

Contents

1	Introduction	2
2	Usage	2
2.1	Project installation	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	intersection	5
3.3.2	stub	5
3.3.3	wire	6
3.3.4	wire-styles	9
3.4	element	11
3.4.1	elmt	11
3.4.2	alu	13
3.4.3	block	14
3.4.4	extender	15
3.4.5	multiplexer	15
3.4.6	group	17
3.5	gates	19
3.5.1	gate	19
3.5.2	gate-and	21
3.5.3	gate-nand	21
3.5.4	gate-buf	22
3.5.5	gate-not	23
3.5.6	gate-or	23
3.5.7	gate-nor	24
3.5.8	gate-xor	24
3.5.9	gate-xnor	25
3.5.10	iec-gate	26
3.5.11	iec-gate-and	28
3.5.12	iec-gate-nand	28
3.5.13	iec-gate-buf	29
3.5.14	iec-gate-not	29
3.5.15	iec-gate-or	30
3.5.16	iec-gate-nor	31
3.5.17	iec-gate-xor	31
3.5.18	iec-gate-xnor	32

1 Introduction

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

2 Usage

Simply import Circuiteria and call the circuit function:

```
#import "@preview/circuiteria:0.2.0"
#circuiteria.circuit({
  import circuiteria: *
  ...
})
```

2.1 Project installation

If you have installed Circuiteria directly in your project, import [src/lib.typ](#) and call the circuit function:

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

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

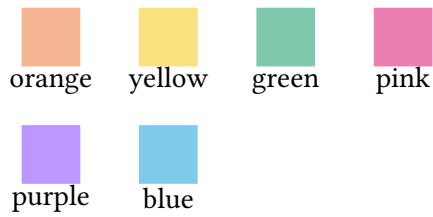
`rotate-anchor(anchor: str) -> str`

anchor `str`

The anchor to rotate

3.2.4 colors

Predefined color palette



3.3 wire

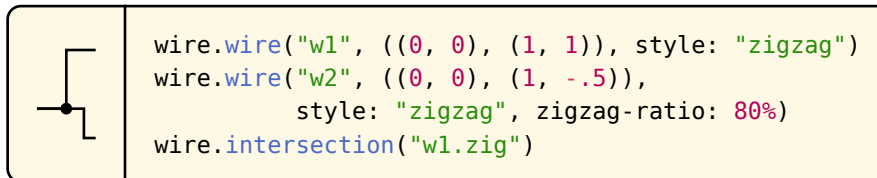
- [intersection\(\)](#)
- [stub\(\)](#)
- [wire\(\)](#)

Variables:

- [wire-styles](#)

3.3.1 intersection

Draws a wire intersection at the given anchor



Parameters

```

intersection(
  pt: point,
  radius: number,
  fill: color
)
  
```

pt point

A CeTZ compatible point / anchor

radius number

The radius of the intersection

Default: `.1`

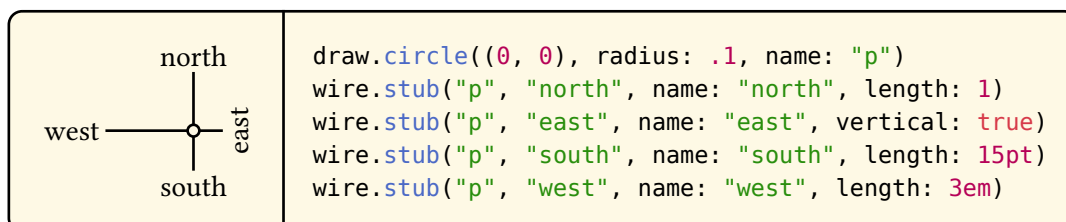
fill color

The fill color

Default: black

3.3.2 stub

Draws a wire stub (useful for unlinked ports)



