SVG Schematic Documentation

Release 1.2.0

Ken Kundert

Mar 29, 2024

CONTENTS

Author
Ken Kundert

Version
1.2.0

Released
2022-06-03

This package allows you to create simple SVG schematics and block diagrams without a mouse. Instead, you build the schematic using Python to instantiate and place the symbols and wires.

CHAPTER

ONE

SIMPLE EXAMPLE

Here is a simple example that demonstrates the package. It generates the schematic of a shunt RLC circuit:

```
from svg_schematic import Schematic, Resistor, Capacitor, Inductor, Wire
from inform import Error, error, os_error

try:
    with Schematic(filename='rlc.svg'):
        r = Resistor(name='R', orient='v')
        c = Capacitor(W=r.E, name='C', orient='v')
        l = Inductor(W=c.E, name='L', orient='v|')
        Wire([r.p, l.p])
        Wire([r.n, l.n])
except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

When run, it produces the following schematic:

CHAPTER TWO

INSTALLATION

Requires Python3. Works best with Python3.6 or newer.

You can download and install the latest stable version of the code from PyPI using:

```
pip3 install --user svg_schematic
```

You can find the latest development version of the source code on Github.

CHAPTER

THREE

ISSUES

Please ask questions or report problems on Github Issues.

CHAPTER

FOUR

CONTRIBUTIONS

Contributions in the form of pull requests are welcome.

I tend to create symbols as I need them. If you create missing symbols, please consider contributing them back to the project.

CHAPTER FIVE

CONTENTS

5.1 Concepts

With *svg_schematic* you simply pick one symbol and place it at the origin and then place the other symbols relative to the ones previously placed. For example:

```python
from svg_schematic import Schematic, Resistor, Capacitor, Inductor, Wire
from inform import Error, error, os_error

try:
    with Schematic(filename='rlc.svg'):
        r = Resistor(name='R', orient='v')
        c = Capacitor(C=r.C, xoff=100, name='C', orient='v')
        l = Inductor(C=c.C, xoff=100, name='L', orient='v')
        Wire([r.p, l.p])
        Wire([r.n, l.n])
except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

When run, it produces the following schematic:

5.1.1 Component Placement

In this example the resistor is placed without a location, and so its center is placed at the origin, (0, 0). You can then access the location of the center of the resistor using `r.C`, which is a XY-pair. That is passed to the capacitor using `C=r.C` with an extra parameter of `xoff=100`, meaning the center of the capacitor is horizontally offset by 100 units from center of the resistor. To give you a sense of how far 100 units is, the length of the resistor is 100 units. Positive horizontal offsets shift the location to the right, positive vertical offsets shift the location down. Finally, the inductor is placed 100 units to the right of the capacitor.

When specifying offsets, you can specify the x-offset using `xoff`, the y-offset using `yoff`, and you can specify both with `off` as a tuple. For example, `off=(50,25)` is equivalent to `xoff=50, yoff=25`.

Wires are added using an list of points, where each point is an XY-pair. In the simplest case, a line is run between each of the points specified. Thus, the first wire runs from `r.p` to `l.p`, where `r` is the resistor and `r.p` is the location of the `p` terminal of the resistor. `l.p` is the location of the positive terminal of the inductor. The second wire connects the negative pins.

5.1.2 Principle Coordinates

Each component is embedded in a tile, and each tile has 9 principle coordinate named C, N, NE, E, SE, S, SW, W, and NW which are short for center, north, northeast, east, southeast, south, southwest, west and northwest.

When placing a component, you can give the location of any of the principle coordinates. And once placed, you can access the location of any of the principle coordinates. Thus, the location of the components in the example could be specified simply by placing the tiles side-by-side:

```
with Schematic(filename = "rlc.svg"):
    r = Resistor(name='R', orient='v')
    c = Capacitor(W=r.E, name='C', orient='v')
    l = Inductor(W=c.E, name='L', orient='v|')
    Wire([r.p, c.p, l.p])
    Wire([r.n, c.n, l.n])
```

This places the west principle coordinate of `c` on the east principle coordinate of `r` and then the west principle coordinate of `l` on the east principle coordinate of `c`.

5.1.3 Pins as Coordinates

You can also specify and access the component pin locations. For example, with the resistor there are two terminals `p` and `n`.

Using this approach you can draw a series RLC using:

```
with Schematic(filename = "rlc.svg"):
    r = Resistor(name='R', orient='h')
    c = Capacitor(n=r.p, name='C', orient='h|')
    l = Inductor(n=c.p, name='L', orient='h')
```

When run, it produces the following schematic:

5.1.4 Orientation

You can flip and rotate the components using the `orient` argument. Specifying `v` implies a vertical orientation, and `h` a horizontal orientation (a component is converted from vertical to horizontal with a -90 degree rotation. Adding `|` implies the component should be flipped along a vertical axis (left to right) and adding `-` implies the component should be flipped along a horizontal axis (up to down).

5.1.5 Name and Value

With most components you can specify a name, and with many components you can also specify a value. The text orientation will always be horizontal regardless of the component orientation. You can also specify `nudge` as a small number to adjust the location of the resulting text. For example:

```
from svg_schematic import (
    Schematic, Capacitor, Ground, Inductor, Resistor, Pin, Source, Wire
)
from inform import Error, error, os_error

try:
    with Schematic(
        filename = 'lpf.svg',
        background = 'none',
    ):
        vin = Source(name='Vin', value='1 V', kind='sine')
        Ground(C=vin.n)
        rs = Resistor(name='Rs', value='50 ', n=vin.p, xoff=25)
        Wire([vin.p, rs.n])
        c1 = Capacitor(name='C1', value='864 pF', p=rs.p, xoff=25)
        Ground(C=c1.n)
        l2 = Inductor(name='L2', value='5.12 H', n=c1.p, xoff=25)
        Wire([rs.p, l2.n])
        c3 = Capacitor(name='C3', value='2.83 nF', p=l2.p, xoff=25)
        Ground(C=c3.n)
        l4 = Inductor(name='L4', value='8.78 H', n=c3.p, xoff=25)
        Wire([l2.p, l4.n])
        c5 = Capacitor(name='C5', value='7.28 nF', p=l4.p, xoff=25)
        Ground(C=c5.n)
        rl = Resistor(name='Rl', value='50 ', p=c5.p, xoff=100, orient='v')
        Ground(C=rl.n)
        out = Pin(name='out', C=rl.p, xoff=50, w=2)
        Wire([l4.p, out.t])
except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.1.6 Kind

Many components allow you to specify `kind`, which allow you to choose a variant of the component symbol. They include

Symbol	Kinds
BJT	`npn`, `pnp` (or `n`, `p`)
MOS	`nmos`, `pmos` (or `n`, `p`)
Amp	`se`, `oa`, `da`, `comp`
Gate	`inv`
Pin	`dot`, `in`, `out`, `none`
Label	`plain`, `arrow`, `arrow|`, `slash`, `dot`
Source	`empty`, `vdc`, `idc`, `sine`, `sum`, `mult` `cv`, `ci`
Switch	`spst`, `spdt`
Wire	`plain`, `|-`, `-|`, `|-|`, `-|-`

These are explained further later when the individual symbols are discussed.

5.1.7 Miscellany

There are a few things to note.

1. SVG coordinates are used, which inverts the y axis (more southern coordinates are more positive than the more northern coordinates).
2. Wires and components stack in layers, with the first that is placed going on the lowest layer. Most components contain concealers, which are small rectangles that are designed to conceal any wires that run underneath the components. This allows you to simply run a wire underneath the component rather than explicitly wire to each terminal, which can simply the description of the schematics. For this to work, the wire must be specified before the component. Also, the color of the concealers matches that of the background, so if you use no background, then you also lose the concealers.
3. Components are placed in invisible tiles. The unit size of a tile is 50. You have limited ability to specify the width and height of some components, and specifying the size as `w=1, h=1` implies the tile will be 50x50. Most components have a size of 2×2 and so sit within a 100x100 tile. You need not specify the size as an integer.
4. When the schematic is used with *Latex*, you can use Latex formatting in the name and value. For example, you can specify: *name='L_1'*. You should use raw strings if your string contains backslashes: *value=r'$10 \mu H$'*.
5. Components provide provide individual attributes for the location of each terminal. For example, the resistor, capacitor, and inductor components provide the *p* and *n* terminal attributes. The MOS component provides the *d*, *g*, and *s* terminal attributes. The diode component provides the *a* and *c* terminal attributes.
6. Components contain attributes for each of the 9 principal coordinates (C, N, NE, E, SE, S, SW, W, NW). For most components, these are the principal coordinates for the component's tile. However, the source places its principal coordinates on the circle used to depict the source.

5.1.8 Placement Strategies

There are two basic approaches to placing components. First, you may specify the coordinate in absolute terms. For example:

