S*a*-Ti*k*Z*

Claudio Fiandrino
claudio.fiandrino@gmail.com

December 31, 2012

Contents

Introduction

The S*a*-Ti*k*Z library helps in drawing *switching-architectures*. In particular, one of its aims, is to help students to verify if their exercises are correct, but it could also help teachers in preparing lecture notes. The repository of the library is https://github.com/cfiandra/Sa-TikZ.

*This package has version number *v*0.4 of December 31, 2012; it is released under and subject to the LaTeX Project Public License (LPPL).

The S*a*-Ti*k*Z library can be loaded by means of:

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

and in this case you should also load manually:

```
\usepackage{tikz}
```

or by means of:

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

In both cases the libraries `calc`, `positioning` and `decorations.pathreplacing` are loaded automatically and in the latter case also the Ti*k*Z package is loaded.

The version *v*0.4 provides a way to draw Clos Networks Strictly-non-Blocking (snb) and Rearrangeable (rear) and Benes Networks; moreover, there is the possibility to fully customize the aspect of the network drawn starting from the dimensions of module, their distance and the font used. Finally, S*a*-Ti*k*Z let the user to draw connections among the stages by accessing the single ports of the modules.

1 Basic usage

The simplest use of the package is to define a

`\node`
Basic command definition.

with one of the following options

`/tikz/clos snb` (no value)
Option for drawing a Clos Network Strictly-non-Blocking.

`/tikz/clos rear` (no value)
Option for drawing a Clos Network Rearrangeable.

`/tikz/benes` (no value)
Option for drawing a Benes Network.

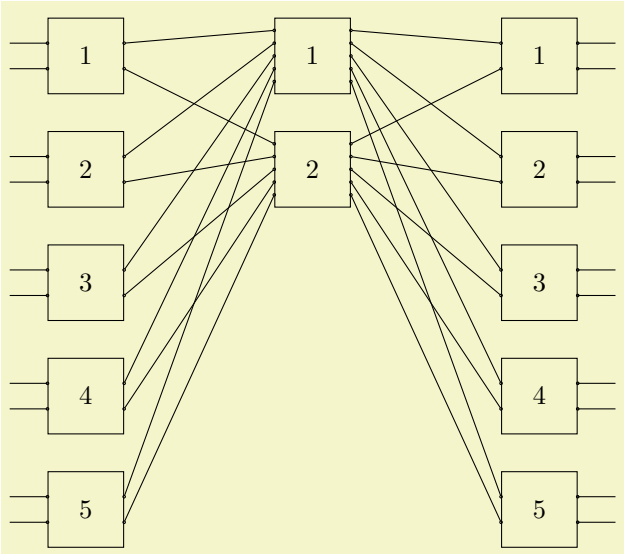
`/tikz/benes complete` (no value)
Option for drawing a Benes Network with the lowest level of recursion.

inside a `tikzpicture` environment:

```
\begin{tikzpicture}[⟨options⟩]
  ⟨environment contents⟩
\end{tikzpicture}
```

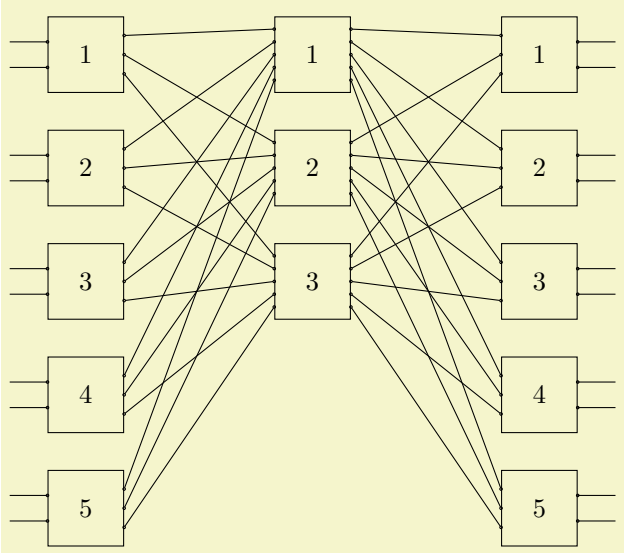
1.1 Examples of Clos Networks

The following example shows a Rearrangeable Clos Network.




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

The following example shows a Strictly-non-Blocking Clos Network.




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

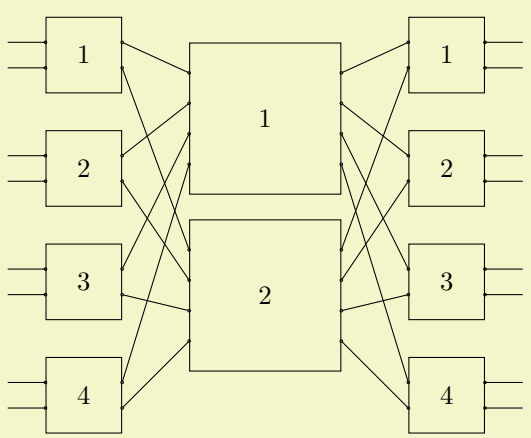
Notice from the examples that automatically the library is able to compute the constraints that define a Clos Network to be Strictly-non-Blocking or Rearrangeable. Moreover, the network drawn is characterized by:

- the first stage with:
  - a number of modules equal to 5;
  - each one with two input ports;
- the last stage with:
  - a number of modules equal to 5;
  - each one with two output ports.

Each module of the network is numbered according to the stage it belongs to.

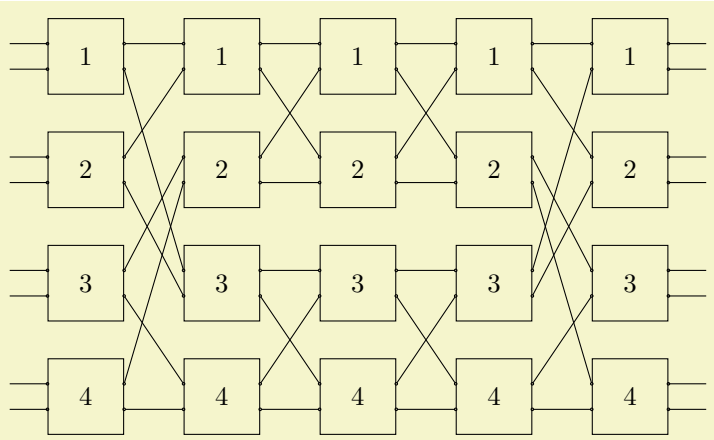
1.2 Examples of Benes Networks

The simplest example of a Benes Network:




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

is a Benes Network in which there are 8 input and output ports. To draw a Benes Network in which all modules are visible, the key `benes complete` should be used rather than the `benes` key. An example:




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

The algorithm in which the internal connections of the `benes complete` networks are drawn is explained in detail in the appendix A.

2 The options

2.1 Designing choices

This subsection illustrates which are the parameters that could be customized to draw Clos and Benes Networks. In particular:

- Clos Networks are analysed in 2.1.1;
- Benes Networks are analysed in 2.1.2.

In each part the keys will be presented and simple examples will be provided.

2.1.1 Clos Networks

The two first important design parameters are the total number of input ports of the first stage and the total number of output ports of the last stage. These two parameters could be modified by means of:

`/tikz/N={`⟨*value*⟩`}` (no default, initially `10`)

This is the number of total input ports in the first stage.

`/tikz/M={`⟨*value*⟩`}` (no default, initially `10`)

This is the number of total output ports in the last stage.

Usually, a second design parameter is the number of modules present in the first and last stage. S*a*-Ti*k*Z defines:

/tikz/r1={⟨*value*⟩} (no default, initially 5)

This is the number of total input ports in the first stage.