Parameters

```

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

name-offset number

The name offset, perpendicular to the stub

Default: 0

3.3.3 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,
  guided-center: array,
  guided-margins: array,
  guided-sides: array,
  directed: bool,
  rotate-name: bool,
  zigzag-ratio: ratio,
  zigzag-dir: str,
  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`**guided-center** `array`

the horizontal and vertical guides of the center guides (only with style "guided")

Default: `(50%, 50%)`**guided-margins** `array`

the start and end of guided margins of the wire (only with style "guided")

Default: `(5%, 5%)`**guided-sides** `array`

the side of start and end array (must be either "north", "south", "west", "east") (only work with style "guided")

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

directed `bool`

If true, the wire will be directed, meaning an arrow will be drawn at the endpoint

Default: `false`

rotate-name `bool`

If true, the name will be rotated according to the wire's slope

Default: `true`

zigzag-ratio `ratio`

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

Default: `50%`

zigzag-dir `str`

The zigzag's direction. As either "vertical" or "horizontal" (only with dstyle "zigzag")

Default: `"vertical"`

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" )`

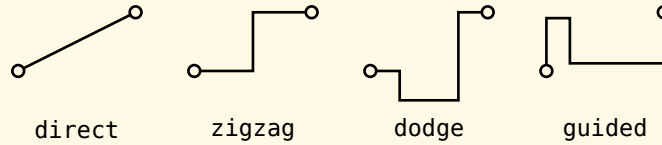
dodge-margins `array`

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

Default: `( 5%, 5% )`

3.3.4 wire-styles

List of valid wire styles



```

for i in range(4) {
  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))
wire.wire("w4", ("p6", "p7"), style: "guided",
  guided-center: (20%, 40%), guided-margins: (90%, 87%),
  guided-sides: ("north", "south"))

```

3.4 element

- `elmt()`
- `alu()`
- `block()`
- `extender()`
- `multiplexer()`
- `group()`

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: dictionary,
    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` 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). The keys are cardinal directions ("north", "east", "south", "west"). The values are tuples of (<start>, <end>) 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 dictionary

Dictionary 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

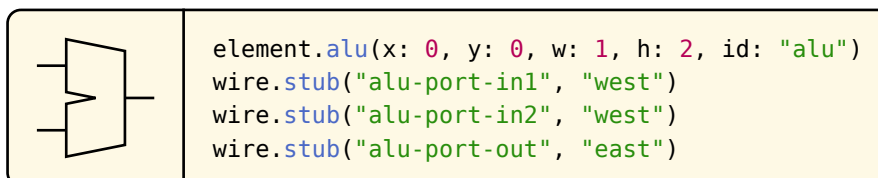
Default: (

ports: false

)

3.4.2 alu

Draws an ALU with two inputs

For parameters description, see [elmt\(\)](#)

Parameters

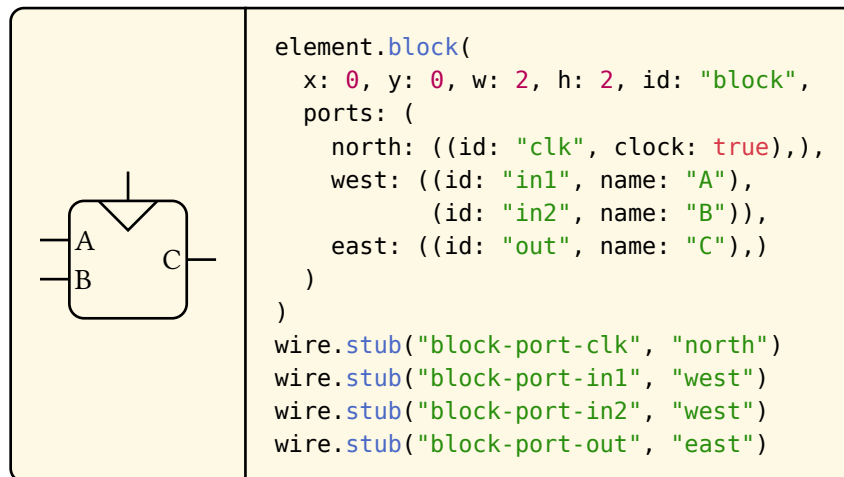
```

