S*a*-Ti*k*Z: a library to draw switching architectures

Claudio Fiandrino

Abstract

The article illustrates how it is possible to draw some types of switching architectures in a simple manner. The S*a*-Ti*k*Z library provides not only the keys devoted to the drawing part, but also the keys devoted to customize the aspect of the architectures in the spirit of the Ti*k*Z syntax.

Sommario

L'articolo illustra come disegnare alcuni tipi di architetture di rete in modo semplice. La libreria S*a*-Ti*k*Z definisce non soltanto chiavi per il disegno, ma anche per personalizzare l'aspetto delle architetture utilizzando come base la sintassi di Ti*k*Z.

1 Introduction

Switching architectures are a combination of hardware and software that essentially move out to the proper direction data coming into the network. Each network node is also called *switching unit* or *module* and, in combination with the integrated circuits, form the hardware part of the architecture; the software part, instead, is in charge of switching the paths the data should follow.

Ti*k*Z is nowadays one of the most used packages for drawing purposes in LaTeX. It is built in a modular fashion by means of libraries: a part from some fundamental ones always loaded, the user is in charge of loading the libraries he needs in the preamble of his document. For example:

```
\usepackage{tikz}
\usetikzlibrary{list of libraries}
```

The list of libraries should be set as a comma-separated list.

The package/library S*a*-Ti*k*Z has been written in such a way that it could be loaded both as a package or as a Ti*k*Z library; therefore:

```
\usepackage{tikz}
\usetikzlibrary{switching-architectures}
```

and

```
\usepackage{sa-tikz}
```

are both valid.

S*a*-Ti*k*Z in particular, is able to represent the following switching architectures:

- Clos Networks;
- Benes Networks.

The project is also available on GitHub and its home page is http://cfiandra.github.com/Sa-TikZ/: several examples are provided and there is even a presentation in which the connections between input and output ports of the architecture are added one at a time.

The rest of the work is organized as follows:

- section 2 provides an overview on the architectures and the basic keys;
- section 3 explains how the drawing mechanism is implemented, considering also the related keys devoted to customize the drawing aspect;
- section 4 shows the keys devoted to quickly draw Clos and Benes Networks: these approaches allows to create network examples;
- section 5 concludes the work.

2 Overview on the architectures

2.1 Clos Networks

Clos Networks are 3-stages networks in which there is full interconnection among the consecutive stages; for example, a module belonging to the first stage has paths towards all the modules in the second stage.

A Clos Network with N input ports and M output ports is defined as shown in figure 1:

- m_1 represents the number of input ports per module in the first stage;
- r_1 represents the number of modules in the first stage;
- m_2 represents the number of input ports per module in the second stage;
- r_2 represents the number of modules in the second stage;
- m_3 represents the number of input ports per module in the third stage;
- r_3 represents the number of modules in the third stage.

Thus:

- $N = m_1 \cdot r_1$
- $M = m_3 \cdot r_3$

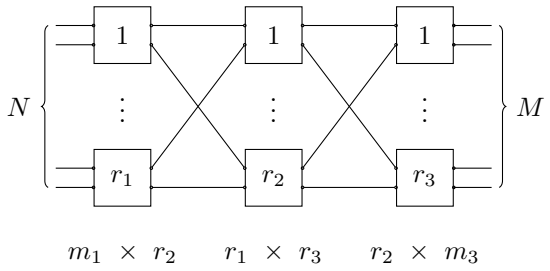


FIGURE 1: Clos network definition

Usually, as design parameters are always provided the number of inputs and outputs ports N and M and the number of modules in the first and third stage r_1 and r_3. The other parameters, m_1, m_3, r_2 and m_2, are automatically computed by S*a*-Ti*k*Z; notice that the number of modules in the second stage r_2 defines the type of the network:

- when $r_2 \geq m_1 + m_3 - 1$ the network is said to be *Strictly non Blocking*;
- when $r_2 = \max\{m_1, m_3\}$ the network is said to be *Rearrangeable*.

This is very important because the two types of networks have not only very different behaviours, but also a different structure. S*a*-Ti*k*Z, very simply, computes these constraints in background upon choosing the network type:

- the `clos snb` key defines a Strictly non Blocking Clos Network by computing:

```
\pgfmathtruncatemacro\rtwo{\mone+\mthree-1}
```

- the `clos rear` key defines a Rearrangeable Clos Network by computing:

```
\pgfmathtruncatemacro\rtwo{%
 max(\mone,\mthree)
}
```

The macro `\pgfmathtruncatemacro` allows to define other macros (`\rtwo` in this case) with a truncated value; this because, in Ti*k*Z, the result of an operation is something like `1.0`, where the notation `.0` means *an angle of 0 degree from the anchor placed on the left of the dot*. With a truncated macro instead, automatically, the result of an operation is stored as `1` in order to avoid this problem.

