

Contents

1 Introduction	2
2 Usage	2
3 Reference	2
3.1 circuit	2
3.1.1 circuit	2
3.2 util	3
3.2.1 lpad	3
3.2.2 opposite-anchor	3
3.2.3 rotate-anchor	3
3.2.4 colors	4
3.3 wire	5
3.3.1 stub	5
3.3.2 wire	6
3.3.3 wire-styles	8
3.4 element	9
3.4.1 elmt	9
3.4.2 alu	11
3.4.3 block	13
3.4.4 extender	15
3.4.5 multiplexer	17

1 Introduction

This package provides a way to make beautiful block circuit diagrams using the CeTZ package.

2 Usage

Simply import `src/lib.typ` and call the `circuit` function:

```
#import "src/lib.typ"
#lib.circuit({
  import lib: *
  ...
})
```

3 Reference

3.1 circuit

- [circuit\(\)](#)

3.1.1 circuit

Draws a block circuit diagram

This function is also available at the package root

Parameters

```
circuit(
  body: none array element,
  length: length ratio
) -> none
```

body `none` or `array` or `element`

A code block in which draw functions have been called

length `length` or `ratio`

Optional base unit

Default: `2em`

3.2 util

- `lpad()`
- `opposite-anchor()`
- `rotate-anchor()`

Variables:

- `colors`

3.2.1 lpad

Pads a string on the left with 0s to the given length

```
#util.lpad("0100", 8)
```

```
00000100
```

Parameters

```
lpad(
  string: str,
  len: int
) -> str
```

string `str`

The string to pad

len `int`

The target length

3.2.2 opposite-anchor

Returns the anchor on the opposite side of the given one

```
#util.opposite-anchor("west")
```

```
east
```

Parameters

```
opposite-anchor(anchor: str) -> str
```

anchor `str`

The input anchor

3.2.3 rotate-anchor

Returns the anchor rotated 90 degrees clockwise relative to the given one

```
#util.rotate-anchor("west")
```

```
north
```

Parameters

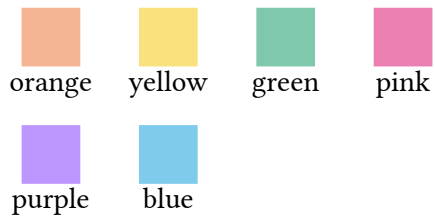
`rotate-anchor`(anchor: `str`) -> `str`

anchor `str`

The anchor to rotate

3.2.4 colors

Predefined color palette



3.3 wire

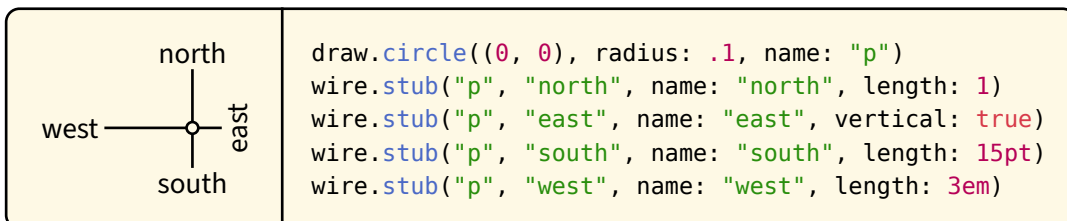
- [stub\(\)](#)
- [wire\(\)](#)

Variables:

- [wire-styles](#)

3.3.1 stub

Draws a wire stub (useful for unlinked ports)



Parameters

```
stub(
  port-id: str,
  side: str,
  name: none or str,
  vertical: bool,
  length: number
)
```

port-id `str`

The port anchor

side `str`

The side on which the port is (one of “north”, “east”, “south”, “west”)

name `none` or `str`

Optional name displayed at the end of the stub

Default: `none`

vertical `bool`

Whether the name should be displayed vertically

Default: `false`

length `number`

The length of the stub

Default: `1em`

3.3.2 wire

Draws a wire between two points

Parameters

```
wire(
  id: str,
  pts: array,
  bus: bool,
  name: none | str | array,
  name-pos: str,
  slice: none | array,
  color: color,
  dashed: bool,
  style: str,
  reverse: bool,
  zigzag-ratio: ratio,
  dodge-y: number,
  dodge-sides: array,
  dodge-margins: array
)
```

id str

The wire's id, for future reference (anchors)

pts array

The two points (as CeTZ compatible coordinates, i.e. XY, relative positions, ids, etc.)

