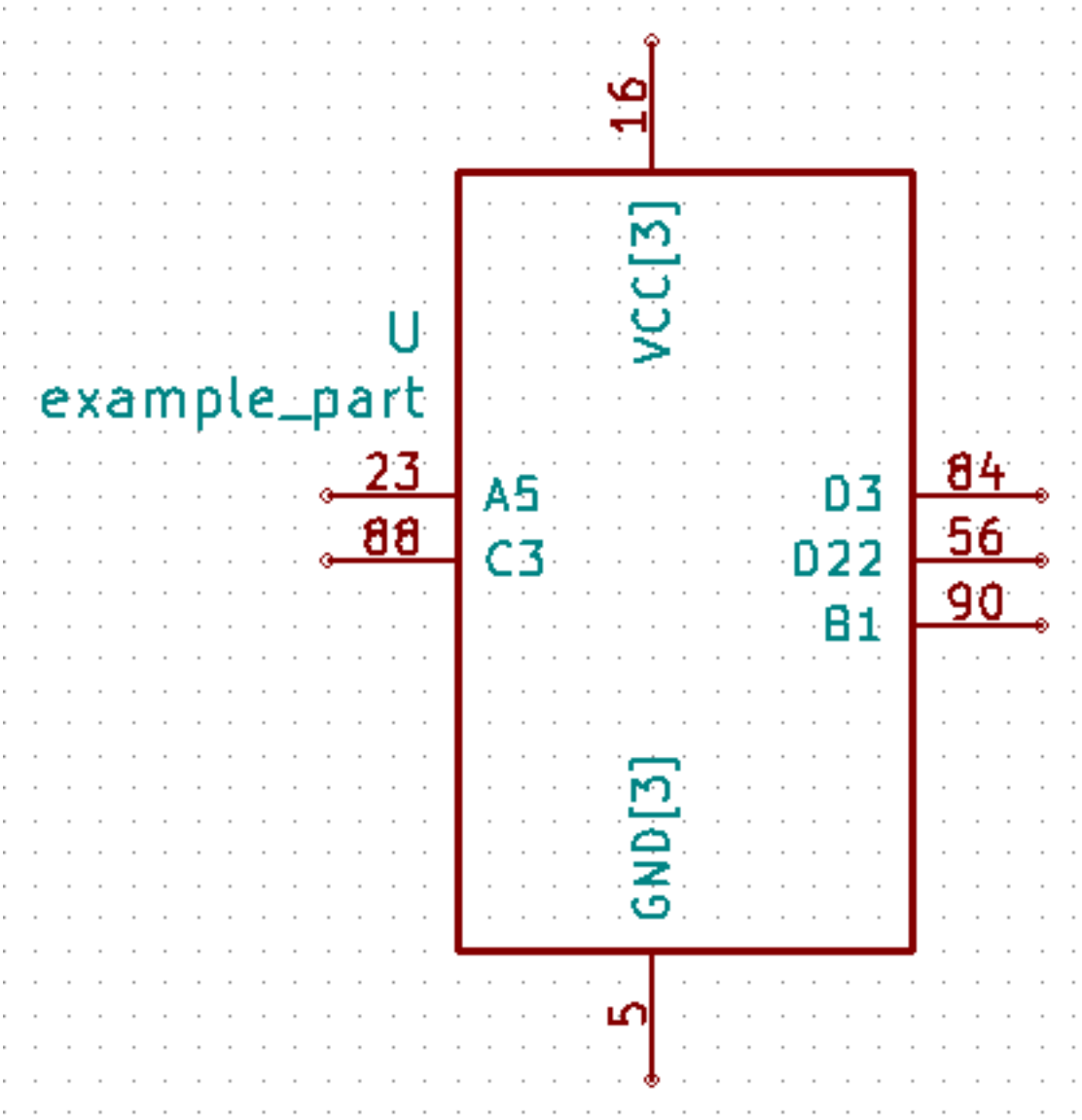
As an alternative, you could go back to a single unit with all the inputs on the left side, all the outputs on the right side, the VCC pins on the top and the GND pins on the bottom:

```
example_part

Pin,     Unit,    Type,           Name,    Side
23,      1,       input,          A5,      left
90,      1,       output,         B1,      right
88,      1,       bidirectional,  C3,      left
56,      1,       tristate,       D22,     right
84,      1,       tristate,       D3,      right
16,      1,       power_in,       VCC,     top
5,       1,       power_in,       GND,     bottom
```

```
29,     1,      power_in,       VCC,    top
98,     1,      power_in,       GND,    bottom
99,     1,      power_in,       VCC,    top
59,     1,      power_in,       GND,    bottom
```