The macros `\mone` and `\mthree` are defined as:

```
\pgfmathtruncatemacro\mone{\N/\rone}
\pgfmathtruncatemacro\mthree{%
 \M/\rthree
}
```

where `\N`, `\rone`, `\M` and `\mone` are all macros associated to `pgfkeys`; their definition is similar, for example, `\N` is:

```
\pgfkeys{/tikz/.cd,%
      N/.initial=10,%
      N/.get=\N,%
      N/.store in=\N,%
}%
```

The `/tikz/.cd` is needed to locate the key under the path `/tikz`: this is helpful because later on the usual `\tikzset` could be used to assign values to the key. The key-path is very important because trying to assign values to a key using a wrong path not only makes the assignment null, but also gives rise to an error. By means of `N/.initial=10` it is possible to assign an initial value to the key; note that this is not the *default* value: indeed, it is used just in the first picture when no values for `N` are provided. When some values are assigned to `N` in a picture, they are stored in the macro `\N` (`N/.store in=\N`) and, for subsequent uses within the picture, `N` keeps the value of `\N` in case no other values are specified: this is possible by means of `N/.get=\N`.

The keys `clos snb` and `clos rear` should be introduced within the usual Ti*k*Z `node` syntax; an example:

```
\begin{tikzpicture}
\node[clos rear]{};
\end{tikzpicture}
```

The major potentiality of the library is that the architectures are drawn by simply choosing the design parameters; to this purpose the following keys should be used:

- `N` and `M` to choose the number of input/output ports for the first and third stage respectively;
- `r1` and `r3` to choose the number of modules for the first and third stage respectively.

Using the same input parameters allows to obtain different networks: examples are shown in figures 2 and 3.

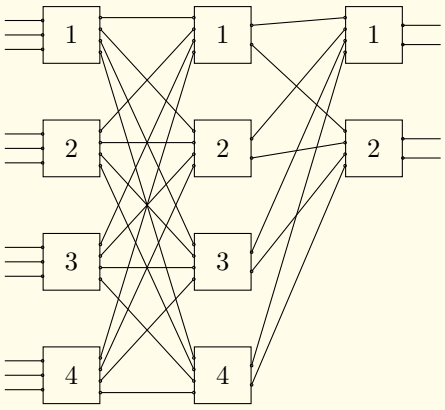
Notice one important fact: since the keys `clos snb` and `clos rear` need the input parameters specification, the possible customization should be declared before specifying the network type. This holds also for all the other keys representing network types.

2.2 Benes Networks

A Benes Network is a symmetric Rearrangeable Clos Network factorized with a factor 2 exploiting 2×2 modules.

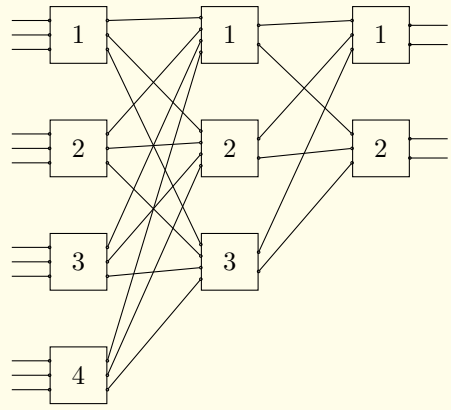
The number of the stages is not fixed, but depends on the size of the network. S*a*-Ti*k*Z takes into account this fact by providing two keys:

- the `benes` key displays the network without showing all the stages: the aspect is similar to a Rearrangeable Clos Network with three stages where the second one contains macro-modules hiding the actual inner stages;




```
\begin{tikzpicture}
\node[N=12,M=4,r1=4,r3=2,clos snb]{};
\end{tikzpicture}
```

FIGURE 2: Strictly non Blocking Clos Network




```
\begin{tikzpicture}
\node[N=12,M=4,r1=4,r3=2,clos rear]{};
\end{tikzpicture}
```

FIGURE 3: Rearrangeable Clos Network

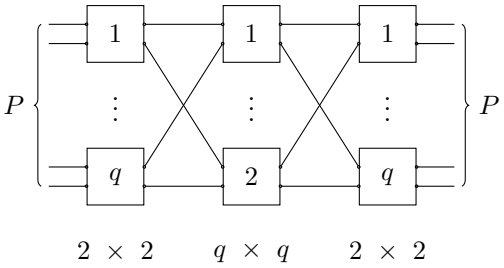


FIGURE 4: Benes Network definition

- the `benes complete` key displays the network showing all the stages.