```
with Schematic(filename = "rlc.svg"):
    Wire([(-75, -50), (75, -50), (75, 50), (-75, 50)])
    Wire([(0, -50), (0, 50)])
    Resistor(C=(-75, 0), name='R', orient='v')
    Capacitor(C=(0, 0), name='C', orient='v')
    Inductor((C=(75, 0), name='L', orient='v|')
```

Notice that a wire is specified as a list of points, where each point is a tuple that contains an XY pair. The wire just connects the points with line segments. The location of the components is given by giving the location of a feature on the component. In this case it is the center (`C`) of the component that is specified. Again the location is an XY-pair.

This approach turns out to be rather cumbersome as it requires a lot of planning and is a lot of work if you need to move things around. In that case you likely have to adjust a large number coordinates. Since schematics of any complexity are often adjusted repeatedly before they are correct and aesthetically appealing, this approach can lead to a lot of tedious work.

The second basic approach to placing component is to place them relative to each other. This approach is the one that is always used in practice. To do so, you would generally take advantage of the fact that components have attributes that contains useful coordinate locations on the component. For example:

```
r = Resistor(C=(0, 0), name='R', orient='v')
```

Now, *r.C*, *r.N*, *r.NE*, *r.E*, *r.SE*, *r.S*, *r.SW*, *r.W*, and *r.NW* contain the coordinates of the center, north, northeast, east, southeast, south, southwest, west, and northwest corners. In addition, *r.p* and *r.n* hold the coordinates of the positive and negative terminals. Finally, wires provide the *b*, *m*, and *e* attributes, which contain the coordinates of their beginning, midpoint, and ending.

Once you place the first component, you then specify the location of the remaining components relative to one that has already been placed. To do so, you would give the location of one of the principle coordinates or the location of a terminal. For example:

```
r = Resistor(C=(0, 0), name='R', orient='v')
c = Capacitor(C=r.C, xoff=75, name='C', orient='v')
l = Inductor((C=c.C, xoff=75, name='L', orient='v|')
Wire([r.p, c.p, l.p], kind='-|-')
Wire([r.n, c.n, l.n], kind='-|-')
```

Notice that the center of `r` is placed at (0,0), then the center of `c` is place 75 units to the right of `r`, then the center of `l` is placed 75 units to the right of `c`. If `c` has to be moved for some reason then `l` will move with it. For example, only changing the line that instantiates the capacitor produces the following results:

```
c = Capacitor(C=r.C, off=(100, 25), name='C', orient='v')
```

The *shift*, *shift_x*, and *shift_y* utility functions are provided to shift the position of a coordinate pair. Examples:

```
shift((x,y), dx, dy) --> (x+dx, y+dy)
shift_x((x,y), dx) --> (x+dx, y)
shift_y((x,y), dy) --> (x, y+dy)
```

To see how these might be useful, consider offsetting the wires so they sit a little further away from the components:

```
r = Resistor(C=(0, 0), name='R', orient='v')
c = Capacitor(C=r.C, xoff=75, name='C', orient='v')
l = Inductor((C=c.C, xoff=75, name='L', orient='v|')
Wire([r.p, shift_y(r.p, -12.5), shift_y(c.p, -12.5), c.p])
Wire([c.p, shift_y(c.p, -12.5), shift_y(l.p, -12.5), l.p])
Wire([r.n, shift_y(r.n, 12.5), shift_y(c.n, 12.5), c.n])
Wire([c.n, shift_y(c.n, 12.5), shift_y(l.n, 12.5), l.n])
```

You can also use *with_x* and *with_y* to replace the *x* or *y* portion of a coordinate pair. They take two arguments, the first is returned with the appropriate coordinate component replaced by the second. The second argument may be a simple number or it may be a coordinate pair, in which case the appropriate coordinate component is used to replace the corresponding component in the first argument:

```
with_x((x1,y1), x2) --> (x2, y1)
with_y((x1,y1), y2) --> (x1, y2)
with_x((x1,y1), (x2,y2)) --> (x2, y1)
with_y((x1,y1), (x2,y2)) --> (x1, y2)
```

Finally, the *midpoint* functions return the point midway between two points:

```
midpoint((x1,y1), (x2,y2) --> ((x1+x2)/2, (y1+y2)/2)
midpoint_x((x1,y1), (x2,y2) --> ((x1+x2)/2, y1)
midpoint_y((x1,y1), (x2,y2) --> (x1, (y1+y2)/2)
```

5.1.9 Arbitrary Drawing Features using SVGwrite

SVG_Schematic subclasses the Python svgwrite package *Drawing* class. So you can call any *Drawing* method from a schematic. In this case you must keep the schematic instance to access the methods:

```
with Schematic(filename = "hello.svg") as schematic:
    schematic.circle(
        center=(0,0), r=100, fill='none', stroke_width=1, stroke='black'
    )
    schematic.text(
        'Hello', insert=(0,0), font_family='sans', font_size=16, fill='black'
    )
```

One thing to note is that *Schematic* normally keeps track of the location and extent of the schematic objects and sizes the drawing accordingly. It will be unaware of anything added directly to the drawing though the *svgwrite* methods. As a result, these objects may fall partially or completely outside the bounds of the drawing. You can add padding when you first instantiate *Schematic* or you can use the *svgwrite viewbox* method to extend the bounds.

5.1.10 Latex

To include these schematics into Latex documents, you need to run Inkscape with the –export-latex command line option to generate the files that you can include in Latex. Here is a Makefile that you can use to keep all these files up to date:

```
DRAWINGS = \
    flash-adc \
    pipeline-adc \
    delta-sigma-adc

SVG_FILES=$(DRAWINGS:=.svg)
PDF_FILES=$(DRAWINGS:=.pdf)
PDFTEX_FILES=$(DRAWINGS:=.pdf_tex)

.PHONY: clean
.PRECIOUS: %.svg

%.svg: %.py
        python3 $<

%.pdf: %.svg
        inkscape -z -D --file=$< --export-pdf=$@ --export-latex

clean:
        rm -rf $(PDF_FILES) $(PDFTEX_FILES) __pycache__
```

To include the files into your Latex document, use:

```
\def\svgwidth{0.5\columnwidth}
\input{delta-sigma.pdf_tex}
```

Finally, to convert your Latex file to PDF, use:

```
pdflatex --shell-escape converters.tex
```

5.1.11 Other Image Formats

You can use Image Magick package to convert SVG files to other image formats. For example:

```
convert receiver.svg receiver.png
```

5.2 Classes and Functions

5.2.1 Schematic

When creating a schematic you may specify the following arguments: `filename`, `font_size`, `font_family` (ex. ‘serif’ or ‘sans-serif’), `line_width`, and `dot_radius`. The dot radius is the radius of solder-dots and pins.

You can also specify `background` and `outline`, both of which are colors. The default background is ‘white’ and the default outline is ‘none’. If you set background to ‘none’ be aware that this makes the concealers transparent, meaning that you cannot wire under components, instead you must wire to the pins. It is common to start by setting outline to allow you to see the SVG drawing area, and then later remove it when your schematic is complete. Pad arguments are used to adjust the size of the SVG

The size of the SVG canvas is automatically sized to fit tightly around the specified schematic objects. You might find that the text associated with input and output pins has a tendency to extend beyond the canvas. This is because no attempt is made to estimate the width of text. Instead, you can increase the width of the pin's tile using its `w` parameter. In addition, you can also add padding when creating the schematic. There are five padding arguments. The most commonly used is `pad`, which simply adds the same padding to all four edges. In addition, you can control the individual edges using `left_pad`, `right_pad`, `top_pad`, and `bottom_pad`. These simply add to `pad` to create the final padding for each edge.

5.2.2 Wire

Draws a wire between two or more points given in sequence. Each point should be specified as a XY pair. Example:

```
Wire([(x0,y0), (x1,y1), (x2,y2), (x3,y3)])
```

Specifying wires before the components places them on a lower level, allowing the component to obscure the wires when needed.

Wire supports the *kind* argument, which may be either `plain`, `|-`, `-|`, `|-|`, or `-|-`. With plain, any-angle line segments are added between each of the points. With `|-`, `-|`, `|-|`, and `-|-` the wires are constrained to follow a Manhattan geometry (between each point there may be one, two, or three line segments that are constrained to be either purely vertical or purely horizontal. With `|-` there are two segments, with the first being vertical. With `-|`, there are also two segments, but the first is horizontal. With `|-|`, and `-|-` there there are three segments with the middle segment being half way between the two points. With `|-|`, the segments are vertical, horizontal, and vertical. With `-|-`, the segments are horizontal, vertical, and horizontal.

For example, if two resistors that are offset both horizontally and vertically are connected by a wire, the results depend on `kind` as follows:

Wire supports the `line_width` and `color` arguments.

Wire also supports arbitrary *svgwrite* drawing parameters. This can be useful to draw the wire with dashed lines:

```
Wire([(x0,y0), (x1,y1)], stroke_dasharray="4 2")
```

Wire provides the `b`, `m`, and `e` attributes that contain the coordinates of the beginning, the midpoint and the ending of the wire.

Label

Place a label. Five kinds of label are available, `plain`, `arrow`, `arrow|`, `slash`, and `dot`.

```
Label(kind='plain', name='plain', loc='se')
Label(kind='arrow', name='arrow', loc='se')
Label(kind='arrow|', name='arrow|', loc='se')
Label(kind='slash', name='slash', loc='se')
Label(kind='dot', name='dot', loc='se')
```

Here the labels are drawn with wires to give better context. The horizontal location of the labels is indicated with the vertical blue line.

Labels take the following arguments: `kind`, `orient`, `name`, `value`, `loc`, `w`, `h`, `color`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `off`, `xoff` and `yoff`. Currently `value` is ignored.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `t` attribute contains the coordinates of the label.

The kind may be 'plain', 'arrow', 'arrow|', 'slash' or 'dot'. If 'plain' is specified, no symbol is added, only the name is displayed. If 'arrow' is specified, an arrow is added and the centered on the specified location. If 'arrow|' is specified, the arrow terminates on the specified location. If 'slash' is specified, a small slash is added through the center. It is generally used with buses to indicate the bus width. Finally, 'dot' adds a solder dot.

By default the width and height of the label are 1, meaning that a unit sized tile (50×50) is used. This is significant if the label is at the edge of the schematic. If the labels extend beyond the tile, they may extend beyond the computed viewbox for the schematic. You can fix this by specifying a larger width.

It is important to remember that C represents the center of the tile used by the label. Since the label will be on one side, C will not coincide with the apparent visual center of the label.

5.2.3 Components

This section documents the available components. Components include an invisible tile in which the component should fit. The tile extent is used when determining the size of the overall schematic. Each component requires that you specify location by giving the location of its principle coordinates or a pin. You can specify an placement offset using `xoff`, `yoff`, or `off`. You can also generally specify the orientation, the name, the value, and a text offset using `orient`, `name`, `value`, and `nudge`.

The `orient` is specified as a string that generally consists of either 'v' or 'h', indicating that a vertical or horizontal orientation is desired, but may include '|' and '-', indicating that the component should be flipped about either the vertical or horizontal axis.

The *name* and *value* are strings that are added to the component as labels, though not all components display the *value*. The *nudge* is a number that adjusts the placement of labels to avoid wires.

In addition, some components support other arguments, such as `kind` or `loc`.

You may pass wires directly under most components. The component will conceal the wire in those places where it should not be shown. This makes it simpler to wire up a schematic as you don't need separate wires between a string of components that all fall in a line. Rather, you would just specify the wire first, and then it will run underneath the components. This trick works as long as you do not specify the schematic background as 'none'.

Components generally place the location of their principle coordinates and the location of all their pins into named attributes.

Resistor

Draw a resistor.

```
Resistor(name='Rs', value='50')
```

Resistors take the following arguments: `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `p`, `n`, `off`, `xoff` and `yoff`.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `p` and `n` attributes contain the locations of the positive and negative terminals.

You may pass a wire directly under the resistor and the wire will be concealed by the resistor.

Capacitor

Draws a capacitor.

```
Capacitor(name='C1', value='1.2pF')
```

Capacitors take the following arguments: `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `p`, `n`, `off`, `xoff` and `yoff`.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `p` and `n` attributes contain the locations of the positive and negative terminals.

You may pass a wire directly under the capacitor and the wire will be concealed by the capacitor. The capacitor is polarized with reference end being terminal `n`.

Inductor

Draws an inductor.

```
Inductor(name='L1', value='1H')
```

Inductors take the following arguments: `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `p`, `n`, `off`, `xoff` and `yoff`.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `p` and `n` attributes contain the locations of the positive and negative terminals.

You may pass a wire directly under the inductor and the wire will be concealed by the inductor.

Diode

Draws a diode.

```
Diode(name='D1')
```

Diodes take the following arguments: `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `p`, `n`, `off`, `xoff` and `yoff`.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `a` and `c` attributes contain the coordinates of the anode and cathode terminals.

You may pass a wire directly under the diode and the wire will be concealed by the diode.

BJT

Draws a bipolar transistor. Two kinds of BJT are available, *npn* and *pnp*.

```
MOS(kind='n', name='Qn')
MOS(kind='p', name='Qp')
```

BJTs take the following arguments: `kind`, `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `p`, `n`, `off`, `xoff` and `yoff`. `kind` may be `npn` or `pnp`, or simply `n` or `p`.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `c`, `b` and `e` attributes contain the coordinates of the collector, base and emitter terminals.

If `kind` is 'p' or 'pnp' a PNP symbol is drawn, otherwise an NPN symbol is drawn.

You may pass a wire directly under the transistor and the wire will be concealed by the transistor.

MOS

Draws a MOSFET. Three kinds of FET are available, *nmos*, *pmos*, and non-polarized.