bus bool

Whether the wire is a bus (multiple bits) or a simple signal (single bit)

Default: `false`

name none or str or array

Optional name of the wire. If it is an array, the first name will be put at the start of the wire, and the second at the end

Default: `none`

name-pos str

Position of the name. One of: "middle", "start" or "end"

Default: `"middle"`

slice `none` or `array`

Optional bits slice (start and end bit indices). If set, it will be displayed at the start of the wire

Default: `none`

color `color`

The stroke color

Default: `black`

dashed `bool`

Whether the stroke is dashed or not

Default: `false`

style `str`

The wire's style (see `wire-styles` for possible values)

Default: `"direct"`

reverse `bool`

If true, the start and end points will be swapped (useful in cases where the start point depends on the end point, for example with perpendiculars)

Default: `false`

zigzag-ratio `ratio`

Position of the zigzag vertical relative to the horizontal span (only with style "zigzag")

Default: `50%`

dodge-y `number`

Y position to dodge the wire to (only with style "dodge")

Default: `0`

dodge-sides `array`

The start and end sides (going out of the connected element) of the wire (only with style "dodge")

Default: `("east", "west")`

dodge-margins array

The start and end margins (i.e. space before dodging) of the wire (only with style “dodge”)

Default: (5%, 5%)

3.3.3 wire-styles

List of valid wire styles

The diagram illustrates three wire styles: 'direct' is a straight line connecting two points; 'zigzag' is a line that goes horizontally from the first point, then vertically up to the second point; 'dodge' is a line that goes horizontally from the first point, then vertically down, then horizontally to the second point, and finally vertically up to the second point.

```

for i in range(3) {
  draw.circle((i * 3, 0), radius: .1, name: "p" + str(i * 2))
  draw.circle((i * 3 + 2, 1), radius: .1, name: "p" + str(i * 2 + 1))
  draw.content((i * 3 + 1, -1), raw(wire.wire-styles.at(i)))
}
wire.wire("w1", ("p0", "p1"), style: "direct")
wire.wire("w2", ("p2", "p3"), style: "zigzag")
wire.wire("w3", ("p4", "p5"), style: "dodge",
  dodge-y: -0.5, dodge-margins: (0.5, 0.5))

```


3.4 element

- [elmt\(\)](#)
- [alu\(\)](#)
- [block\(\)](#)
- [extender\(\)](#)
- [multiplexer\(\)](#)

3.4.1 elmt

Draws an element

Parameters

```
elmt(
  draw-shape: function,
  x: number | dictionary,
  y: number | dictionary,
  w: number,
  h: number,
  name: none | str,
  name-anchor: str,
  ports: dictionary,
  ports-margins: dictionary,
  fill: none | color,
  stroke: stroke,
  id: str,
  auto-ports: bool,
  ports-y: array,
  debug: dictionary
)
```

draw-shape `function`

Draw function

Default: default-draw-shape

x `number` or `dictionary`

The x position (bottom-left corner).

If it is a dictionary, it should be in the format `(rel: number, to: str)`, where `rel` is the offset and `to` is the base anchor

Default: `none`

y `number` or `dictionary`

The y position (bottom-left corner).

If it is a dictionary, it should be in the format `(from: str, to: str)`, where `from` is the base anchor and `to` is the id of the port to align with the anchor

Default: `none`

w number

Width of the element

Default: `none`**h** number

Height of the element

Default: `none`**name** `none` or `str`

Optional name of the block

Default: `none`**name-anchor** `str`

Anchor for the optional name

Default: `"center"`**ports** dictionary

Dictionary of ports. The keys are cardinal directions (“north”, “east”, “south” and/or “west”). The values are arrays of ports (dictionaries) with the following fields:

- `id` (`str`): (Required) Port id
- `name` (`str`): Optional name displayed **in** the block
- `clock` (`bool`): Whether it is a clock port (triangle symbol)
- `vertical` (`bool`): Whether the name should be drawn vertically

Default: `()`**ports-margins** dictionary

Dictionary of ports margins (used with automatic port placement). They keys are cardinal directions (“north”, “east”, “south”, “west”). The values are tuples of (,) margins (numbers)

Default: `()`**fill** `none` or `color`

Fill color

Default: `none`

stroke `stroke`

Border stroke

Default: `black + 1pt`**id** `str`

The block id (for future reference)

Default: `""`**auto-ports** `bool`

Whether to use auto port placements or not. If false, draw-shape is responsible for adding the appropriate ports

Default: `true`**ports-y** `array`Array of the ports y offsets (used with `auto-ports: false`)Default: `()`**debug** `dictionary`

Dictionary of debug options.

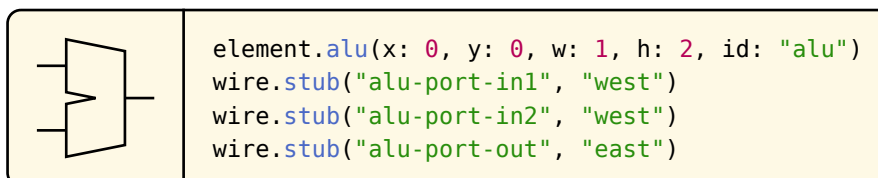
Supported fields include:

- `ports`: if true, shows dots on all ports of the element

Default: `(` `ports: false``)`

3.4.2 alu

Draws an ALU with two inputs



Parameters

```
alu(
  x: number dictionary,
  y: number dictionary,
  w: number,
  h: number,
  name: none str,
  name-anchor: str,
  fill: none color,
  stroke: stroke,
  id: str,
  debug: dictionary
)
```

x number or dictionary

see `elmt()`

Default: `none`

y number or dictionary

see `elmt()`

Default: `none`

w number

see `elmt()`

Default: `none`

h number

see `elmt()`

Default: `none`

name none or str

see `elmt()`

Default: `none`

name-anchor str

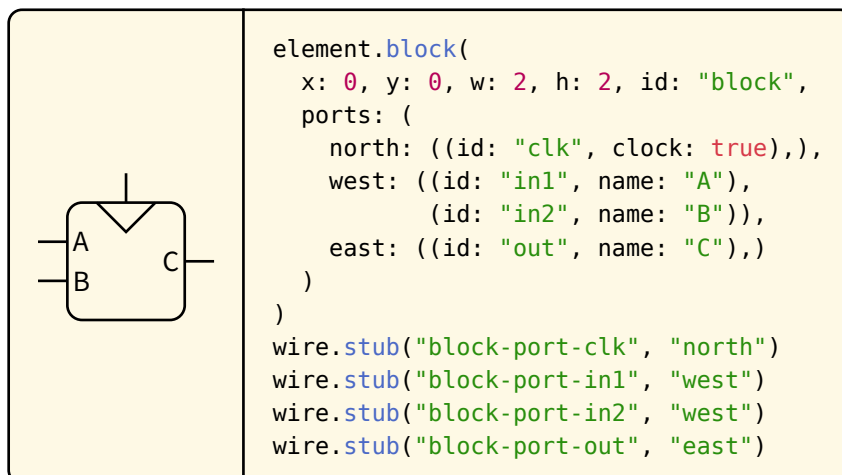
see `elmt()`

Default: `"center"`

fill `none` or `color`see `elmt()`Default: `none`**stroke** `stroke`see `elmt()`Default: `black + 1pt`**id** `str`see `elmt()`Default: `""`**debug** `dictionary`see `elmt()`Default: (
 ports: `false`
)

3.4.3 block

Draws a block element



Parameters

```

block(
  x: number | dictionary,
  y: number | dictionary,
  w: number,
  h: number,
  name: none | str,
  name-anchor: str,
  ports: dictionary,
  ports-margins: dictionary,
  fill: none | color,
  stroke: stroke,
  id: str,
  debug: dictionary
)