The modules are 2×2: this implies that the number of input and output ports per module is fixed. As a consequence, the parameters m_1, m_2 and m_3 now have no meaning; indeed, in the code their respective macros `\mone`, `\mtwo` and `\mthree` been gathered into a single macro `\m`, defined as:

```
\pgfmathtruncatemacro{\m}{2}
```

The network simmetry means that the number of total input and output ports are the same; for the purpose, it has been defined a new key `P`, which simply replaces the previous keys `N` and `M`. Its definition is very similar to `N`'s one:

```
\pgfkeys{/tikz/.cd,%
      P/.initial=8,%
      P/.get=\P,%
      P/.store in=\P,%
}%
```

Actually, `P` can not assume any values: since the architecture is factorized with a factor 2 it can assume just

$$P = 2^p \qquad p = 2, 3, 4, \ldots$$

At the moment, S*a*-Ti*k*Z does not provide any check mechanism, therefore the user is responsible for the correct key setting.

The fixed number of input and output ports and the network symmetry influence the number of modules per stage. In the first and last stage, the parameters r_1 and r_3 and the respective macros `\rone` and `\rthree` have been gathered into a single macro `\r`, defined as:

```
\pgfmathtruncatemacro{\r}{\P/\m}
```

In the graphical representation of a Benes Network in figure 4, `\r` is called q.

The last stage is actually the third stage in case the `benes` key has been used: the second stage, thus contains only 2 macro-modules as per the definition of Rearrangeable Clos Network. If instead the `benes complete` key has been used, the last stage should be computed by means of the following formula:

$$\mathcal{S} = 2 \log_2 P - 1 \tag{1}$$

For example, if `P=16`:

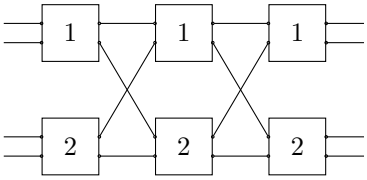
$$\mathcal{S}_{16} = 7$$

In this case, all the stages have the same number of modules, which is equal to q. The interconnections among all the stages are automatically drawn by S*a*-Ti*k*Z thanks to a specific algorithm "dBnc": its explanation, in detail, is reported in the appendix of the manual FIANDRINO (2013). The algorithm in principle works regardless the network size (specified by `P`), but for internal limitations of PGF it is difficult to characterize networks bigger than 128×128.

Examples of the difference between the `benes` and `benes complete` networks are shown in the

figures 5 and 6: since no value is given to P, the initial one is considered, that is 8.

Notice that each macro module in figure 5 actually comprises another Benes Network; an 8×8 Benes Network, indeed, is built recursively with two 4×4 Benes Networks:



For example, in a 32×32 Benes Network, it is possible to identify:

- two 16×16 Benes Networks;
- four 8×8 Benes Networks;
- eight 4×4 Benes Networks.

3 The drawing mechanism

The drawing mechanism is in charge of:

- drawing the modules of the various stages;
- drawing the interconnections between the modules.

3.1 The modules

The drawing method for the modules is very simple for clos snb, clos rear and benes networks because the number of the stages is always fixed, 3, while for benes complete networks is more difficult.

The idea behind all is to draw a path by positioning the modules at a given distance one after the other; the number of modules is always known: it is \rone, \rtwo and \rthree for clos snb and clos rear networks; as for benes networks, instead, it is \r for the first and third stage and 2 for the second stage.

For example, the code drawing the modules in the first stage is:

```
\foreach \i in {1,...,\rone}{
 \path let
 \n1 = {int(0-\i)},
 \n2={0-\i*\moduleysep}
  in node[module,#1,
    module opacity, yshift=1cm]
  (r1-\i) at +(0,\n2) {
   \pgfmathparse{int(multiply(\n1,-1))}
   \pgfmathresult};
...
}
```

During the path creation two things are computed:

- the label of the module by means of \n1 = {int(0-\i)}. Notice that the macro \i which iterates is not a truncated macro: this mechanism allows to print, after the multiplication by -1 done through \pgfmathparse{int(multiply(\n1,-1))}, a truncated number thanks to the int operation. Actually, the method will be improved in the next version by simply using the count option of the foreach.
- the distance of each module thanks to \n2={0-\i*\moduleysep}; notice that \n2 is used as y coordinate in at +(0,\n2): this means that at each iteration a constant distance is increased; the distance ultimately depends on \moduleysep, customizable through the module ysep key:

```
\pgfkeys{/tikz/.cd,%
 module ysep/.initial={1.5},%
 module ysep/.get=\moduleysep,%
 module ysep/.store in=\moduleysep,%
}%
```