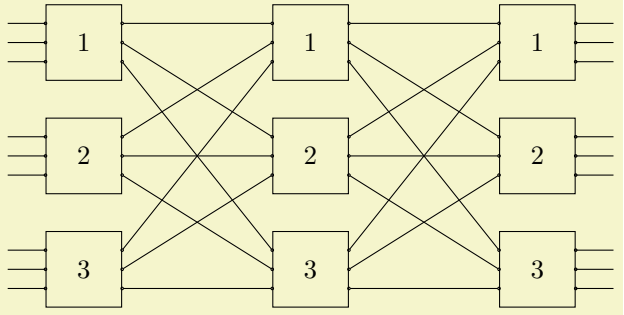
/tikz/r3={⟨*value*⟩} (no default, initially 5)

This is the number of total output ports in the last stage.

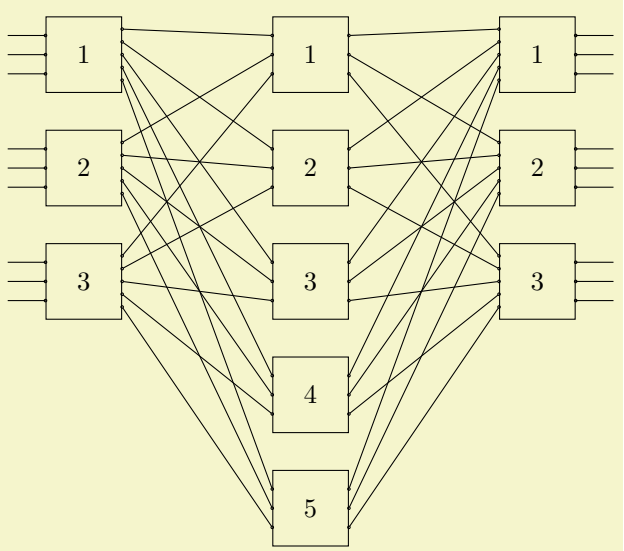
The two design parameters provide the number of ports of each module:

$$m_1 = \frac{N}{r1} \qquad\qquad m_3 = \frac{M}{r3}$$

Some examples considering N=9, r1=3, M=9 and r3=3.

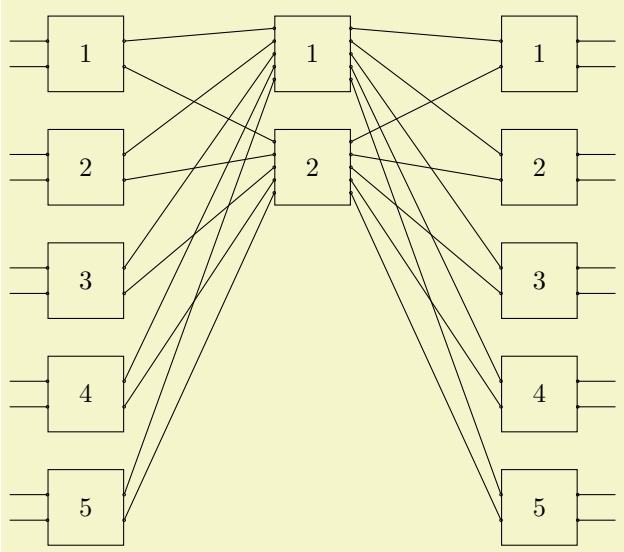



```
\begin{tikzpicture}
    \node[N=9,r1=3,M=9,r3=3,clos rear] {};
\end{tikzpicture}
```




```
\begin{tikzpicture}
    \node[N=9,r1=3,M=9,r3=3,clos snb] {};
\end{tikzpicture}
```

Notice a very important thing: the type of the architecture should be loaded *after* all the design choices in case they have been set in the `\node`; indeed, if you do not respect this constraint you will end up with an architecture with default values. For example:




```
\begin{tikzpicture}
    \node[clos rear,N=9,r1=3,M=9,r3=3] {};
\end{tikzpicture}
```

2.1.2 Benes Networks

Benes Networks are Clos Rearrangeable Networks composed of 2×2 modules, so as design choice it just possible to select which is the number of input/output ports:

`/tikz/P={`⟨*value*⟩`}` (no default, initially `8`)

This is the number of total input/output ports in the first/third stage.

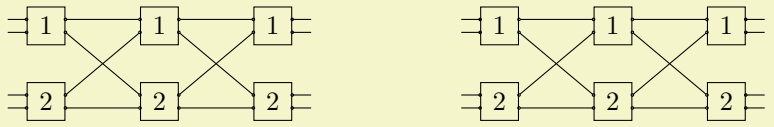
Notice that `P` could assume values

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

and the user is responsible to correctly set this parameter.

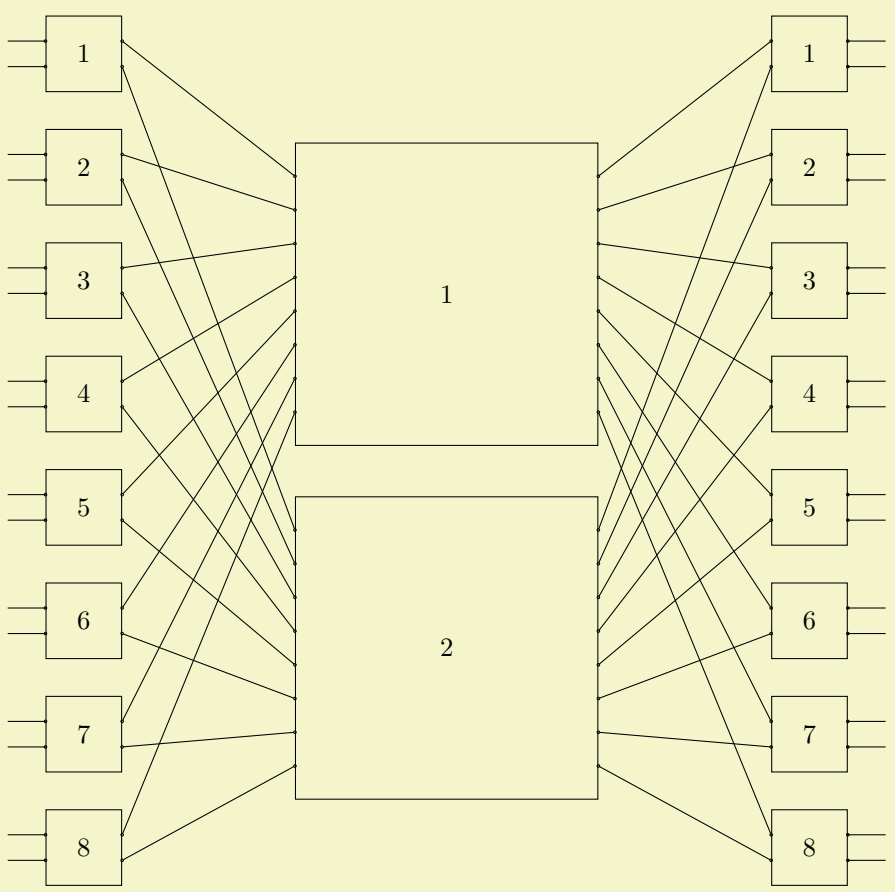
For low values of p there are no problems in visualizing the network, but as p increases the user should take care of the modules' dimension and the modules' separation (vertical and horizontal): they could be customized as explained in the subsection 2.2. Actually, for `benes complete` networks, the number of p is crucial: when it is above 7, thus for networks bigger than 128×128, PGF can not properly work due to internal limitations.