Running the command `kipart -b example.csv -o example6.lib` generates a part symbol with pins on all four sides:

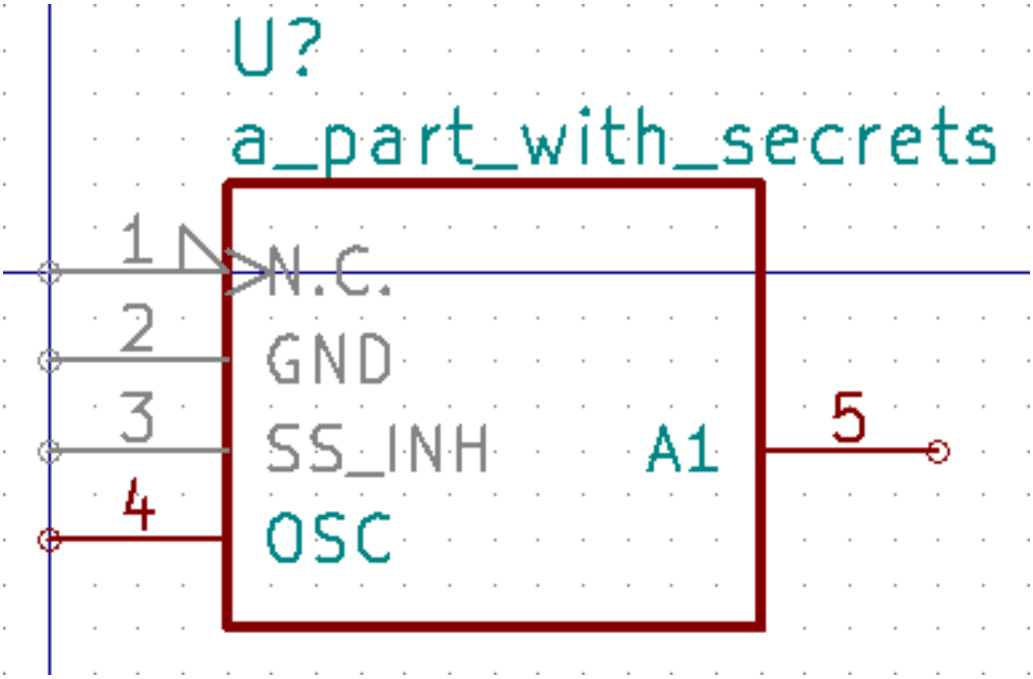


If the input file has a `Hidden` column, then some, none, or all pins can be made invisible:

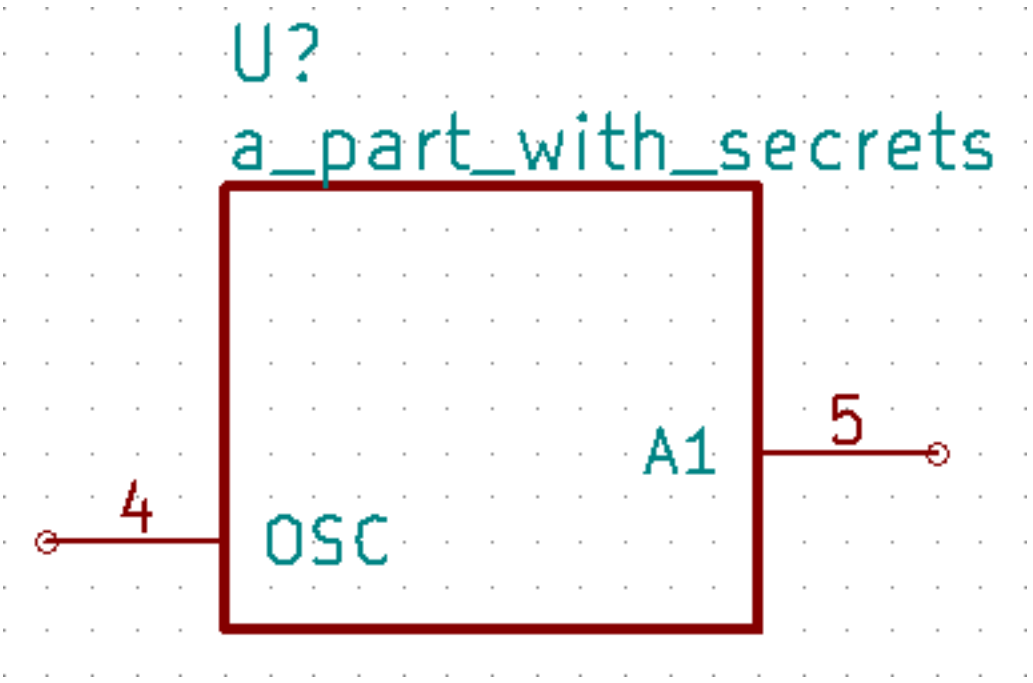
```
a_part_with_secrets

Pin,    Name,   Type,   Side,   Style,      Hidden
1,      N.C.,   in,     left,   clk_low,    Y
2,      GND,    pwr,    left,   ,           yeS
3,      SS_INH, in,     left,   ,           True
4,      OSC,    in,     left,   ,
5,      A1,     out,    right,  ,           False
```

In the Part Library Editor, hidden pins are grayed out:



But in Eeschema, they won't be visible at all:



kilib2csv

Sometimes you have existing libraries that you want to manage with a spreadsheet instead of the KiCad symbol editor. The kilib2csv utility takes one or more library files and converts them into a CSV file. Then the CSV file can be manipulated with a spreadsheet and used as input to KiPart. **(Note that any stylized part symbol graphics will be**

lost in the conversion. KiPart only supports boring, box-like part symbols.)

```
usage: kilib2csv-script.py [-h] [-v] [-o [file.csv]] [-a] [-w]
                           file.lib [file.lib ...]

Convert a KiCad schematic symbol library file into a CSV file for KiPart.

positional arguments:
  file.lib              KiCad schematic symbol library.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -o [file.csv], --output [file.csv]
                        CSV file created from schematic library file.
  -a, --append          Append to an existing CSV file.
  -w, --overwrite       Allow overwriting of an existing CSV file.
```

The utility is easy to use:

```
kilib2csv my_lib1.lib my_lib2.lib -o my_library.csv
```