Some styles and a personalization aspect are applied to the node: module,#1,module opacity,yshift=1cm; the module style is simply defined as:

```
\tikzset{module/.style={%
    draw,
    rectangle,
    minimum size=\modulesize,
    font=\modulefont,
    }
}
```

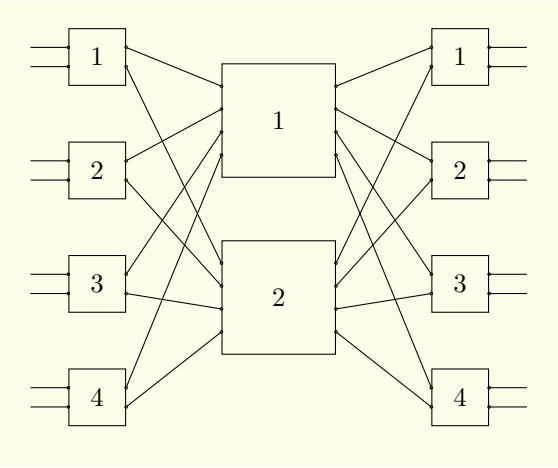
where \modulesize and \modulefont are macros representing the keys module size and module font.

The argument #1 of the style is peculiar: it helps to masquerade the labels by means of the key module label opacity. As all the TikZ opacity keys, when it is equal to 1, it does not alter the visibility, but when it is set to 0, it makes opaque the label. An example of usage is figure 7. The previous setting is implemented with the module opacity style, who sets the label with the desired opacity:

```
\tikzset{module opacity/.style={
  text opacity=\modulelabelopacity,
  }
}
```

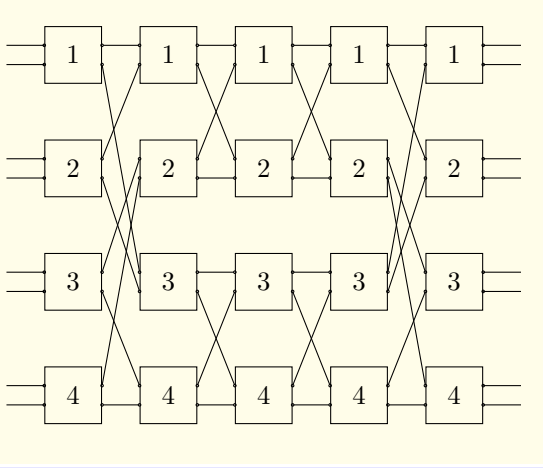
since the \modulelabelopacity is a macro customizable by the key module label opacity:

```
\pgfkeys{/tikz/.cd,%
 module label opacity/.initial={1},%
 module label opacity/.get=%
  \modulelabelopacity,%
 module label opacity/.store in=%
```



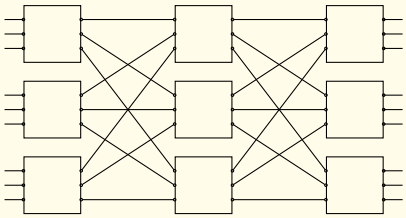

```
\begin{tikzpicture}
\node[benes]{};
\end{tikzpicture}
```

FIGURE 5: Benes Network




```
\begin{tikzpicture}
\node[benes complete]{};
\end{tikzpicture}
```

FIGURE 6: Benes complete Network



```
\tikzset{module size=0.75cm,
 pin length factor=0.5,
 module ysep=1,
 module xsep=2}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
\node[clos rear={module label opacity=0}]{};
\end{tikzpicture}
```

FIGURE 7: Clos Network without the labels

```
  \modulelabelopacity,%
}%
```

Finally, the `yshift=1cm` is just needed to let the first modules be placed on the `0` y coordinate: indeed, using `\n2={0-\i*\moduleysep}` shifts everything down and causes a possible waste of space.