Notice that actually, for P=4 the benes network and the benes complete network are indistinguishable:




```
\begin{tikzpicture}
\tikzset{module size=0.5cm,
 pin length factor=0.5,
 module ysep=1}
 \node[P=4,benes] {};
 \begin{scope}[xshift=6cm]
     \node[module xsep=2.5,P=4,benes complete]{};
 \end{scope}
\end{tikzpicture}
```

Here is an example of Benes Network with P=16:



```
\begin{tikzpicture}
    \node[P=16,benes] {};
\end{tikzpicture}
```

It holds the same thing already said for Clos Networks: set the parameter `P` before declaring the `\node` be a `Benes` Network.

2.2 Output customization

This subsection focuses on how to customize the aspect of the drawing.

`/tikz/module size=`{⟨*value*⟩} (no default, initially `1cm`)

This option allows to set the module dimension.

`/tikz/module ysep=`{⟨*value*⟩} (no default, initially `1.5`)

This option allows to set the vertical module distance factor.

`/tikz/module xsep=`{⟨*value*⟩} (no default, initially `3`)

This option allows to set the horizontal module distance factor.

`/tikz/module label opacity=`{⟨*value*⟩} (no default, initially `1`)

This option allows to mask the module label when the ⟨*value*⟩ is set to 0.

`/tikz/pin length factor=`{⟨*value*⟩} (no default, initially `1`)

This option allows to reduce/increase the length of the pins drawn in input/output. Use a ⟨*value*⟩ [0,1] to reduce the length or, viceversa, a ⟨*value*⟩ greater than 1 to increase the length.

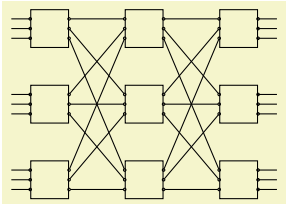
`/tikz/module font=`{⟨*font commands*⟩} (default `\normalfont`)

This option sets the font used for module labels. The ⟨*font commands*⟩ that could be used are those ones related to the font size (i.e. `\Large`) and font shape (i.e `\itshape`).

`/tikz/connections disabled=true|false` (default `false`)

This option, not active by default `connections disabled/.default=false`, allows to remove the connections between the stages when set to true. Beware: this option is valid only for `clos snb`, `clos rear`, `benes` and `benes complete` networks.

The following example shows a Rearrangeable Clos Network with some options modified. Notice that the `module label opacity` should be given as parameter of the desired network.



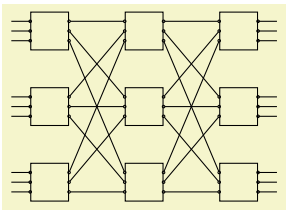
```
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
    \node[module size=0.5cm,pin length factor=0.5,
        module ysep=1, module xsep=1.25,
        clos rear={module label opacity=0}] {};
\end{tikzpicture}
```

The options could also be introduced with the standard Ti*k*Z syntax:

`\tikzset{`⟨*options*⟩`}`

Command that process the various ⟨*options*⟩: they should be provided separated by a comma.

Therefore, the previous example could be modified into:

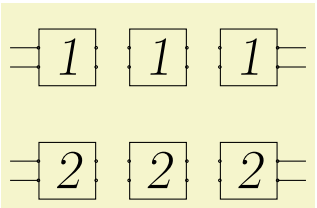


```
\tikzset{module size=0.5cm,pin length factor=0.5,
        module ysep=1, module xsep=1.25}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
    \node[clos rear={module label opacity=0}] {};
\end{tikzpicture}
```

It is also possible to declare styles to set some options for later use: this helps to keep the code clean especially when the same options are re-used several times; an example:

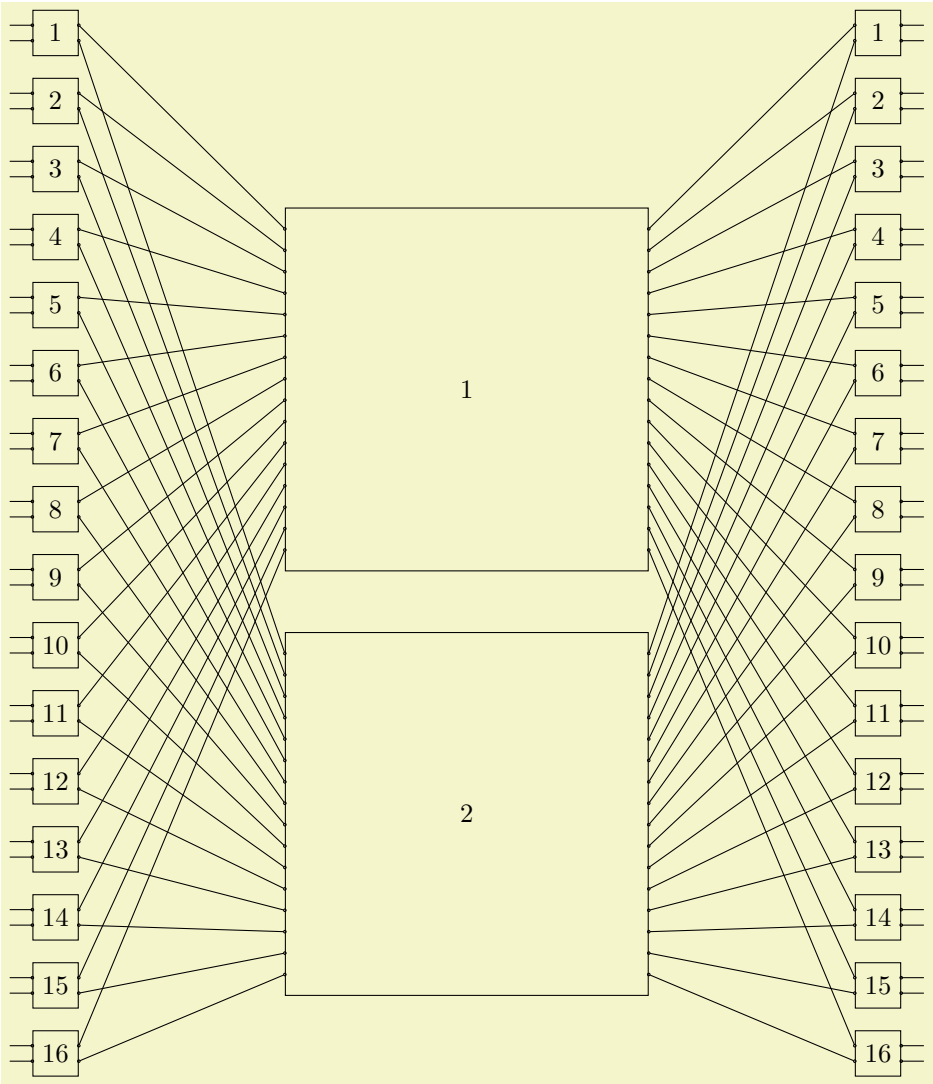
```
\tikzset{module size definition/.style={
  module size=0.75cm,
  pin length factor=0.75,
  module xsep=2,
  module ysep=2,
  }
}
\tikzset{module size definition,
 P=16,
}
\begin{tikzpicture}
    \node[benes] {};
\end{tikzpicture}
```

Here is a Benes Network 4×4 with an extremely large font size for the module labels with the connections disabled:



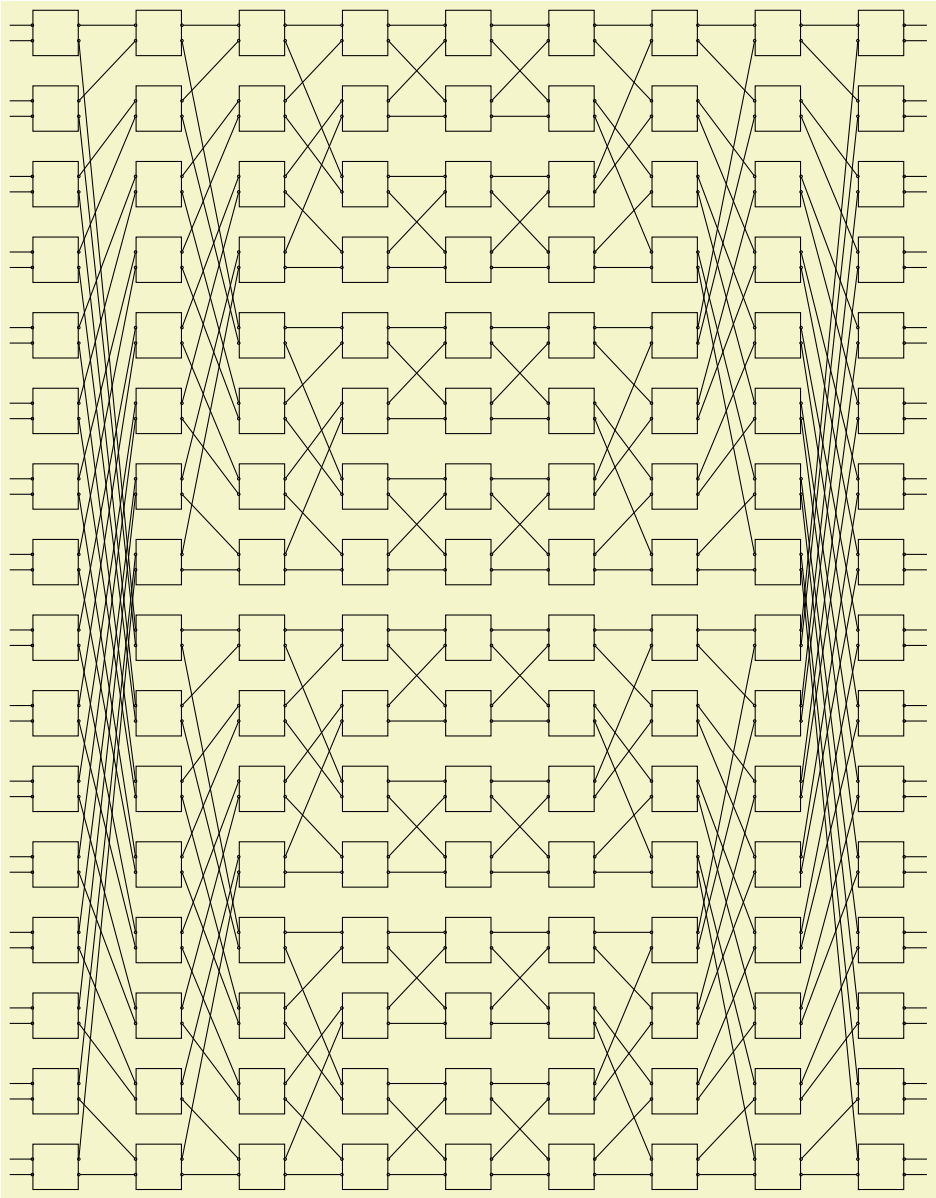

```
\tikzset{my style/.style={
  module size=0.75cm,
  pin length factor=0.75,
  module xsep=2,
  }
}
\tikzset{my style, P=4,
 module font=\huge\slshape,
 connections disabled=true
}
\begin{tikzpicture}
    \node[benes complete] {};
\end{tikzpicture}
```

An example of Benes Network 32×32:




```
\tikzset{module size=0.6cm,pin length factor=0.6,
         module ysep=0.9, module xsep=1.7}
\begin{tikzpicture}[P=32]
    \node[benes] {};
\end{tikzpicture}
```

and its complete form:



```
\tikzset{module size=0.6cm,pin length factor=0.6,
         module ysep=1, module xsep=2.275}
\begin{tikzpicture}[P=32]
    \node[benes complete={module label opacity=0}] {};
\end{tikzpicture}
```

3 Advanced usage

In this section some more advanced examples are shown.

3.1 Identifying front input/output ports

In this subsection it is shown how to reference the front input and output ports for the first and last stage. Each front input port could be accessed by means of:

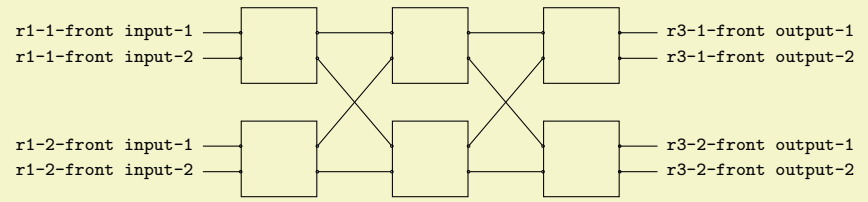
`r1-module number-front input-port number`; example:
`r1-1-front input-1`;

Each front output port could be accessed by means of:

`r3-module number-front output-port number`; example:
`r3-1-front output-1`;

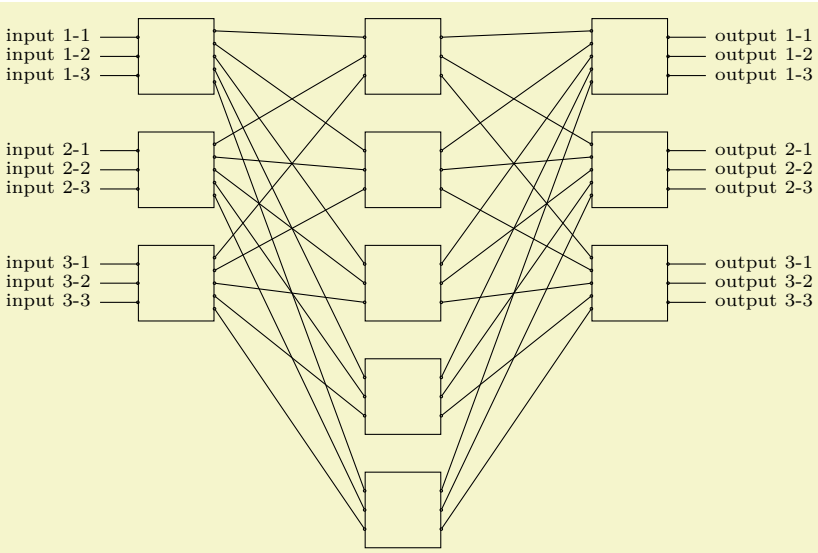
Noticed that the first stage is always 1, but the last stage may be different from 3 in case the `benes complete` network is drawn. Errors will occur in case the last stage number is not correct and the user is responsible for the correct setting.

A simple example with a Rearrangeable Clos network of 4 input and output ports; the first stage and the last one have both 2 modules.




```
\begin{tikzpicture}[module xsep=2]
  \node[N=4,r1=2,M=4,r3=2,clos rear={module label opacity=0}] {};
  \foreach \name
    in {r1-1-front input-1,r1-1-front input-2,
        r1-2-front input-1,r1-2-front input-2}
    \node[left] at (\name) {\scriptsize{\texttt{\name}}};
  \foreach \name
    in {r3-1-front output-1,r3-1-front output-2,
        r3-2-front output-1,r3-2-front output-2}
    \node[right] at (\name) {\scriptsize{\texttt{\name}}};
\end{tikzpicture}
```