```
MOS(kind='n', name='Mn')
MOS(kind='p', name='Mp')
MOS(kind='', name='M')
```

MOSFETs take the following arguments: `kind`, `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `p`, `n`, `off`, `xoff` and `yoff`. `kind` may be `nmos` or `pmos`, or simply `n` or `p`. If an empty string is specified, the terminal locations are those of an *nmos*, but the arrow is not drawn.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `d`, `g` and `s` attributes contain the coordinates of the drain, gate and source terminals.

If `kind` is 'n' or 'nmos' an NMOS symbol is drawn; if `kind` is 'p' or 'pmos' a PMOS symbol is drawn; otherwise an unpolarized symbol is drawn.

You may pass a wire directly under the transistor and the wire will be concealed by the transistor.

Amplifiers and Converters

Draws an amplifier or a converter. Four kinds are available, single-ended (`se`), opamp (`oa`), differential amplifier (`da`) and comparator (`comp`).

```
Amp(kind='se', name='As')
Amp(kind='oa', name='Ao')
Amp(kind='da', name='Ad')
Amp(kind='comp', name='Ac')
Converter(kind='se', name='Cs')
Converter(kind='oa', name='Co')
Converter(kind='da', name='Cd')
Converter(kind='comp', name='Cc')
```

Amplifiers and Converters take the following arguments: kind, orient, name, value, C, N, NE, E, SE, S, SW, W, NW, p, n, off, xoff and yoff. kind may be se, oa, da or comp.

The C, N, NE, E, SE, S, SW, W, NW attributes contain the locations of the principle coordinates. The pi, i, ni po, o, and no attributes contain the coordinates of the positive input, the input, the negative input, the positive output, the output, and the negative output terminals. All 6 pin attributes are always available, even if they do not seem appropriate for the kind of amplifier drawn.

You can reshape the amplifier or converter using w and h to specify the width and height. The default values for each are 2, and you should not deviate too far from 2 or you will end up with an ugly symbol.

You may pass a wire or wires directly under the amplifier or converter and the wire will be concealed.

Gate

Draws a gate. Currently the only supported kind of gate is inv, an inverter.

```
Gate(kind='inv', name='U')
```

Gates take the following arguments: kind, orient, name, value, nudge, C, N, NE, E, SE, S, SW, W, NW, p, n, off, xoff and yoff. kind may be inv.

The C, N, NE, E, SE, S, SW, W, NW attributes contain the locations of the principle coordinates. The i and o attributes contain the coordinates of the input and the output.

You may pass a wire or wires directly under the gate and the wire will be concealed by the gate.

Source

Draws a source. Eight kinds of source are available, empty, vdc, idc, sine, sum (summer), mult (multiplier), cv (controlled voltage) and ci (controlled current).

```
Source(kind='empty', name='Ve')
Source(kind='vdc', name='Vd')
Source(kind='idc', name='Id')
Source(kind='sine', name='Vs')
Source(kind='sum', name='S')
Source(kind='mult', name='M')
Source(kind='cv', name='Vc')
Source(kind='ci', name='Ic')
```

Sources take the following arguments: kind, orient, name, value, nudge, C, N, NE, E, SE, S, SW, W, NW, p, n, off, xoff and yoff. kind may be empty, vdc, idc, sine, sum, mult, cv or ci.

The C, N, NE, E, SE, S, SW, W, NW attributes contain the locations of the principle coordinates, but unlike all other components, these are evenly distributed about the circle that envelopes the source. The p and n attributes contain the coordinates of the positive and negative pins. The pin attributes are always available, even if they do not seem appropriate for the kind of source drawn.

You may pass a wire or wires directly under the source and the wire will be concealed by the source.

Switch

Draws a switch. Two kinds of switch are available, `spst` (single-pole, single-throw) and `spdt` (single-pole, double-throw).

```
Switch(kind='spst', name='₁')
Switch(kind='spdt', name='₂')
```

Switches take the following arguments: `kind`, `orient`, `name`, `value`, `dots`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `i`, `o`, `ot`, `ob`, `off`, `xoff` and `yoff`. `kind` may be `spst` or `spdt`. The *dots* argument determines whether the poles of the switch should be denoted with large dots.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `i` `ot`, `o` and `ob` attributes contain the coordinates of the input, the top output, the output, and the bottom output pins. The pin attributes are always available, even if they do not seem appropriate for the kind of switch drawn.

You may pass a wire or wires directly under the switch and the wire will be concealed by the switch.

Box

Draws a box.

```
Box(name='4 bit', value='Flash')
Box(name='¹', w=1, h=1)
```

Boxes take the following arguments: `orient`, `name`, `value`, `nudge`, `line_width`, `background`, `w`, `h`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `i`, `pi`, `ni`, `o`, `po`, `no`, `off`, `xoff` and `yoff`. In addition, you may specify *SVGwrite* arguments, as shown in the example below.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. They are arranged along the surface of the box.

The `i` `pi`, `ni` and `o`, `po`, `no` attributes contain the coordinates of the input and output pins.

Boxes also support arbitrary *svgwrite* drawing parameters. This can be useful to draw the box with dashed lines:

```
Box(w=1, h=1, stroke_dasharray="4 2")
```

Crossing

Draws a wire crossing in such a was as to maintain symmetry in schematics.

```
Crossing()
Crossing(w=2, h=2)
```

Crossings take the following arguments: , `pass_under`, `w`, `h`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `pi`, `ni`, `po`, `no`, `off`, `xoff` and `yoff`.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `pi` and `pi` attributes contain the coordinates of the input and output pins.

Ground

Draws a ground.

```
Ground()
```

Grounds take the following arguments: `orient`, `name`, `value`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `t`, `off`, `xoff` and `yoff`. Currently `value` is ignored.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `t` attribute contains the coordinates of the ground's terminal.

Pin

Draws a pin. Four kinds of pin are available, `none`, `dot`, `in`, and `out`.

```
Pin(kind='none', name='none', value='none value')
Pin(kind='dot', name='dot', value='dot value')
Pin(kind='in', name='in')
Pin(kind='out', name='out')
```

Here the pins are drawn with wires to give better context. The horizontal location of the pins is indicated with the vertical blue line.

Pins take the following arguments: `kind`, `orient`, `name`, `value`, `nudge`, `w`, `h`, `color`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `t`, `off`, `xoff` and `yoff`. Currently `value` is ignored for `in` and `out` pins.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `t` attribute contains the coordinates of the pin.

It is important to remember that C represents the center of the tile used by the pin. Since the pin label will be on one side, C will not coincide with the apparent visual center of the pin and its label.

Pins of kind `none` do not draw a symbol. Rather they are used to place labels at a particular point. `dot` pins place a small filled circle that is usually used to represent a solder dot (though you can change the color to the background color, generally 'white', and place it between two crossing wires to create a visual gap in the lower wire). Pins of type `in` and `out` render with a hollow circle that is offset slightly a wire terminates on one side. These two pin types ignore the `value` argument.

By default the width and height of the pin are 1, meaning that a unit sized tile (50×50) is used. This is significant if the pin is at the edge of the schematic. If the labels extend beyond the tile, they may extend beyond the computed viewbox for the schematic. You can fix this by specifying a larger width.

Dot

Draw a solder dot (a small filled circle) or a wire gap (a small filled circle with the color of the background that is placed between two crossing wires). Dot is just an alias for Pin, except that the default kind is 'dot'.

It is common to place a dot at a level between two crossing wires and specify a color of white to create a pass-under.

```
Dot()
```

Label

Place a label. Five kinds of label are available, `plain`, `arrow`, `arrow|`, `slash`, and `dot`.