For the second and third stages the procedure is identical: what changes is the x coordinate. For example, in the second stage it is:

```
at +(\modulexsep,\n2)
```

while in the third stage:

```
at +(2*\modulexsep,\n2)
```

What is `\modulexsep`? It is simply a macro associated to the `module xsep` key responsible to customize the horizontal distance between the stages; it is defined as:

```
\pgfkeys{/tikz/.cd,%
  module xsep/.initial={3},%
  module xsep/.get=\modulexsep,%
  module xsep/.store in=\modulexsep,%
}%
```

In the `benes complete` networks, this procedure is applied to all the stages; their number is computed by means of (1), which is translated in code with:

```
\pgfmathtruncatemacro{\stages}{
 2*round(log2(\P))-1
}
```

3.2 The interconnections

The interconnection drawing part is the core of the library. S*a*-Ti*k*Z is designed not only to automatically draw all the interconnections between input and output ports of the modules, but also to provide a method to subsequently improve the final picture by adding, for example, paths representing the connections between an input and output port of the architecture.

3.2.1 Locating the ports

In order to draw the interconnections, the first thing to do is to locate the input and output ports: this phase takes into account the input parameters because, according to their values, the number of ports changes. Supposing that there are `2` input and `3` output ports in a module, the ports are placed at the same distance one with respect to the others; this distance is computed according to:

$$in_{dist} = 1/(2+1) \qquad out_{dist} = 1/(3+1) \quad (2)$$

The code locating the ports of the first module of the `clos snb` networks is:

```
% INPUTS MODULE 1
% the number of inputs module one
% is exactly \mone
\pgfmathsetmacro\roneintervalspace{
 1/(\mone+1)
 }
\foreach \roneinput
 [evaluate=\roneinput as \roneinterval
  using \roneintervalspace*\roneinput]
 in {1,...,\mone}
 \draw ($(r1-\i.north west)!\roneinterval!
 (r1-\i.south west)-(0.5*\pinlength,0)$)
 node[scale=0.1]
 (r1-\i-front input-\roneinput){}-
 ($(r1-\i.north west)!\roneinterval!
 (r1-\i.south west)$)
 node[circle,draw,scale=0.1]
 (r1-\i-input-\roneinput) {};

% OUTPUTS MODULE 1
% the number of outputs of module one is
% the number of modules stage 2 \rtwo
\pgfmathsetmacro\roneintervalspace{
 1/(\rtwo+1)
}
\foreach \roneoutput
 [evaluate=\roneoutput as \roneinterval
  using \roneintervalspace*\roneoutput]
 in {1,...,\rtwo}
 \node[circle,draw,scale=0.1]
  (r1-\i-output-\roneoutput)
  at($(r1-\i.north east)!\roneinterval!
  (r1-\i.south east)$)  {};
```

The macro `\roneintervalspace` is computed according to (2): input and output differ according to the design parameters which influence the values of `\mone` and `\rtwo` (recall that they are the number of input ports per module of the first stage and the number of modules in the second stage respectively). It is interesting to analyze how the ports are ultimately located: this part is computed by `evaluate=\roneoutput as \roneinterval using \roneintervalspace*\roneoutput` within the foreach. The expression simply says that each port, progressively enumerated from 1, is located at an increasing distance with respect to the first port: indeed, the macro `\roneinterval` is set to the product of the basic distance `\roneintervalspace` times the counter reproseting the current port `\roneoutput`. The third port is located down 3 times than the first one, for example. After this definition, `\roneinterval` is used within the position definition by exploiting a facility of the `calc` library: it allows to determine a new position in terms of the distance between two known positions.

Then, for each port to be deployed, nodes with styles `circle,draw,scale=0.1` and particular names are instantiated: the names allows to later access the ports. The name choice is not causal; for internal ports, comprising the input and output ports of the all stages, the name is in the form: `r`*stage number*-*module number*-`input-`*port number*; for example: `r1-2-input-1`.

Notice that, for input ports of the first stage and output ports of the third stage, the port locating phase is not sufficient, but it is necessary also to draw the pins (the lines going out from the modules); in the previous code indeed, it is possible to notice a `\draw` mechanism which is able to realize this. Moreover, the pins have proper names which identify their final point: `r`*stage number*-*module number*-`front output-`*port number*; for example: `r3-3-front output-2`.

3.2.2 Drawing the interconnections

Once the ports have been located, it is possible to draw the interconnections. They concern: output ports of the first stage, input and output ports of the second stage and input ports of the third stage.