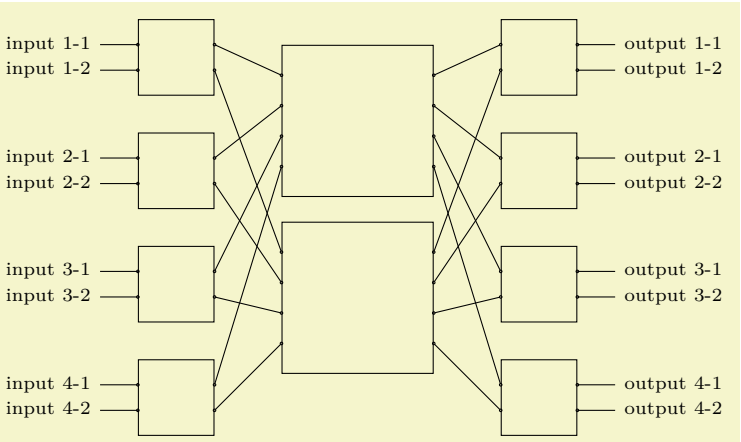
The following is a Strictly-non-Blocking Clos network of 9 input and output ports in which the first and last stage have 3 modules each one.




```
\begin{tikzpicture}
    \node[N=9,r1=3,M=9,r3=3,clos snb={module label opacity=0}] {};

\foreach \startmodule in {1,...,3}{
\foreach \port in {1,...,3}
 \node[left] at (r1-\startmodule-front input-\port)
 {\scriptsize{input \startmodule-\port}};
}
\foreach \startmodule in {1,...,3}{
\foreach \port in {1,...,3}
 \node[right] at (r3-\startmodule-front output-\port)
 {\scriptsize{output \startmodule-\port}};
}
\end{tikzpicture}
```

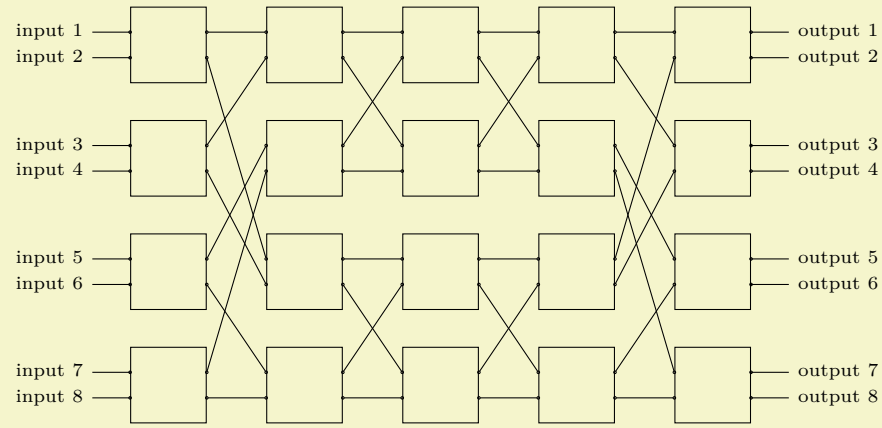
The same applies also for Benes Networks:



```
\begin{tikzpicture}
    \node[benes={module label opacity=0}] {};

\foreach \startmodule in {1,...,4}{
\foreach \port in {1,...,2}
 \node[left] at (r1-\startmodule-front input-\port)
 {\scriptsize{input \startmodule-\port}};
}
\foreach \startmodule in {1,...,4}{
\foreach \port in {1,...,2}
 \node[right] at (r3-\startmodule-front output-\port)
 {\scriptsize{output \startmodule-\port}};
}
\end{tikzpicture}
```

and for the complete form:




```
\begin{tikzpicture}
    \node[benes complete={module label opacity=0}] {};
\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{input \theport}};
}
\setcounter{port}{0}
\foreach \startmodule in {1,...,4}{
\foreach \port in {1,...,2}
\stepcounter{port}
 \node[right] at (r5-\startmodule-front output-\port)
 {\scriptsize{output \theport}};
}
\end{tikzpicture}
```

Notice that in this case to access the `front output` ports, the stage number correct is 5 and not 3 as usual.

3.2 Identifying input/output ports per module

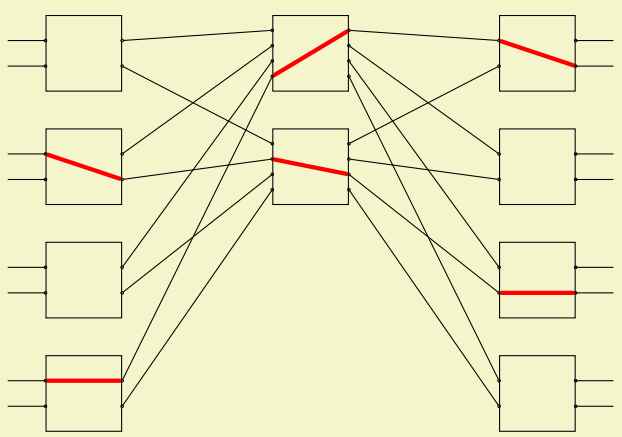
It is also possible to access, for each module of each stage, its input and output ports. The syntax is similar to the one used for the front input and output ports; each input port could be accessed by means of:

`rstage number-module number-input-port number`; example: `r1-1-input-1`;

Each output port could be accessed by means of:

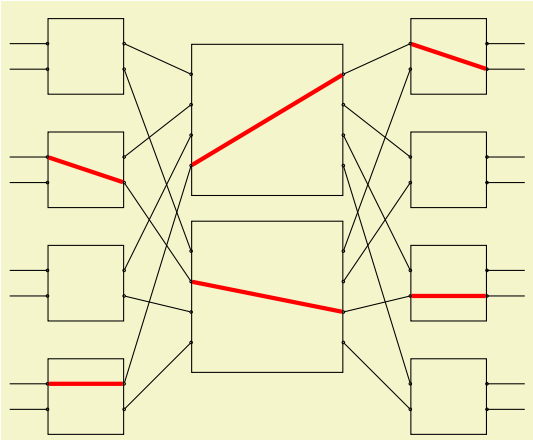
`rstage number-module number-front output-port number`; example: `r2-1-output-1`;

This allows to derive connections from the first stage to the last stage. Here is an example.



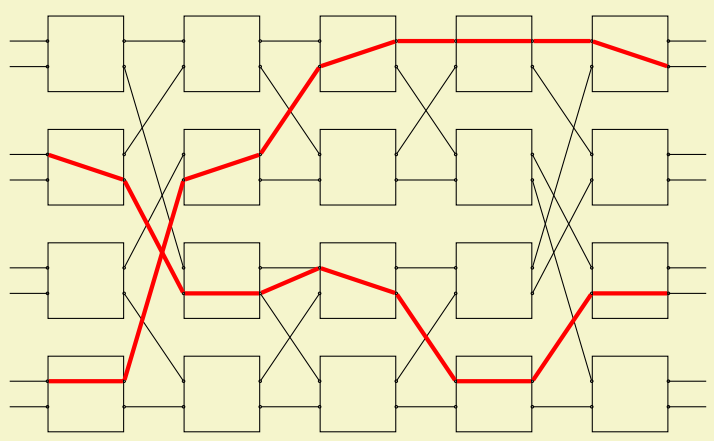
```
\begin{tikzpicture}
    \node[N=8,r1=4,M=8,r3=4,clos rear={module label opacity=0}] {};
    \draw[red,ultra thick](r1-2-input-1)-(r1-2-output-2)
    (r2-2-input-2)-(r2-2-output-3)
    (r3-3-input-2)-(r3-3-output-2);
   \draw[red,ultra thick](r1-4-input-1)-(r1-4-output-1)
    (r2-1-input-4)-(r2-1-output-1)
    (r3-1-input-1)-(r3-1-output-2);
\end{tikzpicture}
```

Similarly, an example in a Benes Network:



```
\begin{tikzpicture}
    \node[benes={module label opacity=0}] {};
    \draw[red,ultra thick](r1-2-input-1)-(r1-2-output-2)
    (r2-2-input-2)-(r2-2-output-3)
    (r3-3-input-2)-(r3-3-output-2);
   \draw[red,ultra thick](r1-4-input-1)-(r1-4-output-1)
    (r2-1-input-4)-(r2-1-output-1)
    (r3-1-input-1)-(r3-1-output-2);
\end{tikzpicture}
```

and in its complete form:



```
\begin{tikzpicture}
    \node[benes complete={module label opacity=0}] {};
    \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[red,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}
```

4 Didactic purposes

To quickly draw a Clos Network it is possible to exploit:

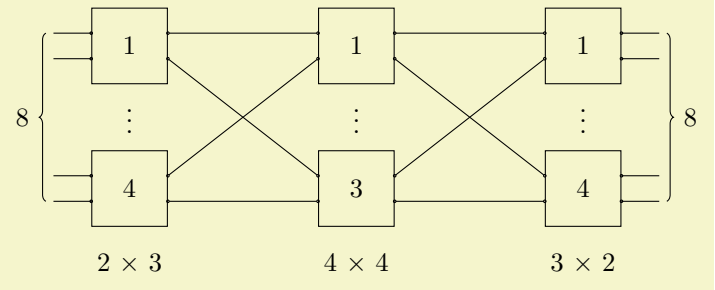
`/tikz/clos snb example` (no value)

Option for quickly drawing a Clos Network Strictly-non-Blocking.