```
Label(kind='plain', name='plain', loc='se')
Label(kind='arrow', name='arrow', loc='se')
Label(kind='arrow|', name='arrow|', loc='se')
Label(kind='slash', name='slash', loc='se')
Label(kind='dot', name='dot', loc='se')
```

Here the labels are drawn with wires to give better context. The horizontal location of the labels is indicated with the vertical blue line.

Labels take the following arguments: `kind`, `orient`, `name`, `value`, `loc`, `w`, `h`, `color`, `nudge`, `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW`, `off`, `xoff` and `yoff`. Currently `value` is ignored.

The `C`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `NW` attributes contain the locations of the principle coordinates. The `t` attribute contains the coordinates of the label.

The kind may be 'plain', 'arrow', 'arrow|', 'slash' or 'dot'. If 'plain' is specified, no symbol is added, only the name is displayed. If 'arrow' is specified, an arrow is added and the centered on the specified location. If 'arrow|' is specified, the arrow terminates on the specified location. If 'slash' is specified, a small slash is added through the center. It is generally used with buses to indicate the bus width. Finally, 'dot' adds a solder dot.

By default the width and height of the label are 1, meaning that a unit sized tile (50×50) is used. This is significant if the label is at the edge of the schematic. If the labels extend beyond the tile, they may extend beyond the computed viewbox for the schematic. You can fix this by specifying a larger width.

It is important to remember that C represents the center of the tile used by the label. Since the label will be on one side, C will not coincide with the apparent visual center of the label.

5.2.4 Location Functions

shift

Shifts a point by specified amounts in both the x and y directions.

`shift`(*point*, *dx*, *dy*)

point is an (x, y) coordinate and dx and dy are numbers. The return value is $(x + dx, y + dy)$.

shift_x

Shifts a point by specified amount in the x direction.

`shift_x`(*point*, *dx*)

point is an (x, y) coordinate and dx is a number. The return value is $(x + dx, y)$.

shift_y

Shifts a point by specified amount in the *y* direction.

shift_y(*point*, *dy*)

point is an (*x*, *y*) coordinate and *dy* is a number. The return value is (*x*, *y* + *dy*).

with_x

Returns the first argument (a coordinate pair) with the *x* value replaced with the second argument. The second argument may either be a number or a coordinate pair.

with_x(*point*, *x*)

with_y

Returns the first argument (a coordinate pair) with the *y* value replaced with the second argument. The second argument may either be a number or a coordinate pair.

with_y(*point*, *y*)

with_min_x

Returns the first argument (a coordinate pair) with the *x* value replaced with the smallest of the remaining arguments. The remaining arguments may either be numbers or a coordinate pairs.

with_min_x(*point*, ...)

with_max_x

Returns the first argument (a coordinate pair) with the *x* value replaced with the largest of the remaining arguments. The remaining arguments may either be numbers or a coordinate pairs.

with_max_x(*point*, ...)

with_min_y

Returns the first argument (a coordinate pair) with the *y* value replaced with the smallest of the remaining arguments. The remaining arguments may either be numbers or a coordinate pairs.

with_min_y(*point*, ...)

with_max_y

Returns the first argument (a coordinate pair) with the *y* value replaced with the largest of the remaining arguments. The remaining arguments may either be numbers or a coordinate pairs.

with_max_y(*point*, ...)

midpoint

Returns the point midway between two points.

midpoint(*point1*, *point2*)

midpoint_x

Returns the point with *x* value midway between two points and the *y* value of the first point.

midpoint_x(*point1*, *point2*)

midpoint_y

Returns the point with *y* value midway between two points and the *x* value of the first point.

midpoint_y(*point1*, *point2*)

5.2.5 Exceptions

In the rare cases where *SVG_Schematic* it raises an error, it uses Inform Error. *SVG_Schematic* is a wrapper around svgwrite. It is not clear what exceptions it will raise, but at a minimum it would raise *OSError* if it is unable to open or close the SVG file. Thus you should catch these two exceptions. See *Non Inverting Amplifier* to see how this is done.

5.3 Examples

5.3.1 Non Inverting Amplifier

Here is an example of a typical schematic with exception handling.

```python
#!/usr/bin/env python3

from svg_schematic import (
    Schematic, Amp, Dot, Ground, Label, Pin, Resistor, Source, Wire
)
from inform import Error, error, os_error

try:
    with Schematic(filename = "noninverting.svg", line_width=2):

        vin = Source(kind='sine')
        Label(C=vin.p, name='Vin', loc='n')
```

(continues on next page)

(continued from previous page)

```
        Ground(C=vin.n)
        amp = Amp(pi=vin.p, xoff=100, kind='oa')
        Label(C=amp.ni, xoff=-25, name='Vf', loc='n')
        Wire([vin.p, amp.pi])
        out = Pin(C=amp.o, xoff=50, name='out', w=2)
        Wire([amp.o, out.C])
        oj = Dot(C=amp.o, xoff=25)
        r1 = Resistor(p=amp.ni, off=(-25, 50), name='R1', orient='v')
        Wire([r1.N, amp.ni], kind='|-')
        r2 = Resistor(C=amp.C, yoff=75, name='R2')
        Wire([r1.p, r2.W], kind='|-')
        Wire([oj.C, r2.E], kind='|-')
        fj = Dot(C=r2.W, xoff=-25)
        Ground(C=r1.n)

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.2 Inverting Amplifier

This schematic uses `line_width = 1` give the schematic a lighter look. It uses a 16 point serif font.

```
#!/usr/bin/env python3

from svg_schematic import (
    Schematic, Amp, Dot, Ground, Label, Pin, Resistor, Source, Wire
)
from inform import Error, error, os_error

try:
    with Schematic(
        filename = "inverting.svg",
        font_size=16,
        font_family='serif'
    ):

        vin = Pin(kind='in', name='in', w=1.5)
        vout = Pin(C=vin.C, xoff=350, kind='out', name='out', w=2)
        Wire([vin.C, vout.C])
        rin = Resistor(W=vin.C, xoff=25, name='Rin')
        vg = Dot(C=rin.E, xoff=25)
        rfb = Resistor(W=vg.C, xoff=25, name='Rfb')
        oj = Dot(C=rfb.E, xoff=25)
        amp = Amp(C=rfb.C, yoff=75, orient='-', kind='oa')
        Wire([oj.C, amp.o], kind='|-')
        gnd = Ground(C=amp.pi, xoff=-25, orient='h|')
        Wire([gnd.C, amp.pi])
```

(continues on next page)

(continued from previous page)

```
        Wire([vg.C, amp.ni], kind='|-')
        Label(C=vg.C, name='Vg', loc='sw')

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.3 Charge Pump

This example has a transparent background and so all wires terminate at terminals rather than passing underneath components. The labels are intended to be rendered by *Latex*. It sets `line_width` to 2 give the schematic a heavier look.

