Connector - Reference manual

Gianluigi Forte

August 24, 2021

Introduction

Connector is a library written in *Asymptote* language to generate figures of electronic schematics. LaTeXand *Asymptote*, differently from other programs like Word or Write, are languages designed to produce documents with the best quality outputs for prints or presentations starting from a description, of the content to be produced, expressed programmatically in a source code text file and then produced in output after the compilation process. The idea behind the *connector* library is to provide a ready to use set of functions written in Asymptote language to draw electrical components, link them with connection (wires), decorate with labels and produce the output figure as pdf or png files to be embedded in a larger Latex document or any kind of other usage like websites, videos, presentations and so on. An example of the image generated with *connector* is shown in figure 1. The source code `ThreePhaseInverter.asy` to produce the figure is included in the library.

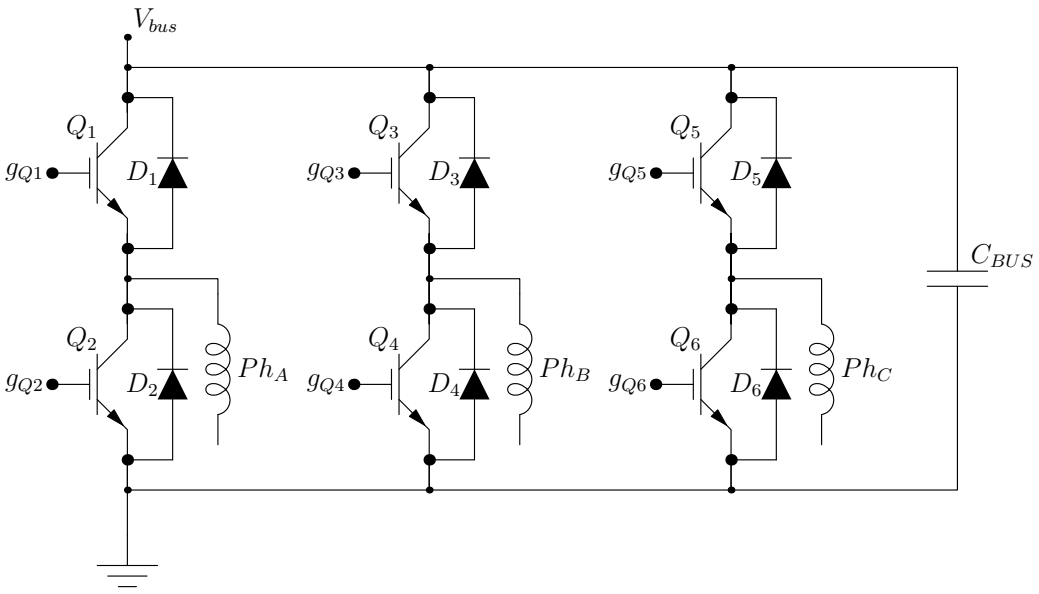


Figure 1: Example of schematics generated with *connector*

Getting started

Before to use the *Connector* library is necessary to install the *Asymptote* compiler. Follow the instruction described in the *Asymptote* website at this link. It is required to install the LaTeXenvironment as mentioned in the *Asymptote* installation documentation. It is also suggested to install *Node.JS*

(link) to execute the script `watch.js` that is able to generate automatically the output of the *.asy* file you are working every time you modify it. Finally, it is also kindly suggested to install *Visual Studio Code* (link) and some of its fantastic extensions to support *Asymptote* and LaTeXlanguages.

Download and unpack the *Connector* library in a working folder of your file system, take a look at the examples `.asy` file that are provided to take inspiration. Create a `newFile.asy` with your code and compile it with the command line `asy newFile.asy`. The compiled `.pdf` file will be created in the current folder. Or use `node watch.js newFile.asy` for the automatic generation after each file modification. The compiled `.pdf` file will be created in the `generated` folder in this case. It is possible to produce a `.png` file adding the line `settings.outformat="png";` to your code or replacing the line `settings.outformat="pdf"` with `"png"` if it is already present.

It is also kindly suggested to learn the *Asymptote* language to be familiar with the *Connector* library and to use it at the best.