alu(
  x,
  y,
  w,
  h,
  name,
  name-anchor,
  fill,
  stroke,
  id,
  debug
)

```

3.4.3 block

Draws a block element



For parameters description, see [elmt\(\)](#)

Parameters

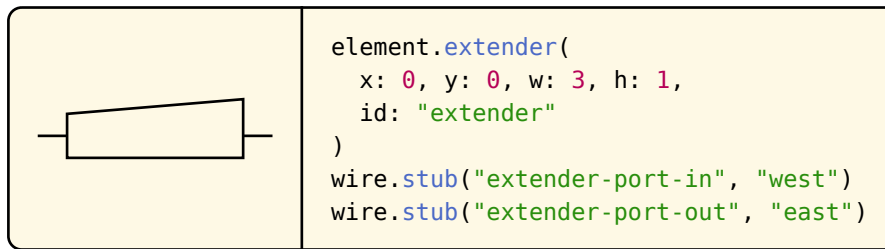
```

block(
  x,
  y,
  w,
  h,
  name,
  name-anchor,
  ports,
  ports-margins,
  fill,
  radius,
  stroke,
  id,
  debug
)

```

3.4.4 extender

Draws a bit extender



For other parameters description, see [elmt\(\)](#)

Parameters

```

extender(
  x,
  y,
  w,
  h,
  name,
  name-anchor,
  fill,
  stroke,
  id,
  h-ratio: ratio,
  align-out: bool,
  debug
)

```

h-ratio `ratio`

The height ratio of the left side relative to the full height

Default: `75%`

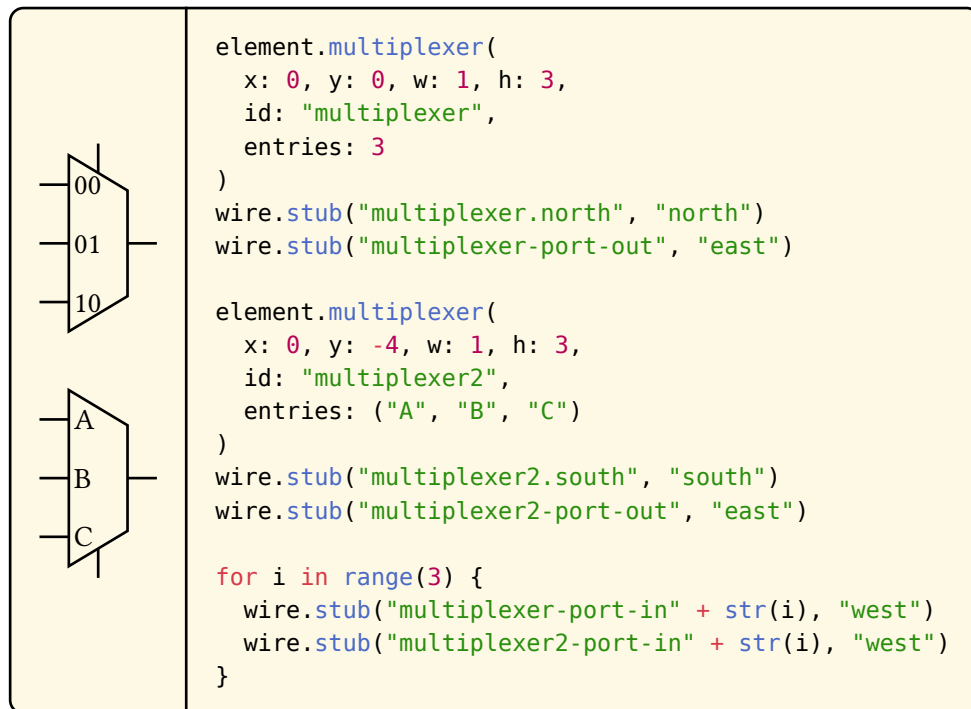
align-out `bool`

If true, the output and input ports are aligned, otherwise, the output port is centered on the right side

Default: `true`

3.4.5 multiplexer

Draws a multiplexer



For other parameters description, see [elmt\(\)](#)

Parameters