```
#!/usr/bin/env python3

from svg_schematic import Schematic, Capacitor, Diode, Ground, Pin, Wire, Dot, with_x
from inform import Error, error, os_error

try:
    with Schematic(
        filename = "charge-pump.svg", line_width=2, background='none'):

        vin = Pin(kind='in', name=r'$V_{\rm in}$', w=2)
        p1 = Pin(C=vin.C, yoff=150, kind='in', name=r'$\phi_1$', w=2)
        p2 = Pin(C=p1.C, yoff=50, kind='in', name=r'$\phi_2$', w=2)
        d1 = Diode(a=vin.C, xoff=25, orient='h')
        c1 = Capacitor(p=d1.c, off=(25, 25), orient='v')
        d2 = Diode(a=d1.c, xoff=50, orient='h')
        c2 = Capacitor(p=d2.c, off=(25, 25), orient='v')
        d3 = Diode(a=d2.c, xoff=50, orient='h')
        c3 = Capacitor(p=d3.c, off=(25, 25), orient='v')
        d4 = Diode(a=d3.c, xoff=50, orient='h')
        c4 = Capacitor(p=d4.c, off=(25, 25), orient='v')
        d5 = Diode(a=d4.c, xoff=50, orient='h')
        c5 = Capacitor(p=d5.c, off=(25, 25), orient='v')
        vout = Pin(C=d5.c, xoff=75, kind='out', name=r'$V_{\rm out}$', w=2)
        Ground(t=c5.n)

        Wire([vin.t, d1.a])
        Wire([d1.c, d2.a])
        Wire([d2.c, d3.a])
        Wire([d3.c, d4.a])
        Wire([d4.c, d5.a])
        Wire([d5.c, vout.t])
        Wire([with_x(vin.t, c1.C), c1.p])
        Wire([with_x(vin.t, c2.C), c2.p])
        Wire([with_x(vin.t, c3.C), c3.p])
        Wire([with_x(vin.t, c4.C), c4.p])
```

(continues on next page)

(continued from previous page)

```
        Wire([with_x(vin.t, c5.C), c5.p])
        Wire([p1.t, c1.n], kind='-|')
        Wire([p2.t, c2.n], kind='-|')
        co = Dot(C=with_x(p1.C, c2.C), color='white')     # wire cross over
        Wire([p1.t, c3.n], kind='-|')
        Wire([p2.t, c4.n], kind='-|')

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.4 Inverter

This example has a transparent background and so all wires terminate at terminals rather than passing underneath components. The labels are intended to be rendered by *Latex*. It sets `line_width` to 2 give the schematic a heavier look.

```
#!/usr/bin/env python3

from svg_schematic import Schematic, MOS, Ground, Gate, Label, Pin, Wire, midpoint
from inform import Error, error, os_error

try:
    with Schematic(filename = 'inverter.svg', line_width=2, background='none'):

        # transistor version
        mp = MOS(kind='p')
        mn = MOS(N=mp.S, kind='n')
        vin = Pin(C=midpoint(mp.g, mn.g), xoff=-50, kind='in', name=r'$V_{\rm in}$', w=2)
        vout = Pin(C=midpoint(mp.d, mn.d), xoff=50, kind='out', name=r'$V_{\rm out}$',␣
↪w=2)
        Label(C=mp.s, loc='n', name=r'$V_{\rm dd}$')
        Ground(C=mn.s)
        Wire([vin.t, mp.g], kind='-|')
        Wire([vin.t, mn.g], kind='-|')
        Wire([vout.t, mp.d], kind='-|')
        Wire([vout.t, mn.d], kind='-|')

        # gate version
        inv = Gate(N=mn.S, yoff=25, kind='inv')
        vin = Pin(t=inv.i, xoff=-25, kind='in', name=r'$V_{\rm in}$', w=2)
        vout = Pin(t=inv.o, xoff=25, kind='out', name=r'$V_{\rm out}$', w=2)
        Wire([inv.o, vout.t])
        Wire([inv.i, vin.t])

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.5 Oscillator

This example has a transparent background and so all wires terminate at terminals rather than passing underneath components. The labels are intended to be rendered by *Latex*. This schematic uses `line_width = 2` give the schematic a heavier look.

```
#!/usr/bin/env python3

from svg_schematic import (
    Schematic, Capacitor, MOS, Inductor, Label, Source, Wire, Crossing,
    midpoint, shift, shift_y,
)
from inform import Error, error, os_error

try:
    with Schematic(filename = "oscillator.svg", background='none', line_width=2):

        # resonator
        vdd = Label(loc='n', nudge=10, name=r'$V_{\rm dd}$')
        Wire([vdd.C, shift_y(vdd.C, 25)])
        ll = Inductor(p=shift(vdd.C, -125, 25), orient='v', name=r'$\frac{1}{2} L$')
        lr = Inductor(p=shift(vdd.C, 125, 25), orient='v|', name=r'$\frac{1}{2} L$')
        c = Capacitor(C=midpoint(ll.n, lr.n), orient='h', name='$C$')
        Wire([ll.p, lr.p])
        Wire([ll.n, c.p])
        Wire([lr.n, c.n])

        # gain stage
        ml = MOS(d=ll.n, yoff=75, orient='|')
        mr = MOS(d=lr.n, yoff=75, orient='')
        Wire([ll.n, ml.d])
        Wire([lr.n, mr.d])
        cross = Crossing(C=midpoint(ml.g, mr.g), yoff=-50, orient='v', pass_under='white
↪')
        Wire([ml.g, cross.pi], kind='-|')
        Wire([mr.g, cross.ni], kind='-|')
        Wire([lr.n, cross.po], kind='|-')
        Wire([ll.n, cross.no], kind='|-')
        Wire([ml.s, shift_y(ml.s, 12), shift_y(mr.s, 12), mr.s])
        Source(p=midpoint(ml.s, mr.s), yoff=12, kind='idc', value=r'$I_{\rm ss}$')

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.6 Passive Low Pass Filter

This example uses QuantiPhy to compute the values for the components in a low pass filter and then constructs the schematic using those values. It sets `line_width` to 2 and employs dots at wire junctions to give the schematic a heavier look.

```
"""
Draw a 5th Order Low Pass Passive Filer with Maximally Flat Envelope Delay

Use the following parameters:
    Fo = 1MHz    - 3dB corner frequency
    Rref = 50  - termination impedance

Design equations:
    0 = 2**Fo
    Lscale = Rref/0
    Cscale = 1/(Rref*0)

    Rs = 1.0000 * Rref   ""
    C1 = 0.2715 * Cscale "F"
    L2 = 0.6541 * Lscale "H"
    C3 = 0.8892 * Cscale "F"
    L4 = 1.1034 * Lscale "H"
    C5 = 2.2873 * Cscale "F"
"""

try:
    from svg_schematic import (
        Schematic, Capacitor, Dot, Ground, Inductor, Label, Resistor, Pin,
        Source, Wire
    )
    from inform import Error, error, os_error
    from quantiphy import Quantity
    from math import pi
except ImportError:
    print(
        'Run `pip install --user -r requirements.txt`',
        'to install missing packages.'
    )
    raise SystemExit

Quantity.set_prefs(map_sf=Quantity.map_sf_to_greek, prec=2)
globals().update(
    Quantity.extract(__doc__, predefined={'': pi})
)

try:
    with Schematic(filename = 'mfed.svg', line_width=2, background = 'none'):

        vin = Source(kind='sine')
        Ground(C=vin.n)
        rs = Resistor(name='Rs', value=Rref, n=vin.p, xoff=25)
        Wire([vin.p, rs.n])
```

(continues on next page)

(continued from previous page)

```
        c1 = Capacitor(name='C1', value=C1, p=rs.p, xoff=25)
        Ground(C=c1.n)
        l2 = Inductor(name='L2', value=L2, n=c1.p, xoff=25)
        Wire([rs.p, l2.n])
        c3 = Capacitor(name='C3', value=C3, p=l2.p, xoff=25)
        Ground(C=c3.n)
        l4 = Inductor(name='L4', value=L4, n=c3.p, xoff=25)
        Wire([l2.p, l4.n])
        c5 = Capacitor(name='C5', value=C5, p=l4.p, xoff=25)
        Ground(C=c5.n)
        rl = Resistor(name='Rl', value=Rref, p=c5.p, xoff=100, orient='v')
        Ground(C=rl.n)
        out = Pin(name='out', C=rl.p, xoff=50, w=2)
        Wire([l4.p, out.t])
        Label(S=c3.N, yoff=-50, name=f'{Fo} LPF', loc='s')
        Dot(C=c1.p)
        Dot(C=c3.p)
        Dot(C=c5.p)
        Dot(C=rl.p)

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.7 Buck Regulator

This example uses a white dot as a pass-under (on the line that runs from ramp generator to the comparator.