If you want to collaborate with the *Connector* project to add other symbols, for the internationalization of the documentation, to fix or report some bugs, to request other features, or for any other kind of collaborations, please let me know at *koalakoker@gmail.com*.

Electronic Symbols

The following list of components have been included in the library and can be used out-of-the-box: node, resistor, capacitor, inductor, fuse, diode, relay, relay SPDT, IGBT, MOS, power ground, signal ground. Maybe other will be added in the future.

In the figures from 2 to 13 are shown the components, the anchor points as a black dot. The anchor point direction is indicated with a green line starting from the dot and going outward, the anchor point index number is indicated near the end of the green line. It is also indicated the position of the pivot point $(0, 0)$ and also the position of the $(1, 0)$ point for a spatial reference; both are indicated with a red cross.

In figure 2 is shown a generic node symbol. See `nodeInfo.asy` code to reproduce the figure. Note that this symbol has four anchor points going in four different directions. It can be very useful to connect different part of the schematics with a connector line.

To draw any symbol without the indication of the anchor points, pivot point and all other info simply call the `draw` method without the `drawOpt` parameter or setting it to `null`.

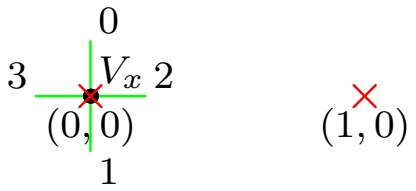


Figure 2: Generic node symbol

In figure 3 is shown the resistor symbol. See `resistorInfo.asy` code to reproduce the figure.

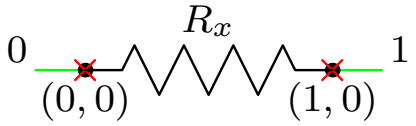


Figure 3: Resistor symbol

In figure 4 is shown the capacitor symbol. See `capaciorInfo.asy` code to reproduce the figure.

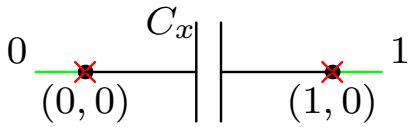


Figure 4: Capacitor symbol

In figure 5 is shown an inductor symbol. See `inductorInfo.asy` code to reproduce the figure.

In figure 6 is shown the fuse symbol. See `fuseInfo.asy` code to reproduce the figure.

In figure 7 is shown the diode symbol. See `diodeInfo.asy` code to reproduce the figure.

In figure 8 is shown the relay symbol. See `relayInfo.asy` code to reproduce the figure.

In figure 9 is shown the relay SPDT symbol. See `relaySPDTInfo.asy` code to reproduce the figure.

In figure 10 is shown the IGBT symbol. See `igbtInfo.asy` code to reproduce the figure.

In figure 11 is shown the MOSFET transistor symbol. See `mosInfo.asy` code to reproduce the figure.

In figure 12 is shown the power ground symbol. See `gndPowerInfo.asy` code to reproduce the figure.

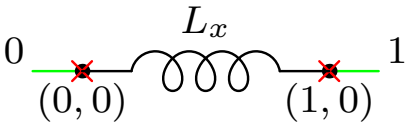


Figure 5: Inductor symbol

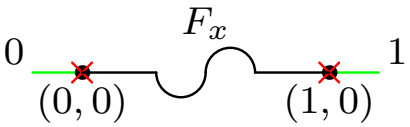


Figure 6: Fuse symbol

In figure 13 is shown the signal ground symbol. See `gndSignalInfo.asy` code to reproduce the figure.