```

x number or dictionary

see `elmt()`

Default: `none`

y number or dictionary

see `elmt()`

Default: `none`

w number

see `elmt()`

Default: `none`

h number

see `elmt()`

Default: `none`

name none or str

see `elmt()`

Default: `none`

name-anchor str

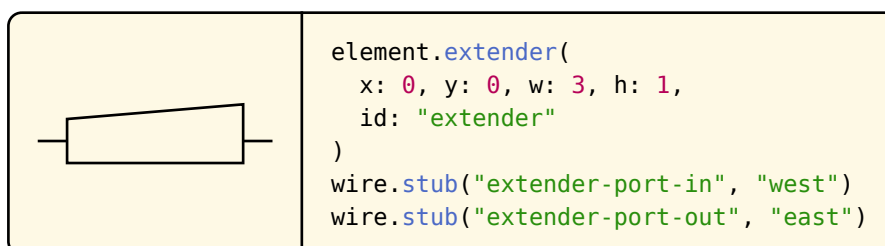
see `elmt()`

Default: `"center"`

ports dictionarysee `elmt()`Default: `()`**ports-margins** dictionarysee `elmt()`Default: `()`**fill** `none` or `color`see `elmt()`Default: `none`**stroke** `stroke`see `elmt()`Default: `black + 1pt`**id** `str`see `elmt()`Default: `""`**debug** dictionarysee `elmt()`Default: `(`
 `ports: false`
`)`

3.4.4 extender

Draws a bit extender



Parameters

```

extender(
  x: number dictionary,
  y: number dictionary,
  w: number,
  h: number,
  name: none str,
  name-anchor: str,
  fill: none color,
  stroke: stroke,
  id: str,
  debug: dictionary
)

```

x number or dictionary

see `elmt()`

Default: `none`

y number or dictionary

see `elmt()`

Default: `none`

w number

see `elmt()`

Default: `none`

h number

see `elmt()`

Default: `none`

name none or str

see `elmt()`

Default: `none`

name-anchor str

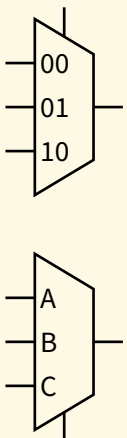
see `elmt()`

Default: `"center"`

fill `none` or `color`see `elmt()`Default: `none`**stroke** `stroke`see `elmt()`Default: `black + 1pt`**id** `str`see `elmt()`Default: `""`**debug** `dictionary`see `elmt()`Default: (
 ports: `false`
)

3.4.5 multiplexer

Draws a multiplexer

	<pre> element.multiplexer(x: 0, y: 0, w: 1, h: 3, id: "multiplexer", entries: 3) wire.stub("multiplexer.north", "north") wire.stub("multiplexer-port-out", "east") element.multiplexer(x: 0, y: -4, w: 1, h: 3, id: "multiplexer2", entries: ("A", "B", "C")) wire.stub("multiplexer2.south", "south") wire.stub("multiplexer2-port-out", "east") for i in range(3) { wire.stub("multiplexer-port-in" + str(i), "west") wire.stub("multiplexer2-port-in" + str(i), "west") } </pre>
---	---

Parameters

```

multiplexer(
  x: number dictionary,
  y: number dictionary,
  w: number,
  h: number,
  name: none str,
  name-anchor: str,
  entries: int array,
  fill: none color,
  stroke: stroke,
  id: str,
  debug: dictionary
)

```

x number or dictionary

see `elmt()`

Default: `none`

y number or dictionary

see `elmt()`

Default: `none`

w number

see `elmt()`

Default: `none`

h number

see `elmt()`

Default: `none`

name none or str

see `elmt()`

Default: `none`

name-anchor str

see `elmt()`

Default: `"center"`

entries `int` or `array`

If it is an integer, it defines the number of input ports (automatically named with their binary index). If it is an array of string, it defines the name of each input.

Default: `2`

fill `none` or `color`

see `elmt()`

Default: `none`

stroke `stroke`

see `elmt()`

Default: `black + 1pt`

id `str`

see `elmt()`

Default: `""`

debug `dictionary`

see `elmt()`

Default: (
 ports: `false`
)