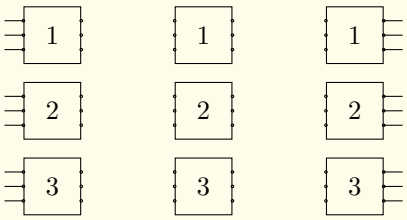
The code to create the interconnections for the `clos snb` networks is:

```
% Test if connections should be removed
\ifconnectiondisabled
 \relax
\else
 % DRAWING CONNECTIONS
 %% from r1 to r2
 \foreach \startmodule in {1,...,\rone}{
   \foreach \conn in {1,...,\rtwo}
     \draw(r1-\startmodule-output-\conn)
      -
      (r2-\conn-input-\startmodule);
 }
 %% from r2 to r3
 \foreach \startmodule in {1,...,\rthree}{
    \foreach \conn in {1,...,\rtwo}
      \draw(r3-\startmodule-input-\conn)
       -
       (r2-\conn-output-\startmodule);
 }
\fi
```

At first, a check if the connections should be displayed or not it is performed; for some purposes, perhaps, it is needed to have them hidden. Ti*k*Z allows also to define conditionals keys; indeed the `\ifconnectiondisabled` is defined as:

```
% disable the connections
\newif\ifconnectiondisabled%
\pgfkeys{/tikz/.cd,
 connections disabled/.is if=
  connectiondisabled
}%
\pgfkeys{/tikz/.cd,
 connections disabled/.default=.
 false
}%
```

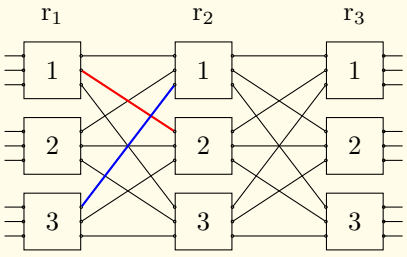
Therefore, it is sufficient to set to `true` the key `connections disabled` as per the example shown in figure 8.




```
\tikzset{module size=0.75cm,
pin length factor=0.5,
module ysep=1,
module xsep=2,
connections disabled=true}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
\node[clos rear]{};
\end{tikzpicture}
```

FIGURE 8: Rearrangeable Clos Network without connections

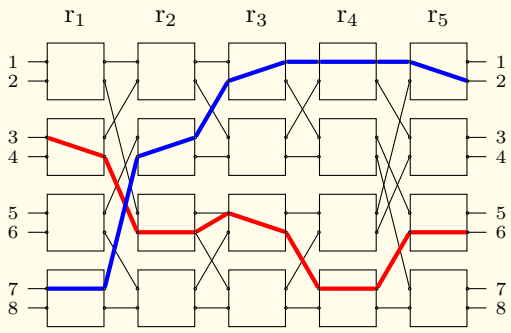
By default, `connections disabled` is set to `false`, so the drawing procedure can start. For each start module in the starting stage and for each output port this module has, a connection is established to the proper ending module of the next stage. Since during the location of the ports, they have been placed according to the input parameters, everything works with a simple trick: the starting module is identified by `r1-\startmodule-output-\conn` while the ending module by `r2-\conn-input-\startmodule`). In this way, correctly, `r1-1-output-2` is connected to `r2-2-input-1` and `r1-3-output-1` is connected to `r2-1-input-3` as per figure 9.




```
\tikzset{module size=0.75cm,module ysep=1,
pin length factor=0.5,module xsep=2,}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
\node[clos rear]{};
\foreach \x in {1,...,3}{
 \node[above=0.1cm of r\x-1] {r$_\x$};
}
\draw[red,thick] (r1-1-output-2)-
 (r2-2-input-1);
\draw[blue,thick] (r1-3-output-1)-
 (r2-1-input-3);
\end{tikzpicture}
```

FIGURE 9: Highlighting connections in a Clos Network

The procedure explained so far works for `clos snb`, `clos rear` and `benes` networks: for `benes`




```
\tikzset{module size=0.75cm,
  pin length factor=0.5,
  module ysep=1,
  module xsep=2,
}
\begin{tikzpicture}
\node[benes complete={
 module label opacity=0
 }]{};

% Labels: input ports
\newcounter{port}\setcounter{port}{0}
\foreach \startmodule in {1,...,4}{
 \foreach \port in {1,...,2}{
  \stepcounter{port}
  \node[left] at
  (r1-\startmodule-front input-\port)
  {\scriptsize{\theport}};
  }
}

% Labels: output ports
\setcounter{port}{0}
\foreach \startmodule in {1,...,4}{
 \foreach \port in {1,...,2}{
  \stepcounter{port}
  \node[right] at
  (r5-\startmodule-front output-\port)
  {\scriptsize{\theport}};
  }
}

% Labels: modules
\foreach \x in {1,...,5}{
 \node[above=0.1cm of r\x-1] {r$_\x$};
}

% Connection paths
\draw[red,ultra thick](r1-2-input-1)-
(r1-2-output-2)-(r2-3-input-2)-
(r2-3-output-2)-(r3-3-input-1)-
(r3-3-output-2)-(r4-4-input-1)-
(r4-4-output-1)-(r5-3-input-2)-
(r5-3-output-2);
\draw[blue,ultra thick](r1-4-input-1)-
(r1-4-output-1)-(r2-2-input-2)-
(r2-2-output-1)-(r3-1-input-2)-
(r3-1-output-1)-(r4-1-input-1)-
(r4-1-output-1)-(r5-1-input-1)-
(r5-1-output-2);
\end{tikzpicture}
```