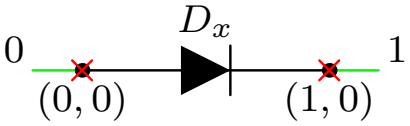


Figure 7: Diode symbol

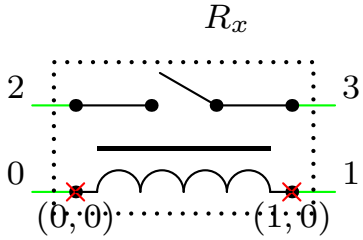


Figure 8: Relay symbol

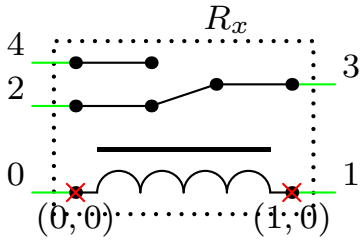


Figure 9: Relay SPDT symbol

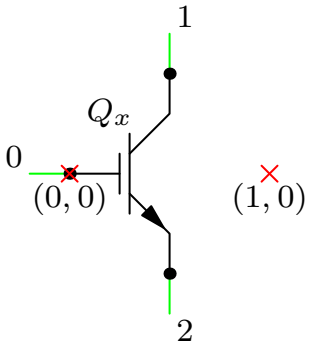


Figure 10: Igbt symbol

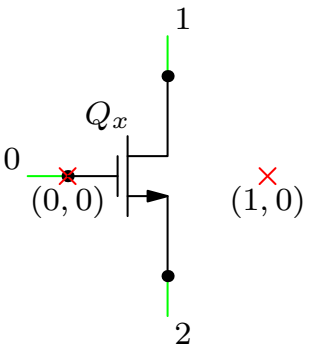


Figure 11: MOSFET transistor symbol

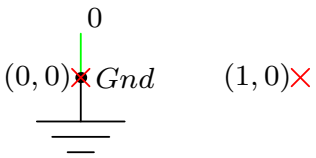


Figure 12: Power ground symbol

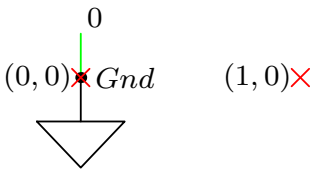


Figure 13: Signal ground symbol

Symbol positioning and orientation

For each symbol instantiated it is possible to define its placement in the drawing calling the methods `setPos` and passing the position of the pivot point as parameter before calling the `draw` method. For example, the code:

```
size(4cm);
defaultpen(fontsize(8pt));
import resistor;
import drawOptions;
Resistor r = Resistor();
r.setPos((1,0));
r.draw(DrawOption(showOrigin = true));
```

will draw the image shown in figure 14. The same effect can be done passing the position of the pivot point as parameter in the constructor function `Resistor r = Resistor((1,0))`.

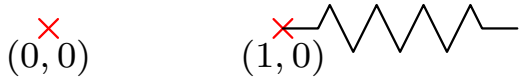


Figure 14: Placing the pivot point of the symbol in $(1, 0)$

In a similar way is possible to define the orientation of the symbol calling the methods `setOrient` and passing the orientation of the symbol around the pivot point as parameter before calling the `draw` method. For example, the code:

```
size(4cm);
defaultpen(fontsize(8pt));
import resistor;
import drawOptions;
Resistor r = Resistor();
r.setOrient(90);
r.draw(DrawOption(showOrigin = true));
```

will draw the image shown in figure 15. The same effect can be done passing the orientation of the symbol around the pivot point as parameter in the constructor function `Resistor r = Resistor(orient = 90)`. The only valid options for the orient parameter are 0, 90, -90 and 180. The anchor points of the symbol will be re-oriented according to the orientation of the symbol.

It is possible to use the same instance of the symbol to print it several times in different positions and/or different orientations. In this case can be

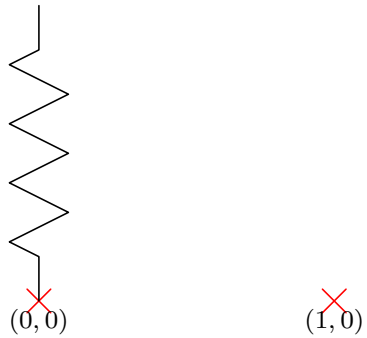


Figure 15: Set the orientation of the symbol around the pivot point as 90

useful to change the label of the printed instance using the method `setLabel` and passing the new label as parameter before calling the `draw` method. For example, the code:

```
size(4cm);
defaultpen(fontsize(8pt));
import components;
import drawOptions;
import connector;
Resistor r = Resistor("$R_1$");
Capacitor c = Capacitor((1,0), -90,"$C_1$");
Node n = Node((0,-1.1));
n.draw();
r.draw(DrawOption(showOrigin = true));
drawAnchorConnector(n, 2, c, 1);
c.draw();
r.setPos((1,0));
r.setLabel("$R_2$");
r.draw();
c.setPos((2,0));
c.setLabel("$C_2$");
c.draw();
drawAnchorConnector(n, 2, c, 1);
```

will draw the image shown in figure 16.