```
#!/usr/bin/env python3

from svg_schematic import (
    Schematic, Amp, Box, Capacitor, Converter, Dot, Ground, Inductor, Label, MOS,
    Pin, Resistor, Switch, Wire, shift, shift_x, shift_y, with_x, with_y, midpoint
)
from inform import Error, error, os_error

try:
    with Schematic(
        filename = "buck.svg", line_width=2, background='none'):

        pvdd = Pin(kind='in', name='pvdd', w=3)
        avdd = Pin(C=pvdd.C, yoff=50, kind='in', name='avdd', w=3)
        Wire([avdd.C, shift_x(avdd.C, 50)])

        lvl = Pin(C=avdd.C, yoff=90, kind='in', name='lvl', w=3)
        reference = Converter(i=lvl.C, xoff=75, name='ref')
        lvl2ref = Wire([lvl.C, reference.i])
```

(continues on next page)

(continued from previous page)

```
        Label(C=lvl2ref.m, kind='slash', name='6', loc='se')

        amp = Amp(pi=reference.o, xoff=50, kind='oa', name='amp')
        Wire([reference.o, amp.pi])
        C1 = Capacitor(p=amp.C, off=(-12.5, 75), orient='h')
        R1 = Resistor(p=C1.p, xoff=12.5, orient='h')
        C2 = Capacitor(C=midpoint(R1.C, C1.C), yoff=50, orient='h')
        Wire([C1.n, with_y(C1.n, amp.o)], kind='-|')
        Wire([R1.n, amp.ni], kind='|-')
        Wire([C2.p, R1.n], kind='-|')
        Wire([C2.n, C1.n], kind='-|')

        cmp = Amp(pi=amp.o, xoff=125, kind='comp', name='cmp')
        Wire([amp.o, cmp.pi])

        gd = Box(i=cmp.o, xoff=50, name='gate', value='drivers')
        Wire([cmp.o, gd.i])

        pfet = MOS(g=gd.N, yoff=-50, kind='p', orient='h')
        Wire([pfet.g, gd.N])
        Wire([pfet.s, pvdd.C])

        nfet = MOS(g=gd.o, xoff=25, kind='n', orient='v')
        Wire([nfet.g, gd.o])
        Wire([nfet.d, pfet.d], kind='|-')
        Ground(C=nfet.s)

        ind = Inductor(n=with_x(pfet.d, nfet.E), orient='h')
        Wire([pfet.d, ind.n])
        cap = Capacitor(p=ind.p, orient='v')
        Ground(C=cap.n)
        out = Pin(C=ind.p, xoff=100, kind='out', name='out', w=3)
        out_wire = Wire([ind.p, out.C])

        fb = shift_y(R1.n, 100)

        Rt = Resistor(n=with_x(fb, out_wire.m), orient='v')
        Rb = Resistor(p=with_x(fb, out_wire.m), orient='v')
        Wire([Rt.p, with_y(Rt.p, out.C)])
        Ground(C=Rb.n)

        RG = Box(o=C1.n, yoff=175)
        rg2cmp = Wire([RG.o, cmp.ni], kind='-|-')
        Dot(C=with_y(rg2cmp.m, fb), color='white')
        Wire([shift_x(RG.C, -30), shift(RG.C, -25, 25), shift(RG.C, 25, -25), shift_x(RG.
↪C, 30)])
        Label(C=RG.S, loc='s', name='ramp_gen')
        Wire([R1.n, fb, Rt.n], kind='|-')

        en = Pin(C=lvl.C, yoff=250, kind='in', name='en', w=3)
        Wire([en.C, shift_x(en.C, 50)])
        avss = Pin(C=en.C, yoff=50, kind='in', name='avss', w=3)
```

(continues on next page)

(continued from previous page)

```
        Wire([avss.C, shift_x(avss.C, 50)])

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.8 Pipelined ADC

This block diagram has a white background and so could route wires under components rather than wiring to terminals, but it largely does not. It uses `line_width = 2` give the diagram a heavier look.

```
#!/usr/bin/env python3

from svg_schematic import (
    Schematic, Amp, Box, Label, Pin, Source, Wire,
    midpoint, shift_x, shift, with_x, with_y,
)
from inform import Error, error, os_error

try:
    with Schematic(filename = 'pipeline-adc.svg', line_width=2):

        # Stage 1
        i = Pin(kind='in', name='in')
        s1 = Box(NW=i.t, off=(25,-62.5), w=10.5, h=4.5, background='lightgray')
        Label(C=s1.SE, loc='nw', name='Stage 1')
        adc = Box(W=i.t, off=(75,100), name='2 bit', value='Flash')
        dac = Box(i=adc.o, xoff=50, name='2 bit', value='DAC')
        sh = Box(C=with_x(i.t, midpoint(adc.C, dac.C)), name='SAH')
        sum = Source(W=with_x(i.t, dac.E), xoff=25, kind='sum', orient='h|')
        Label(C=sum.W, loc='nw', name='+')
        Label(C=sum.S, loc='se', name='')
        amp = Amp(i=sum.E, xoff=25, kind='se', name='4×')
        Wire([i.t, sh.i])
        Wire([sh.o, sum.W])
        Wire([sum.E, amp.i])
        Wire([shift_x(i.t, 50), adc.i], kind='|-')
        Wire([adc.o, dac.i])
        Wire([dac.o, sum.S], kind='-|')

        # Stages 2, 3, 4
        s2 = Box(N=dac.S, off=(25,75), name='Stage 2')
        s3 = Box(W=s2.E, xoff=50, name='Stage 3')
        s4 = Box(W=s3.E, xoff=50, name='4 bit', value='Flash')
        Wire([s2.o, s3.i])
        Wire([s3.o, s4.i])
        Wire([
            amp.o,
```

(continues on next page)

(continued from previous page)

```
            shift_x(amp.o, 50),
            shift(s1.SE, 25,25),
            shift(s2.NW, -25, -25),
            shift_x(s2.W, -25),
            s2.W,
        ])

        # Error correction
        ec = Box(NW=s2.SW, off=(-75, 50), name='Digital Error Correction', w=9, h=1)
        out = Pin(t=shift_x(ec.o, 50), kind='out', name='out', w=2)
        Wire([ec.o, out.t])
        Label(C=shift_x(ec.E, 25), kind='slash', loc='s', name='8')

        w1 = Wire([midpoint(adc.o, dac.i), with_y(midpoint(adc.o, dac.i), ec.N)])
        w2 = Wire([s2.S, with_y(s2.S, ec.N)])
        w3 = Wire([s3.S, with_y(s3.S, ec.N)])
        w4 = Wire([s4.S, with_y(s4.S, ec.N)])
        Label(C=w1.e, yoff=-25, kind='slash', loc='w', name='2', nudge=8)
        Label(C=w2.e, yoff=-25, kind='slash', loc='w', name='2', nudge=8)
        Label(C=w3.e, yoff=-25, kind='slash', loc='w', name='2', nudge=8)
        Label(C=w4.e, yoff=-25, kind='slash', loc='w', name='4', nudge=8)

except Error as e:
    e.report()
except OSError as e:
    error(os_error(e))
```

5.3.9 Receiver

This block diagram has a white background and so could route the wires underneath the components, but does not. It uses `line_width = 2` give the diagram a heavier look. It looks small because it is quite wide, and it is scaled to fit the page.

```
#!/usr/bin/env python3

from svg_schematic import (
    Schematic, shift_x, shift_y, Amp, Box, Ground, Label, Pin, Source, Wire
)

with Schematic(
    filename = 'receiver.svg',
    font_size = 12,
    font_family = 'sans',
    outline = 'none',
    pad = 20,
    line_width = 2,
) as schematic:

    # Input from horn antenna
```