```

multiplexer(
  x,
  y,
  w,
  h,
  name,
  name-anchor,
  entries: int array,
  h-ratio: ratio,
  fill,
  stroke,
  id,
  debug
)

```

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 strings, it defines the name of each input.

Default: `2`

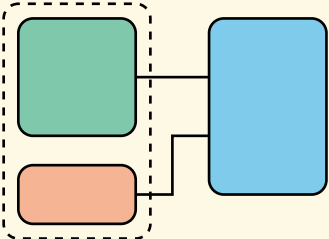
h-ratio `ratio`

The height ratio of the right side relative to the full height

Default: `60%`

3.4.6 group

Draws a group of elements



```

element.group(
  id: "g1", name: "Group 1", stroke: (dash: "dashed"),
  {
    element.block(id: "b1", w: 2, h: 2,
      x: 0, y: 1.5,
      ports: (east: ((id: "out")),),
      fill: util.colors.green
    )
    element.block(id: "b2", w: 2, h: 1,
      x: 0, y: 0,
      ports: (east: ((id: "out")),),
      fill: util.colors.orange
    )
  }
)
element.block(id: "b3", w: 2, h: 3,
  x: (rel: 1, to: "g1.east"),
  y: (from: "b1-port-out", to: "in1"),
  ports: (west: ((id: "in1"), (id: "in2"))),
  fill: util.colors.blue
)
wire.wire("w1", ("b1-port-out", "b3-port-in1"))
wire.wire("w2", ("b2-port-out", "b3-port-in2"),
  style: "zigzag")

```

Parameters

```

group(
  body: elements function,
  id: str,
  name: str,
  name-anchor: str,
  fill: color,
  stroke: stroke,
  padding: float length array dictionary,
  radius: number
)

```

body elements or function

Elements to group

id str

see `elmt()`

Default: ""

name `str`

The group's name

Default: `none`

name-anchor `str`

The anchor for the name.

Note: the name will be placed on the **outside** of the group

Default: `"south"`

fill `color`

see `elm()`

Default: `none`

stroke `stroke`

see `elm()`

Default: `black + 1pt`

padding `float` or `length` or `array` or `dictionary`

The inside padding:

- float / length: same for all sides
- array: either (`<all>`), (`<vertical>`, `<horizontal>`) or (`<top>`, `<right>`, `<bottom>`, `<left>`)
- dictionary: valid keys are "top", "right", "bottom" and "left"

Default: `0.5em`

radius `number`

The corner radius

Default: `0.5em`

3.5 gates

- [gate\(\)](#)
- [gate-and\(\)](#)
- [gate-nand\(\)](#)
- [gate-buf\(\)](#)
- [gate-not\(\)](#)
- [gate-or\(\)](#)
- [gate-nor\(\)](#)
- [gate-xor\(\)](#)
- [gate-xnor\(\)](#)
- [iec-gate\(\)](#)
- [iec-gate-and\(\)](#)
- [iec-gate-nand\(\)](#)
- [iec-gate-buf\(\)](#)
- [iec-gate-not\(\)](#)
- [iec-gate-or\(\)](#)
- [iec-gate-nor\(\)](#)
- [iec-gate-xor\(\)](#)
- [iec-gate-xnor\(\)](#)

3.5.1 gate

Draws a logic gate. This function is also available as `element.gate()`

Parameters