In figure 16 the R_1 and R_2 are drown using the same *Resistor* instance r. In the same way C_1 and C_2 are drown using the same *Capacitor* instance c. The *Node* n is used to draw the connectors of the bottom of the figure. If the positioning of the symbol label is not perfect, it is possible to send a *pair* as second parameter of the method `setLabel` to set a displacement position

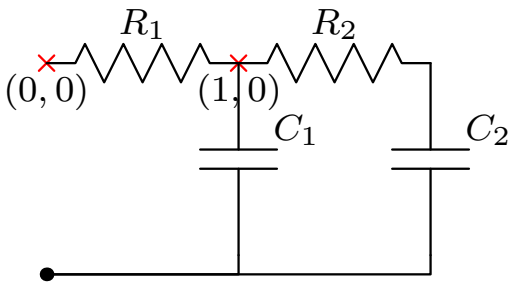


Figure 16: Example of multiple symbol placement with the same instance

only for the label.

Anchor points

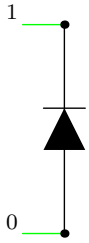
The anchor points are used to connect the symbol using *Connectors* as described in section Connectors. Each symbol has a set of anchor points already defined. Use figures from 2 to 13 as reference to know which anchor points are defined in the symbol.

An anchor point has a position indicated with a dot in the symbol figures, this will be the starting or ending point of the connectors. The anchor point has an index, indicated with a number in the symbol figures. This index will be used to identify the anchor point among the various present in the symbol. The anchor point has a direction indicated with the green line in the symbol figures, that start from the dot and go outward. This will be the direction of the connector that starts o ends in this specific anchor point.

It is possible to get the position of an anchor point of an instance of a symbol with the function `GetAnchorPos(obj, index)`. Each symbol is a derived class from a generic object. The function returns a *pair* that indicate the coordinate of the anchor position; this can be used to place other symbols aligned with that position. Many examples provided with the library make use of this strategy to align the symbols. E.g., `ThreePhaseInverter.asy`.

It is possible to change the orientation of an anchor point with the function `setAnchorDirection(object, index, direction)`. Where `object` is the instance of the symbol on which operate, `index` is the index of the anchor point to be modified and `direction` is the new direction. The available directions are: *DN* (equal to north or up), *DS* (equal to south or down), *DE* (equal to east or right) or *DW* (equal to west or left). In figure 17 are shown respectively the outputs of the examples `anchorExample1.asy` 17a and `anchorExample1.asy` 17b. It is an example on how to customize the direction of the default anchor points.

(a) Default directions



(b) Custom directions

Figure 17: Changing the anchor points default directions

It is also possible to add other anchor points to a symbol using the function `AddAnchorPoint(object, position, direction)`. Where `object` is the instance of the symbol on which operate, `position` is a *pair* that defines the position of the new anchor point; note that this `position` is the absolute position of the anchor point in the drawing, while the relative position respects the parent symbol is computed by the function. And `direction` is the direction of the new anchor point. The available directions are, with the same meaning mentioned before: *DN*, *DS*, *DE*, *DW*. In figure 18 is shown the output of the `anchorExample3.asy` code that shows how to add other anchor points to a symbol; this can be very useful to put different symbols in parallel.

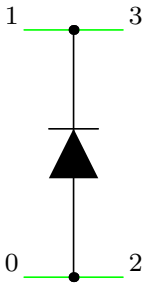


Figure 18: Adding new anchor points to a symbol

Connectors

It is possible to draw connectors (wires) between symbols with the function `drawAnchorConnector`. In particular the connection is done starting from one anchor point to another anchor point that are defined in the symbol. See from figure 2 to figure 13 to check the default anchor points defined in each symbol. See section Anchor points for details about anchor points. The path of the connector is automatically computed by the library in the best way and is drown according to the direction of the anchor point. The figure 19 show an example of connections between generic objects.