`/tikz/clos rear example` (no value)

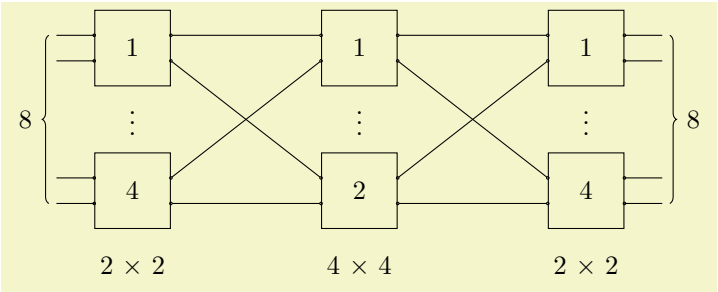
Option for quickly drawing a Clos Network Rearrangeable.

In this way the network is not seen in its whole complexity, but it is synthetically depicted. An example of a Strictly-non-Blocking Clos Network drawn with this approach:




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

Similarly, an example of a Rearrangeable Clos Network:




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

The networks drawn, automatically display the values at which the input parameters `N`, `M`, `r1` and `r3` have been set. However, to let the user to have the possibility of deploying labels rather than the input parameter values, the following option is available:

`/tikz/clos example with labels` (no value)

Option for quickly drawing a Clos Network with custom labels.

The labels could be customized by means of:

`/tikz/N label={`⟨*value*⟩`}` (default `N`)

This options sets the label representing the total number of ports in the first stage.

`/tikz/r1 label={`⟨*value*⟩`}` (default $\mathtt{r}_1$)

This options sets the label representing the number of modules in the first stage.

`/tikz/m1 label={`⟨*value*⟩`}` (default $\mathtt{m}_1$)

This options sets the label representing the number of ports per module in the first stage.

`/tikz/r2 label={`⟨*value*⟩`}` (default $\mathtt{r}_2$)

This options sets the label representing the number of modules in the second stage.

`/tikz/M label={`⟨*value*⟩`}` (default `M`)

This options sets the label representing the total number of ports in the last stage.

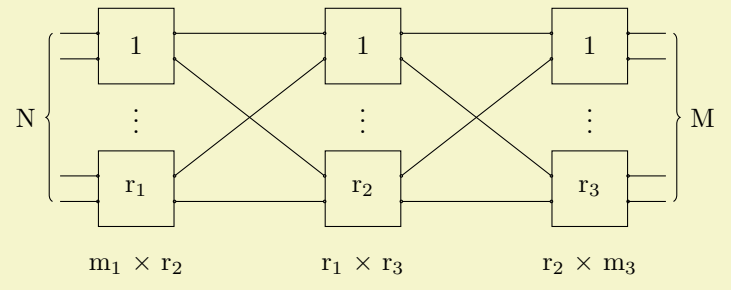
`/tikz/r3 label={`⟨*value*⟩`}` (default $\mathtt{r}_3$)

This options sets the label representing the number of modules in the last stage.

`/tikz/m3 label={`⟨*value*⟩`}` (default m_3)

This options sets the label representing the number of ports per module in the last stage.

An example with the default values:



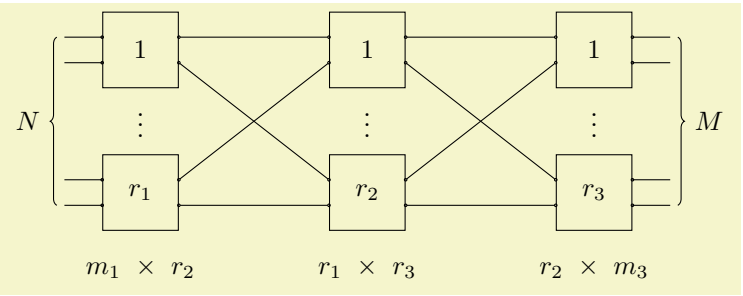

```
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
    \node[clos example with labels] {};
\end{tikzpicture}
```

To have automatically all labels in math mode, use:

`/tikz/set math mode labels=true|false` (default `false`)

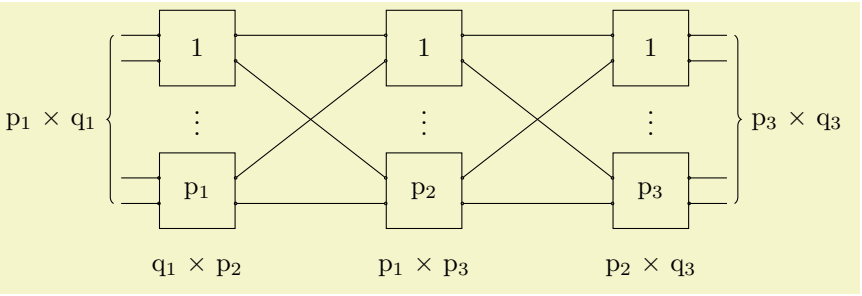
This option is normally disabled `set math mode labels/.default=false`; to ensure labels be set completely in math mode is sufficient set `set math mode labels=true` before the type of the network.

An example:




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

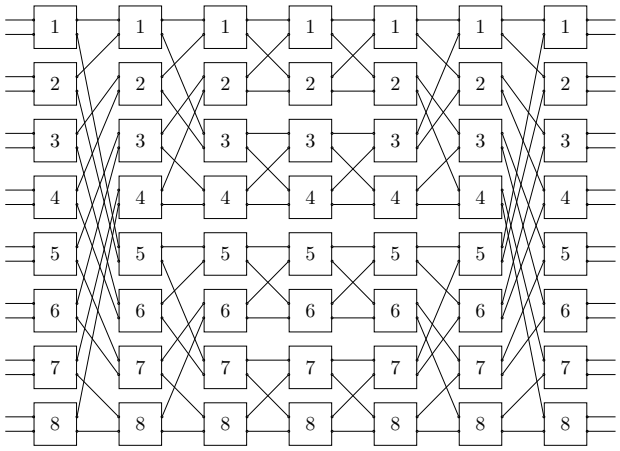
This example, instead, represents a `clos example with labels` network with custom labels introduced by means of the `\tikzset` syntax.




```
\tikzset{N label={p$_1$ $\times$ q$_1$},M label={p$_3$ $\times$ q$_3$},
r1 label=p$_1$, m1 label=q$_1$, r2 label=p$_2$,r3 label=p$_3$, m3 label=q$_3$}
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
    \node[clos example with labels] {};
\end{tikzpicture}
```