```
gate(
  draw-shape: function,
  x: number | dictionary,
  y: number | dictionary,
  w: number,
  h: number,
  inputs: int,
  fill: none | color,
  stroke: stroke,
  id: str,
  inverted: str | array,
  inverted-radius: number,
  debug: dictionary
)
```

draw-shape `function`

see [elmt\(\)](#)

Default: `default-draw-shape`

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`

inputs int

The number of inputs

Default: `2`

fill none or color

see `elmt()`

Default: `none`

stroke stroke

see `elmt()`

Default: `black + 1pt`

id str

see `elmt()`

Default: `""`

inverted str or array

Either “all” or an array of port ids to display as inverted

Default: `()`

inverted-radius number

The radius of inverted ports dot

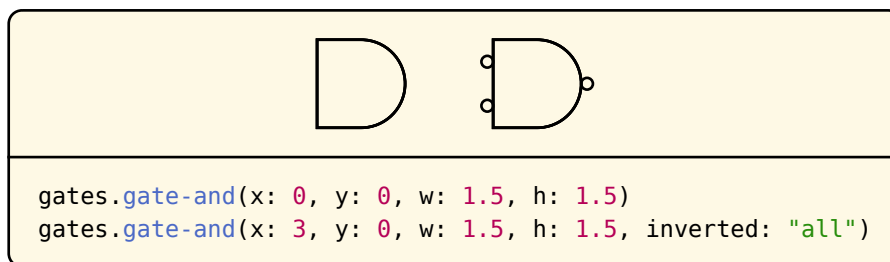
Default: **0.1****debug** dictionarysee [elmt\(\)](#)

Default: (

ports: **false**

)

3.5.2 gate-and

Draws an AND gate. This function is also available as `element.gate-and()`For parameters, see [gate\(\)](#)

Parameters

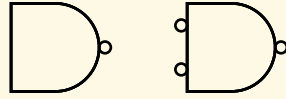
```

gate-and(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```

3.5.3 gate-nand

Draws an NAND gate. This function is also available as `element.gate-nand()`For parameters, see [gate\(\)](#)



```
gates.gate-nand(x: 0, y: 0, w: 1.5, h: 1.5)
gates.gate-nand(x: 3, y: 0, w: 1.5, h: 1.5, inverted: "all")
```

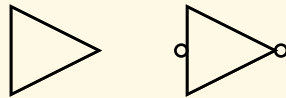
Parameters

```
gate-nand(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.4 gate-buf

Draws a buffer gate. This function is also available as `element.gate-buf()`

For parameters, see [gate\(\)](#)



```
gates.gate-buf(x: 0, y: 0, w: 1.5, h: 1.5)
gates.gate-buf(x: 3, y: 0, w: 1.5, h: 1.5, inverted: "all")
```

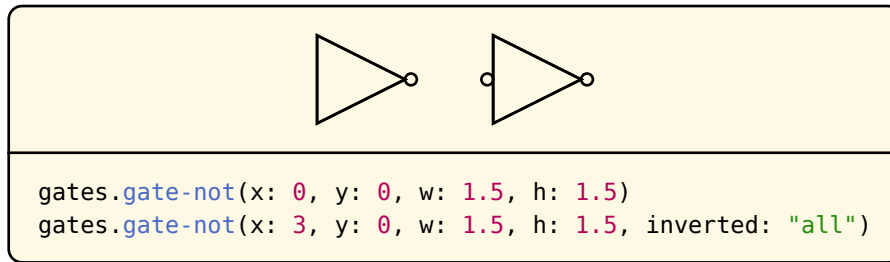
Parameters

```
gate-buf(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.5 gate-not

Draws a NOT gate. This function is also available as `element.gate-not()`

For parameters, see [gate\(\)](#)



Parameters

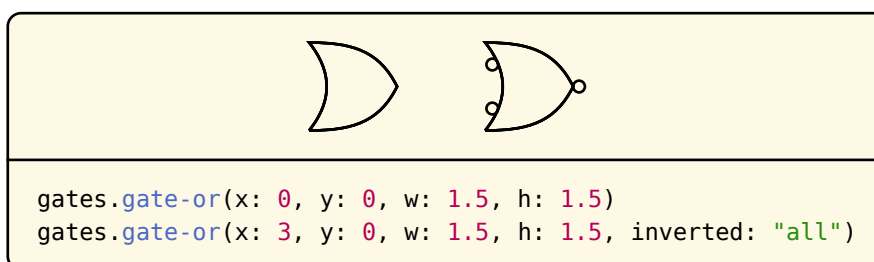
```

gate-not(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
  