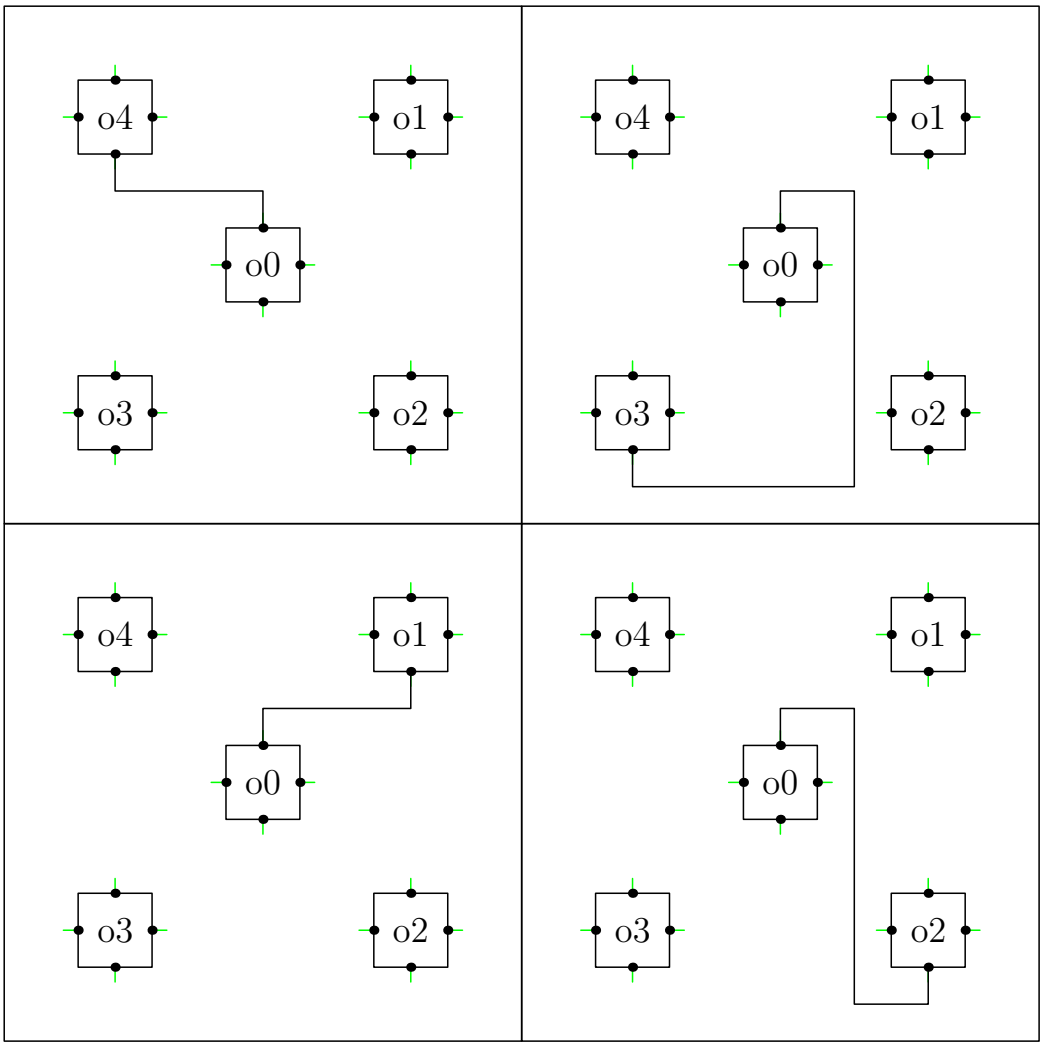
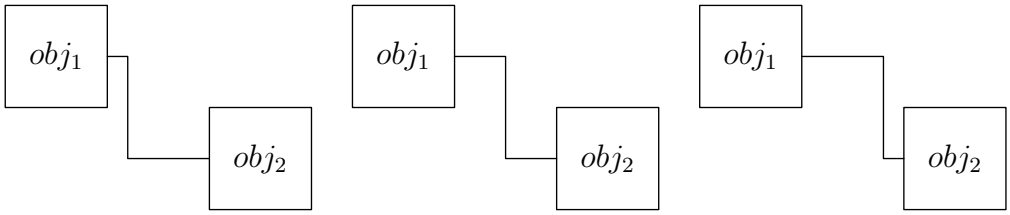