A Benes complete internal connections algorithm

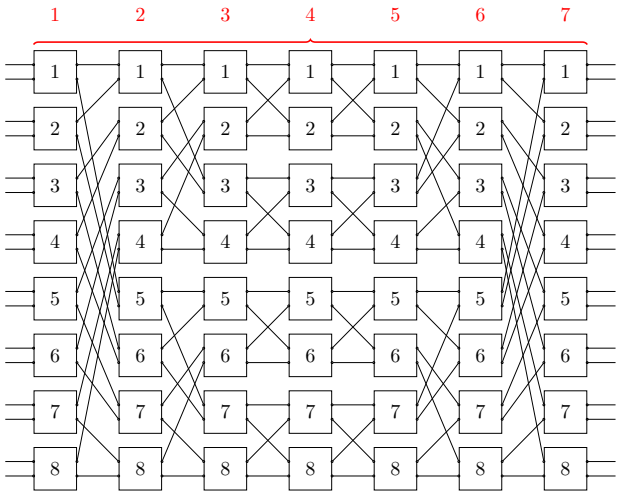
To explain how the connections of the `benes complete` networks are drawn, the following reference example will be considered:



The network is 16×16, thus the number of stages $\mathcal{S}$ is:

$$\mathcal{S} = 2\log_2 P - 1 \implies \mathcal{S}_{16} = 7$$

Indeed:

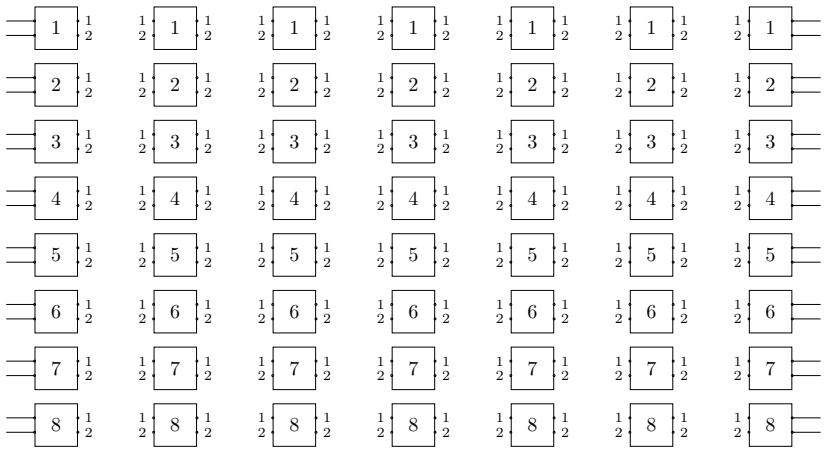


This parameter, therefore, allows to correctly draw all the modules of the network and, as it will be pointed out later better, its knowledge is important also to define the range of applicability of the algorithm. Notice the network symmetry: the connections from stage 1 to stage 4 are exactly the same from stage 7 to stage 4.

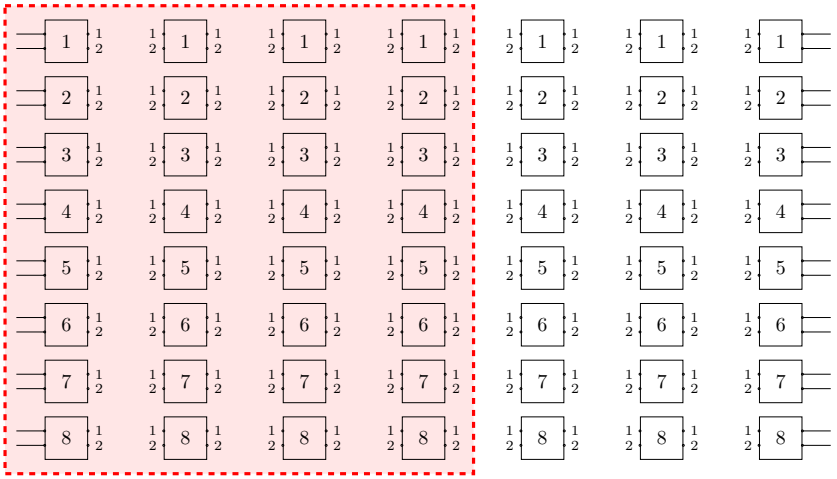
The first step is *labelling* modules and ports. S*a*-Ti*k*Z uses this philosophy:

- progressive numeration for modules of the same stage;
- progressive numeration for ports of the same module.

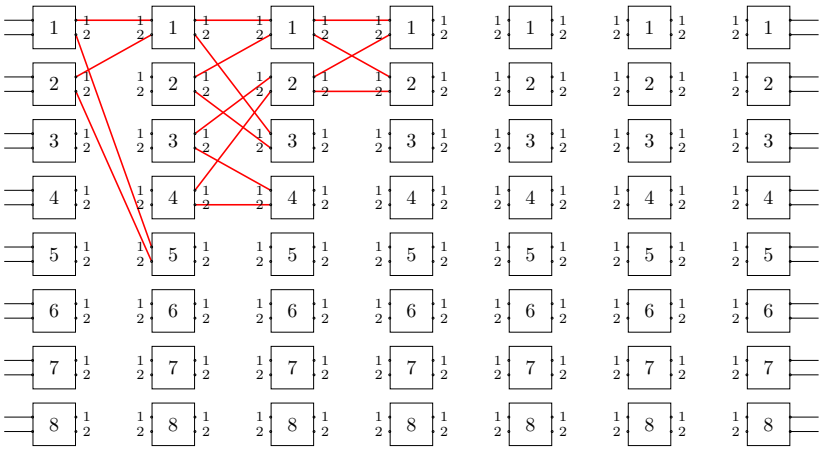
Thus:



Due to the network symmetry, at the beginning the attention will be focused only on the left side of the network, because for the right part things are dual:



Now, by drawing some connections, it is possible to find a common behaviour:



- if the start module st and the output port are odd (i.e. module 1, port 1), then it will be connected to

$$\text{end module} = \frac{st+1}{2}\ ,\ \text{port} = 1$$

- if the start module st is odd and the output port is even (i.e. module 1, port 2), then it will be connected to

$$\text{end module} = \frac{st+1+\gamma}{2}\ ,\ \text{port} = 1$$

- if the start module st is even and the output port is odd (i.e. module 2, port 1), then it will be connected to

$$\text{end module} = \frac{st}{2}\ ,\ \text{port} = 2$$

- if the start module st and the output port are even (i.e. module 2, port 2), then it will be connected to

$$\text{end module} = \frac{st+\gamma}{2}\ ,\ \text{port} = 2$$

What is the term γ? It is a corrective term that depends on the starting stage. Consider indeed the connection of the output port 2 of the module 1 for the first and the second stage:

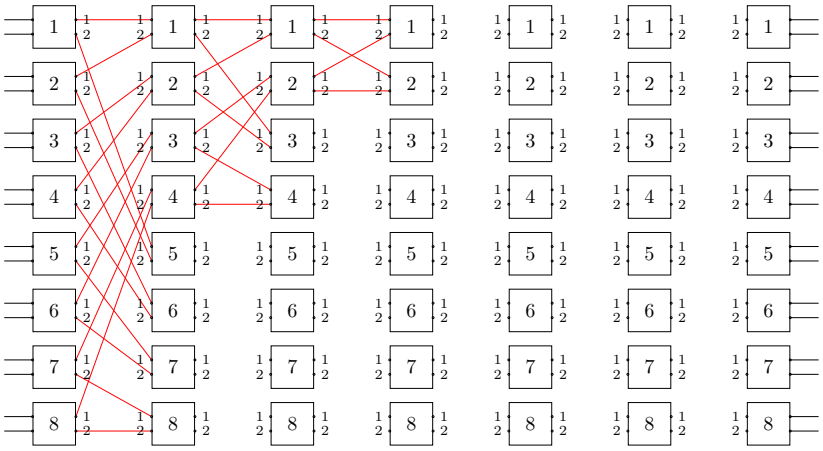
`r1-1-output-2` $\longrightarrow$ `r2-5-input-1`
`r2-1-output-2` $\longrightarrow$ `r3-3-input-1`

In the first case it points to module 5 while in the second case to module 3, thus in the first case $\gamma = 8$ and in the second case $\gamma = 4$. This suggest that γ is related

in some sense to the stage of the start module: in the example P=16 and there are 8 modules so the relation is

$$\gamma = \frac{P}{2^{stage}}$$

Following this strategy, however, allows to draw just part of the connections:



thus it is possible to claim that the algorithm has a *module applicability range* that ultimately depends on the stage:

- in the first stage it could be applied for all modules;
- in the second stage it could be applied for half of the modules;
- in the third stage it could be applied just for two modules.