```

3.5.6 gate-or

Draws an OR gate. This function is also available as `element.gate-or()`

For parameters, see [gate\(\)](#)



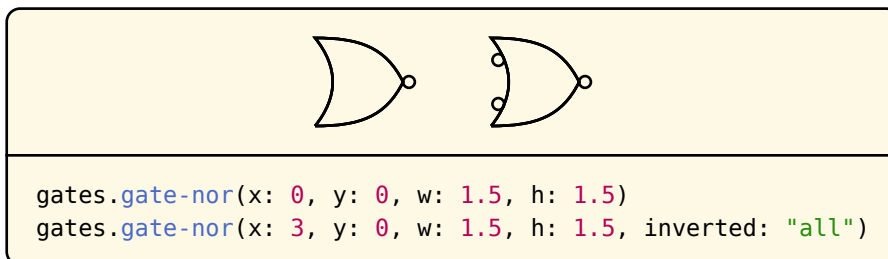
Parameters

```
gate-or(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.7 gate-nor

Draws a NOR gate. This function is also available as `element.gate-nor()`

For parameters, see [gate\(\)](#)

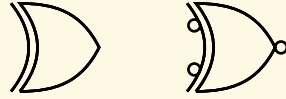
**Parameters**

```
gate-nor(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.8 gate-xor

Draws a XOR gate. This function is also available as `element.gate-xor()`

For parameters, see [gate\(\)](#)



```
gates.gate-xor(x: 0, y: 0, w: 1.5, h: 1.5)
gates.gate-xor(x: 3, y: 0, w: 1.5, h: 1.5, inverted: "all")
```

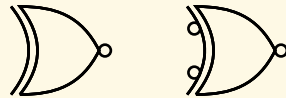
Parameters

```
gate-xor(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.9 gate-xnor

Draws a XNOR gate. This function is also available as `element.gate-xnor()`

For parameters, see [gate\(\)](#)



```
gates.gate-xnor(x: 0, y: 0, w: 1.5, h: 1.5)
gates.gate-xnor(x: 3, y: 0, w: 1.5, h: 1.5, inverted: "all")
```

Parameters

```
gate-xnor(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.10 iec-gate

Draws a logic gate. This function is also available as `element.iec-gate()`

Parameters

```

iec-gate(
  draw-shape: function,
  x: number | dictionary,
  y: number | dictionary,
  w: number,
  h: number,
  inputs: int,
  fill: none | color,
  stroke: stroke,
  id: str,
  inverted: str | array,
  inverted-radius: number,
  debug: dictionary,
  symbol: str
)

```

draw-shape `function`

see `elmt()`

Default: `default-draw-shape`

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`

inputs `int`

The number of inputs

Default: `2`**fill** `none` or `color`see `elm()`Default: `none`**stroke** `stroke`see `elm()`Default: `black + 1pt`**id** `str`see `elm()`Default: `""`**inverted** `str` or `array`

Either “all” or an array of port ids to display as inverted

Default: `()`**inverted-radius** `number`

The radius of inverted ports dot

Default: `0.1`**debug** `dictionary`see `elm()`Default: (
 ports: `false`
)**symbol** `str`

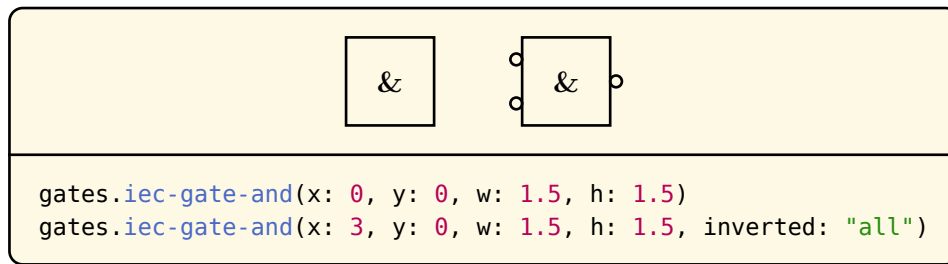
The symbol to display at the center of the gate