FIGURE 10: Adding elements to the architecture

complete networks things are a bit different. Indeed, in order to draw the interconnections for that architecture the "dBnc" algorithm has been developed: before, to the best of my knowlegde, no automatic tools were able to perform such a task.

The algorithm takes into account the symmetric property of the Benes Networks and defines specific rules which are able to deal with the network size and the variable number of stages. In particular, the target is achieved thanks to the concepts of *module applicability range* and *repeatability* of the rules.

One of the peculiarities of the algorithm is that it needs to determine whether a port number is odd or even: Ti*k*Z itself does not provide a function within its mathematical engine. Thus, S*a*-Ti*k*Z defines a macro `\pgfmathisodd` able to perform the check and to store the result into another macro:

```
\newcommand*{\pgfmathisodd}[2]{
 \pgfmathparse{mod(#1,2)}
 \pgfmathtruncatemacro\res\pgfmathresult
 \global\expandafter\edef\csname
 #2\endcsname{\res}
}
```

An example of usage:

```
\pgfmathisodd{32}{output}
\ifnum\output=1
  \node{\output};
\fi
```

3.3 Adding elements

By means of the names of the modules' ports and the pins it is possible to add elements to the architecture; these elements could be:

- port labels;
- connection paths within the architecture;
- stage labels.

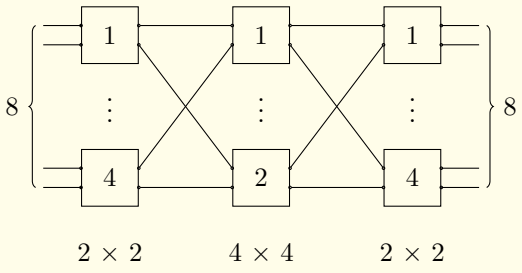
A complete example is shown in figure 10.

4 Network examples

This section takes into account how to create example pictures of switching architectures; the main keys devoted to the purpose are:

- `clos snb example`;
- `clos rear example`;
- `clos example with labels`.

With the first two keys it is possible to draw examples showing the design parameters. Although there are no equivalent keys for Benes Networks, it is even possible to represent them by simply selecting in the proper way the design parameters; an example is figure 11.




```
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
\node[clos rear example] {};
\end{tikzpicture}
```

Figure 11: Benes Network Example

An example of Clos Network, instead, is shown in figure 12.

In these drawings the number of modules per stage is fixed; in the code, this simplification is evident looking at how the modules are placed. For example, the modules in the third stage are located with the code:

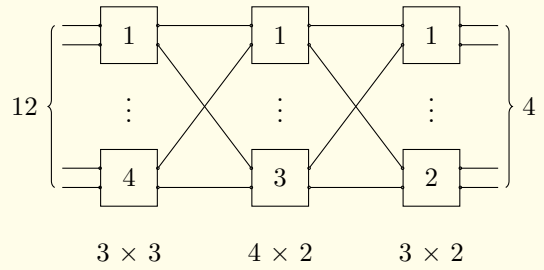
```
\node[module,#1,module opacity]
 (r3-1) at (2*\modulexsep,0) {1};
\node[below of=r3-1,yshift=0.75ex]
 (r3-dots) {\vdots};
\node[module,#1,module opacity,
 below of=r3-dots]
 (r3-2) {\rthree};
```

No particular methods are needed: simple Ti*k*Z nodes are drawn with the proper styles, described in subsection 3.1. Also the interconnection phase is more simple: just two input/output ports per module are needed, but, for simplicity in developing the code, the mechanism used is still the one described in subsection 3.2. It is interesting to analyze how the braces comprising the input and output ports are realized: thanks to the possibility to access to the pin names, `front input` and `front output` port, the braces are added by means of the `decorations.pathreplacing`.