Then you can generate a consistent library from the CSV file:

```
kipart my_library.csv -o my_library_new.lib
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/xesscorp/kipart/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

KiPart could always use more documentation, whether as part of the official KiPart docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/xesscorp/kipart/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *kipart* for local development.

1. Fork the *kipart* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/kipart.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv kipart
$ cd kipart/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

CHAPTER 5

Credits

Development Lead

- XESS Corp. <info@xess.com>

Contributors

- Hasan Yavuz OZDERYA (https://github.com/hyOzd)
- John Donovan (https://github.com/GeoSpark)
- Adrien Descamps (https://github.com/descampsa)

CHAPTER 6

History

0.1.30 (2017-10-05)

- Specifying `-a` option allows new parts to be written to an existing library but prevents overwriting existing parts. Using `-w` in conjunction with `-a` allows added parts to overwrite existing parts.
- Part name, reference prefix, footprint, and manf. part num. are now allowed on beginning row of part info in a CSV file.
- Expanded the lists of mnemonics for pin types and styles.

0.1.29 (2017-07-31)

- Fixed erroneous library generation when part number is omitted from first line of CSV file.
- Changed default output library to `kipart.lib` if no output library is specified.
- Changed default output CSV file of kilib2csv to `kipart.csv` if no output CSV file is specified.

0.1.28 (2017-07-27)

- Added reader for Lattice FPGA devices (except iCE40). (Thanks, Adrien Descamps!)

0.1.27 (2017-05-24)

- Fixed issue #11 (blank lines in CSV file were skipped and multiple parts ran together).

0.1.26 (2017-05-21)

- Fixed issue #18 (crash when symbol side for pin was left blank).

0.1.25 (2017-05-03)

- Fixed problem caused by pin side designators not being lower-case (e.g., “Left”).

0.1.24 (2016-12-22)

- Fixed Xilinx reader function to parse leading comments in their FPGA pin files.

0.1.23 (2016-12-13)

- Added ability to create hidden pins.

0.1.22 (2016-11-29)

- Fixed readers for Xilinx, STM32, PSoC devices.
- Pins on multiple sides of a symbol are now distributed in a more attractive manner.

0.1.21 (2016-09-20)

- Extra stuff on starting line of library no longer kill kilib2csv.

0.1.20 (2016-09-16)

- Fixed bug where kilib2csv was choking on footprint lists in part definitions.

0.1.19 (2016-09-16)

- Added utility to test kilib2csv and kipart on randomly-generated CSV part files.

0.1.18 (2016-09-14)

- kilib2csv utility added to convert KiCad schematic symbol libraries into CSV files suitable for input to KiPart.

0.1.17 (2016-06-15)

- Use same type of sorting for unit names as for pin names so (for example) unit 'ADC_12' comes before unit 'ADC_2'.

0.1.16 (2016-06-12)

- Added reader for CSV-formatted pinout files exported from the STM32CubeMx tool. (Thanks, Hasan Yavuz OZDERYA!)

0.1.15 (2016-02-17)

- Added reader for Xilinx Ultrascale FPGAs.
- Fixed insertion of spaces between groups of pins when pin number starts with '*'.
- Replaced call to warnings.warn with issues() function.
- fix_pin_data() now strips leading/trailing spaces from pin information.

0.1.14 (2016-01-30)

- Fixed incorrect y-offset of pins for symbols that only have pins along the right side.

0.1.13 (2015-09-09)

- The number of pins in a bundle is now appended to the pin name instead of an '*'.

0.1.12 (2015-09-03)

- Added capability to insert non-existent "gap" pins that divide groups of pins into sections.

0.1.11 (2015-09-02)

- future module requirement added to setup.py.

0.1.10 (2015-08-26)

- Now runs under both Python 2.7 and 3.4.

0.1.9 (2015-08-21)

- The bundling option now only bundles pins where that operation makes sense: power input pins (e.g., VCC and GND) and no-connect pins.

0.1.8 (2015-08-17)

- Input data from the CSV file is now scanned for errors and fixed before it can cause problems in the library file.

0.1.7 (2015-08-14)

- Added reader functions for Xilinx Virtex-6 and Spartan-6.
- Broke-out reader functions into separate modules.
- TXT and CSV files are now acceptable as part data files, but the reader has to be built to handle it.

0.1.6 (2015-08-13)

- Fuzzy string matching is now used for the column headers.
- Choice-type options are now case-insensitive.

0.1.5 (2015-07-29)

- Multiple parts can now be described in a single CSV file.
- Added function and option for reading Cypress PSoC5LP CSV files.
- Simplified key generators for sorting pins by name or number.
- Improved ordering of pins by name.

0.1.4 (2015-07-27)

- Added option for approximate (fuzzy) matching for pin types, styles and orientations (sides).

0.1.3 (2015-07-26)

- Multiple pins with the same name are now hidden by reducing their pin number size to zero (rather than enabling the hidden flag which can cause problems with power-in pins).

0.1.2 (2015-07-24)

- Symbols can now have pins on any combination of left, right, top and bottom sides.
- Added option to append parts to an existing library.
- Refactored kipart routine into subroutines.
- Added documentation.

0.1.1 (2015-07-21)

- Fixed calculation of pin name widths.
- Made CSV row order the default for arranging pins on the schematic symbol.
- Fixed sorting key routine for numeric pin numbers.
- Spaces are now stripped between fields in a CSV file.

0.1.0 (2015-07-20)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search