Default: `""`

3.5.11 iec-gate-and

Draws an IEC-AND gate. This function is also available as `element.iec-gate-and()`

For parameters, see [iec-gate\(\)](#)



Parameters

```

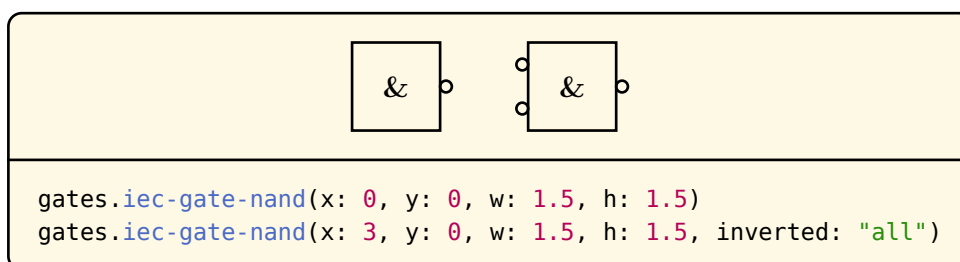
iec-gate-and(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```

3.5.12 iec-gate-nand

Draws an IEC-NAND gate. This function is also available as `element.iec-gate-nand()`

For parameters, see [iec-gate\(\)](#)



Parameters

```

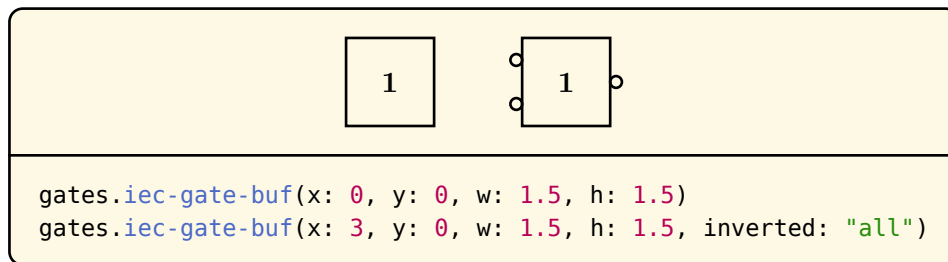
iec-gate-nand(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```

3.5.13 iec-gate-buf

Draws an IEC buffer gate. This function is also available as `element.iec-gate-buf()`

For parameters, see [iec-gate\(\)](#)

**Parameters**

```

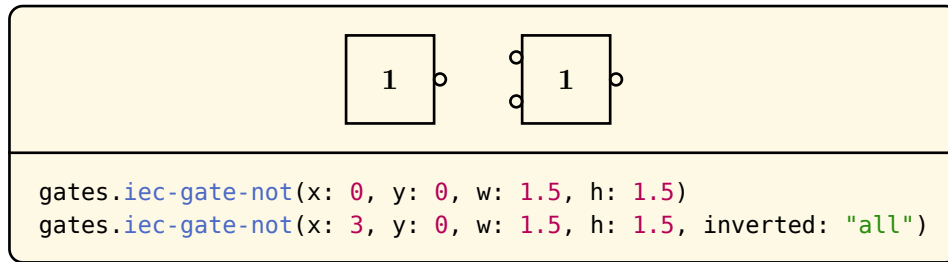
iec-gate-buf(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```

3.5.14 iec-gate-not

Draws an IEC NOT gate. This function is also available as `element.iec-gate-not()`

For parameters, see [iec-gate\(\)](#)



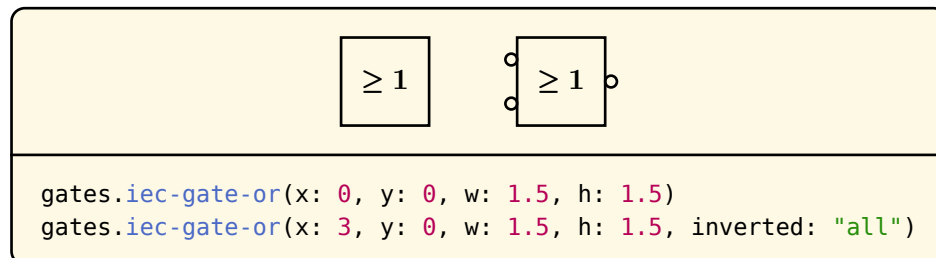
Parameters