(continues on next page)

(continued from previous page)

```
    rf_in = Pin(kind='in', name='L-band Horn', w=3)
    Label(C=rf_in.C, yoff=15, loc='w', name='fc=1420MHz', w=3.5, nudge=14)
    Label(C=rf_in.C, yoff=30, loc='w', name='BW=15MHz', w=3.5, nudge=14)

    # First preamp
    rf_preamp1 = Amp(i=rf_in.t, xoff=25, kind='se', name='RF Preamp1')
    Label(C=rf_preamp1.S, loc='s', name='A>=26dB')
    Wire([rf_in.t, rf_preamp1.i])

    # Second preamp
    rf_preamp2 = Amp(i=rf_preamp1.o, xoff=25, kind='se', name='RF Preamp2')
    Label(C=rf_preamp2.S, loc='s', name='A>=26dB')
    Wire([rf_preamp1.o, rf_preamp2.i])

    # RF band-pass filter
    rf_bpf = Box(i=rf_preamp2.o, xoff=25, name='RF BPF')
    l = Label(C=rf_bpf.S, loc='s', name='fc=1380MHz')
    Label(C=l.S, name='BW=320MHz')
    Wire([rf_preamp2.o, rf_bpf.i])

    # First RF amplifier
    rf_amp1 = Amp(i=rf_bpf.o, xoff=25, kind='se', name='RF Amp1')
    Label(C=rf_amp1.S, loc='s', name='A<=20dB')
    Wire([rf_bpf.o, rf_amp1.i])

    # Second RF amplifier
    rf_amp2 = Amp(i=rf_amp1.o, xoff=25, kind='se', name='RF Amp2')
    Label(C=rf_amp2.S, loc='s', name='A<=20dB')
    Wire([rf_amp1.o, rf_amp2.i])

    # RF mixer
    rf_mixer = Source(W=rf_amp2.o, xoff=25, kind='mult')
    Label(C=rf_mixer.N, loc='n', name='DSB')
    Wire([rf_amp2.o, rf_mixer.W])

    # RF local oscillator
    rf_lo = Source(
        p=rf_mixer.S, yoff=50, kind='sine',
        name='fo=1230MHz', value='P=10dBm'
    )
    Ground(t=rf_lo.n)
    Wire([rf_mixer.S, rf_lo.N])

    # IF band-pass filter
    if_bpf = Box(i=rf_mixer.E, xoff=25, name='IF BPF')
    l= Label(C=if_bpf.S, loc='s', name='fc=190MHz')
    Label(C=l.S, name='BW=22MHz')
    Wire([rf_mixer.E, if_bpf.i])

    # IF amplifier
    if_amp = Amp(i=if_bpf.o, xoff=25, name='IF Amp')
    Label(C=if_amp.S, loc='s', name='A=20dB')
```

(continues on next page)

(continued from previous page)

```
    Wire([if_bpf.o, if_amp.i])

    # IF mixer
    if_mixer = Source(W=if_amp.o, xoff=20, kind='mult')
    Label(C=if_mixer.N, loc='n', name='SSB')
    Wire([if_amp.o, if_mixer.W])

    # IF local oscillator
    if_lo = Source(
        p=if_mixer.S, yoff=50, kind='sine',
        name='fo=190MHz', value='P=10dBm'
    )
    Ground(t=if_lo.n)
    Wire([if_mixer.S, if_lo.p])

    # Baseband low-pass filter
    bb_lpf = Box(i=if_mixer.E, xoff=25, name='BB LPF')
    Label(C=bb_lpf.S, loc='s', name='BW=2MHz (var)')
    Wire([if_mixer.E, bb_lpf.i])

    # Analog-to-digital converter
    adc = Box(i=bb_lpf.o, xoff=25, name='ADC')
    Label(C=adc.S, loc='s', name='Fclk=var')
    Wire([bb_lpf.o, adc.i])

    # Output
    bb_out = Pin(t=adc.o, xoff=50, kind='out', name='out', w=1.5)
    w = Wire([adc.o, bb_out.t])
    Label(C=w.m, kind='slash', loc='s', name='8')
```

5.3.10 Network Map

This is another block diagram.

```
from svg_schematic import Schematic, Box, Wire, Label, shift_x, shift_y


with Schematic(filename='network-map.svg', line_width=2):
    # work network
    work = Box(w=6.5, h=4.5, stroke_dasharray="4 2")
    Label(C=work.SW, loc='ne', name='work')
    bastion = Box(S=work.S, yoff=-25, w=5.5, h=2, color='lightgray')
    Wire([bastion.E, shift_x(bastion.E, 75)])
    Label(C=bastion.SW, loc='ne', name='bastion')
    www = Box(NE=bastion.N, off=(-12.5, 25), w=2, h=1, color='white', name='www')
    mail = Box(NW=bastion.N, off=(12.5, 25), w=2, h=1, color='white', name='mail')
    dump = Box(SW=bastion.NW, yoff=-25, w=2.5, h=1, name='dump')
    laptop = Box(SE=bastion.NE, yoff=-25, w=2.5, h=1, name='my laptop', stroke_dasharray=
↪"2 2")
```

(continues on next page)

(continued from previous page)

```
    # home network
    home = Box(N=work.S, yoff=50, w=6.5, h=2, stroke_dasharray="4 2")
    Label(C=home.SW, loc='ne', name='home')
    laptop = Box(SW=home.SW, off=(25, -25), w=2.5, h=1, color='lightgray', name='my␣
↪laptop', stroke_dasharray="2 2")
    media = Box(SE=home.SE, off=(-25, -25), w=2.5, h=1, name='media')
    Wire([media.E, shift_x(media.E, 75)])

    # internet
    internet = Wire([shift_x(work.NE, 50), shift_x(home.SE, 50)], line_width=4)
    Label(C=internet.e, loc='s', name='internet')

    # external network
    github = Box(NW=internet.b, off=(50, 25), w=3, h=1, name='github')
    Wire([github.W, shift_x(github.W, -50)])
    cloud = Box(N=github.S, yoff=25, w=3, h=1, name='vps')
    Wire([cloud.W, shift_x(cloud.W, -50)])
    backups = Box(N=cloud.S, yoff=25, w=3, h=1, name='backups')
    Wire([backups.W, shift_x(backups.W, -50)])
    hotspot = Box(N=backups.S, yoff=25, w=3, h=2, stroke_dasharray="4 2")
    Label(C=hotspot.SW, loc='ne', name='a wifi hotspot')
    laptop = Box(C=hotspot.C, w=2, h=1, name='my laptop', stroke_dasharray="2 2")
```

5.4 Releases

Latest development release:

Version: 1.2.0
Released: 2022-06-03

- added Crossing symbols
- added Converter symbols.

1.0 (2020-04-16):

- reorganized documentation into a formal manual.

0.7 (2020-03-02):

Moves almost fully to relative placement of components.

- add ability to place a component by specifying location of its pins or principle coordinates.
- add ability to add offsets when placing components.

This version is incompatible with previous versions.

- location of component must be specified with named argument.
- `orientation` arguments have been renamed to `orient`.
- terminal array (`t`) has been removed

You can upgrade previous schematics to this version by:

  - adding `C=` to leading argument of all components.
  - replacing `orientation` with `orient` in the argument lists of all components.
  - replacing use of `t` with the actual names of the pins.

0.6 (2019-09-03):

- genindex

INDEX