The definition is:

```
\draw[decorate,decoration={brace}]
 ($(r1-2-front input-2)-(0.1,0)$)-
 ($(r1-1-front input-1)-(0.1,0)$)
 node[midway,
  left=0.1cm,
  set math mode labels]
 {\Nlabel};
\draw[decorate,decoration={brace}]
 ($(r3-1-front output-1)+(0.1,0)$)-
 ($(r3-2-front output-2)+(0.1,0)$)
 node[midway,
  right=0.1cm,
  set math mode labels]
 {\Mlabel};
```

The `calc` library helps in positioning the start and ending point of the brace a little shifted with



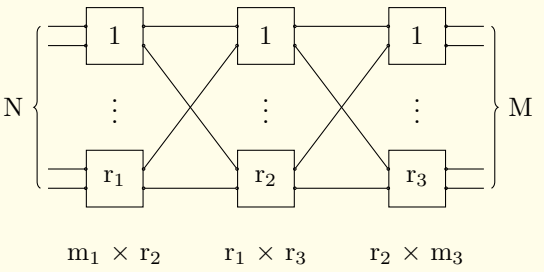

```
\begin{tikzpicture}[N=12,M=4,r1=4,r3=2]
\node[clos rear example]{};
\end{tikzpicture}
```

FIGURE 12: Rearrangeable Clos Network Example

respect to the input and output ports; in order to display the label, a node has been introduced with the options:

- `midway` to put it in the middle of the path;
- `left/right` to make the label not overlap the brace;
- `set math mode labels` is a key discussed later.

With respect to the keys `clos snb example` and `clos rear example`, the `clos example with labels` key differs only in what it prints: generic labels rather than the design parameters. In order to customize the labels, S*a*-Ti*k*Z defines several keys with the syntax: `<key> label` where `<key>` is one of the design parameters; for example, the key `N label` allows to customize the label defining the total number of input ports of the first stage. The figures 1 and 4 are drawn in this way.

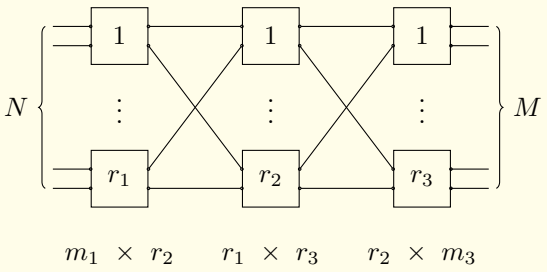



```
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
\node[clos example with labels] {};
\end{tikzpicture}
```

FIGURE 13: Clos Network Example with labels

When using `clos example with labels` networks, the input parameter definition is useless; for instance, figure 13 shows that those definitions are simply ignored.

What it is also possible to see from figure 13, is that the letters are displayed in roman; to have them in italic, the key `set math mode labels` is of help as per figure 14. The key `set math mode`




```
\begin{tikzpicture}[
 set math mode labels=true]
\node[clos example with labels] {};
\end{tikzpicture}
```

FIGURE 14: Clos Network Example with labels in italic

`labels` is a boolean key by default set to false; its definition is a bit complex:

```
\tikzset{math mode labels/.style={%
   execute at begin node=$,%
   execute at end node=$,%
  }
}
\pgfkeys{/tikz/.cd,%
   use math mode labels/.is choice,%
   use math mode labels/true/.style={%
    math mode labels
   },%
   use math mode labels/false/.style={},%
}%

\tikzset{set math mode labels/.style={%
 use math mode labels=#1,%
 },%
 set math mode labels/.default=false,%
}
```

To understand the definition it is necessary to start from the key `math mode labels`; it is a key that before and after the text node *physically* adds a pair of $ thanks to `execute at begin node=$` and `execute at end node=$`: in such a way, the text of the node is rendered in math-mode. Subsequently, a boolean key `use math mode labels` is defined: when the choice is set to `true` then the key `math mode labels` becomes active. On top of `use math mode labels` there is, finally, `set math mode labels`: it receives an argument passed to `use math mode labels` and this argument by default is set to `false`. To summarize, when `set math mode labels=true` is set, `true` is passed to `use math mode labels`; this makes `math mode labels` active. As a consequence the labels are rendered in italic.

5 Conclusions

The article presented S*a*-Ti*k*Z by discussing the code of the library and showing several examples.

This is an example of how the Ti*k*Z code can be used for practical purposes providing an easy and manageable tool thanks to the key-interface. Indeed, one of the aims of the library, is to give to the students the possibility to verify the correctness of their exercises; as an instrument, it has been practically used in the academic course of Switch and Router Architectures held by prof. Paolo Giaccone in Politecnico di Torino.

References

FIANDRINO, C. (2013). *Sa-TikZ*. URL `http://www.ctan.org/pkg/sa-tikz`. Available typing in the terminal `texdoc sa-tikz`

▷ Claudio Fiandrino
`claudio dot fiandrino at gmail dot com`