```
iec-gate-not(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.15 iec-gate-or

Draws an IEC-OR gate. This function is also available as `element.iec-gate-or()`

For parameters, see [iec-gate\(\)](#)



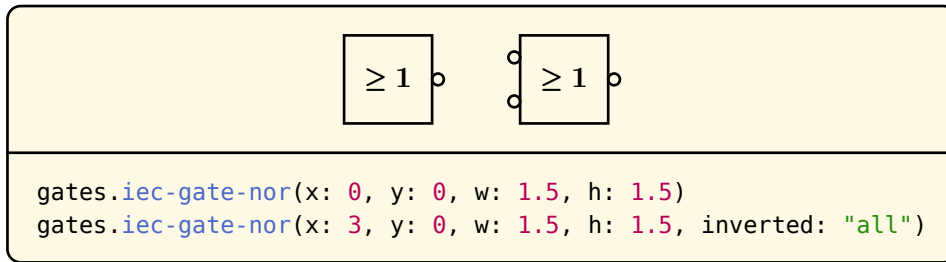
Parameters

```
iec-gate-or(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)
```

3.5.16 iec-gate-nor

Draws an IEC-NOR gate. This function is also available as `element.iec-gate-nor()`

For parameters, see [iec-gate\(\)](#)



Parameters

```

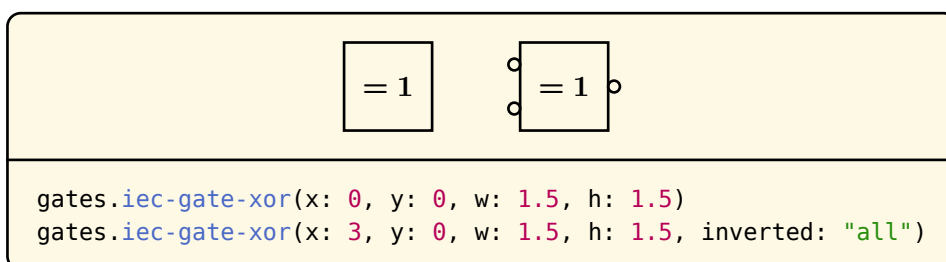
iec-gate-nor(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```

3.5.17 iec-gate-xor

Draws an IEC-XOR gate. This function is also available as `element.iec-gate-xor()`

For parameters, see [iec-gate\(\)](#)



Parameters

```

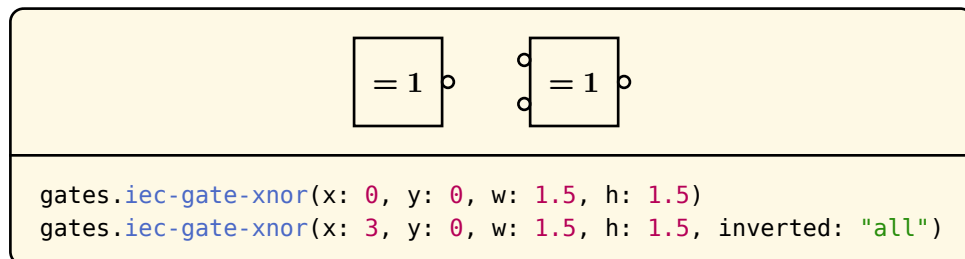
iec-gate-xor(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```

3.5.18 iec-gate-xnor

Draws an IEC-XNOR gate. This function is also available as `element.iec-gate-xnor()`

For parameters, see [iec-gate\(\)](#)

**Parameters**

```

iec-gate-xnor(
  x,
  y,
  w,
  h,
  inputs,
  fill,
  stroke,
  id,
  inverted,
  debug
)

```