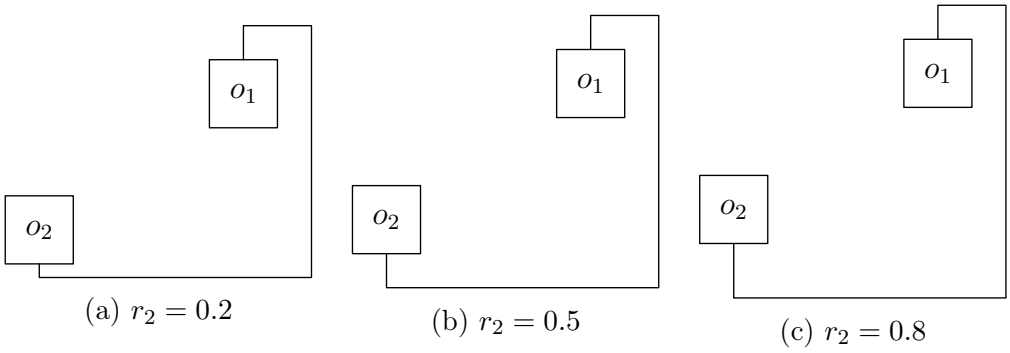
Figure 19: Example of use of connectors between objects

The `drawAnchorConnector` function takes as parameters respectively:



(a) $r_1 = 0.2$ (b) $r_1 = 0.5$ (c) $r_1 = 0.8$

Figure 20: Effect of r_1 parameter value on connector



(a) $r_2 = 0.2$ (b) $r_2 = 0.5$ (c) $r_2 = 0.8$

Figure 21: Effect of r_2 parameter value on connector

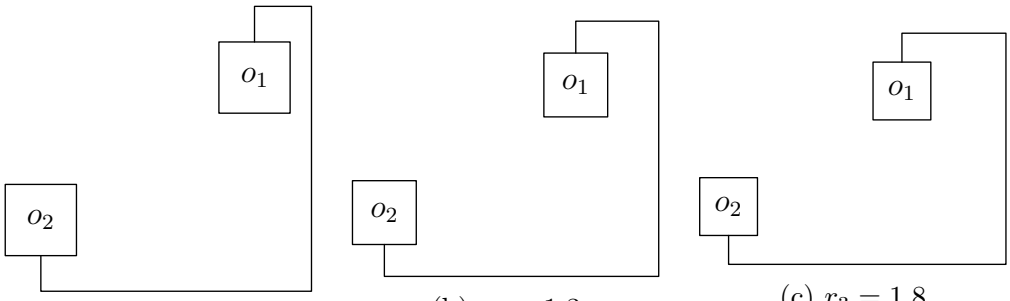
the first object to be connected, the anchor point index of the first object, the second object to be connected and the index of the anchor point of the second object.

```
drawAnchorConnector(obj1, Anchor1, obj2, Anchor2)
```

There are also other three optional parameters that can be used to set the aspect of the connection (r_1, r_2, r_3). As shown in figure 20, the r_1 parameter can be used to define the distance of the two corners of the connector line (and so the distance of the vertical line in figure 20). The parameter r_1 defines the distance of the first corner of the connector line from the first object expressed in percentage of the distance of the two objects.

The figure 21 shows the r_2 parameters. It affects the distance between the second object and the first corner of the connector line.

The figure 22 shows the r_3 parameters. It affects the distance between the first corner and the second corner (or equivalently the distance of the vertical bar) of the connector line.



(a) $r_3 = 0.8$ (b) $r_3 = 1.3$ (c) $r_3 = 1.8$

Figure 22: Effect of r_2 parameter value on connector