But, in the first stage $\gamma = 8$, in the second stage $\gamma = 4$ and in the third stage $\gamma = 2$: this means that γ defines the *module applicability range*.

Notice now, that actually for the second stage and the third stage, the algorithm should be simply repeated:

- in the second stage 2 times;
- in the third stage 4 times.

The repetition ψ depends on the stage with this relation:

$$\psi = 2^{stage-1}$$

Now, to draw automatically all the connections, the algorithm should know which are the starting module and ending module of the *module applicability range* during the repetitions: for example, in the second stage, how to identify automatically the applicability range `1-4`, `5-8`? They could be defined as:

- starting module: $st_m = 1 + (\psi - 1) \cdot \gamma$;

- ending module: $end_m = (st_m + \gamma) - 1$.

Indeed for the second stage we have that $\gamma = 4$ and $\psi = 2 \implies \{1, 2\}$, thus there are two starting and ending modules:

- starting modules: $st_{m_1} = 1 + (1 - 1) \cdot 4 = 1$ and $st_{m_2} = 1 + (2 - 1) \cdot 4 = 5$;
- ending modules: $end_{m_1} = (1 + 4) - 1 = 4$ and $end_{m_2} = (5 + 4) - 1 = 8$.

Unfortunately, just the knowledge of the starting and ending modules per stage is not sufficient to reach the goal: this because the algorithm works and draws the connections perfectly when the module labels start with 1, but during the repetitions the new starting module labels are different, so the computation of the end connection point fails. This difference should be compensated with *shifts* of the ending modules that depend on the level of repetition. The rules are:

- if $\psi = 1$ (the algorithm works for all modules of the stage), then the ending module of the connection is computed as:
  - if the start module st and the output port are odd (i.e. module 1, port 1), then it will be connected to
  $$\text{end module} = \frac{st + 1}{2}\ ,\ \text{port} = 1$$
  - if the start module st is odd and the output port is even (i.e. module 1, port 2), then it will be connected to
  $$\text{end module} = \frac{st + 1 + \gamma}{2}\ ,\ \text{port} = 1$$
  - if the start module st is even and the output port is odd (i.e. module 2, port 1), then it will be connected to
  $$\text{end module} = \frac{st}{2}\ ,\ \text{port} = 2$$
  - if the start module st and the output port are even (i.e. module 2, port 2), then it will be connected to
  $$\text{end module} = \frac{st + \gamma}{2}\ ,\ \text{port} = 2$$
- if $\psi = 2$ (the algorithm should be repeated twice), then the ending module of the connection is computed as
  - if the start module st and the output port are odd (i.e. module 1, port 1), then it will be connected to
  $$\text{end module} = \frac{st + 1}{2} + \frac{\gamma}{2}\ ,\ \text{port} = 1$$

- if the start module st is odd and the output port is even (i.e. module 1, port 2), then it will be connected to

$$\text{end module} = \frac{st + 1 + \gamma}{2} + \frac{\gamma}{2}\ ,\ \text{port} = 1$$

- if the start module st is even and the output port is odd (i.e. module 2, port 1), then it will be connected to

$$\text{end module} = \frac{st}{2} + \frac{\gamma}{2}\ ,\ \text{port} = 2$$

- if the start module st and the output port are even (i.e. module 2, port 2), then it will be connected to

$$\text{end module} = \frac{st + \gamma}{2} + \frac{\gamma}{2}\ ,\ \text{port} = 2$$

- if $\psi > 2 \implies t = 3, \ldots, \psi$ (the algorithm should be repeated more than twice), then the ending module of the connection is computed as:
  - if the start module st and the output port are odd (i.e. module 1, port 1), then it will be connected to

$$\text{end module} = \frac{st + 1}{2} + \left(\frac{\gamma}{2} \cdot (t - 2)\right)\ ,\ \text{port} = 1$$

  - if the start module st is odd and the output port is even (i.e. module 1, port 2), then it will be connected to

$$\text{end module} = \frac{st + 1 + \gamma}{2} + \frac{\gamma}{2} + \left(\frac{\gamma}{2} \cdot (t - 2)\right)\ ,\ \text{port} = 1$$

  - if the start module st is even and the output port is odd (i.e. module 2, port 1), then it will be connected to

$$\text{end module} = \frac{st}{2} + \left(\frac{\gamma}{2} \cdot (t - 2)\right)\ ,\ \text{port} = 2$$

  - if the start module st and the output port are even (i.e. module 2, port 2), then it will be connected to

$$\text{end module} = \frac{st + \gamma}{2} + \frac{\gamma}{2} + \left(\frac{\gamma}{2} \cdot (t - 2)\right)\ ,\ \text{port} = 2$$

Unfortunately, the rule $\psi > 2$ when applied to the intermediate stages

$$I_1 = \lfloor \mathcal{S} \div 2 \rfloor \qquad I_2 = \mathcal{S} - (I_1 - 1)$$

does not work; this implies that:

- on the left side of the network the applicability of the algorithm is from the starting stage 1 up to the starting stage $I_1 - 1$ (in the example P=16: from the starting stage 1 up to the starting stage 2);
- on the right side of the network the applicability of the algorithm is from the starting stage $\mathcal{S}$ up to the starting stage $I_2 - 1$ (in the example P=16: from the starting stage 7 up to the starting stage 6);
- for the intermediate starting stages I_1 and I_2 (in the example P=16: the stages 3 and 5) another rule should be used:
  - if the start module st and the output port are odd (i.e. module 1, port 1), then it will be connected to

$$\text{end module} = st \ , \ \text{port} = 1$$

  - if the start module st is odd and the output port is even (i.e. module 1, port 2), then it will be connected to

$$\text{end module} = st + 1 \ , \ \text{port} = 1$$

  - if the start module st is even and the output port is odd (i.e. module 2, port 1), then it will be connected to

$$\text{end module} = st - 1 \ , \ \text{port} = 2$$

  - if the start module st and the output port are even (i.e. module 2, port 2), then it will be connected to

$$\text{end module} = st \ , \ \text{port} = 2$$

To summarize, the algorithm to **d**raw **B**enes **n**etwork **c**onnections (dBnc) is reported in 1: for the rules, please refer to the descriptions mentioned above.

Algorithm 1: **d**raw **B**enes **n**etwork **c**onnections (dBnc)

```
1  compute S = 2 log_2 P − 1;
2  compute I_1 = ⌊S ÷ 2⌋;
3  compute I_2 = S − (I_1 − 1);
4  from left to right;
5  for stg ← 1 to (I_1 − 1) do
6  |  compute γ = P ÷ 2^stg;
7  |  compute ψ = 2^(stg−1);
8  |  for rep ← 1 to ψ do
9  |  |  compute starting point x = 1 + ((rep − 1) · γ);
10 |  |  compute ending point y = (x + γ) − 1;
11 |  |  foreach start module s in set (x, y) do
12 |  |  |  if rep == 1 then
13 |  |  |  |  if s is odd then
14 |  |  |  |  |  use rules ψ = 1 for starting module odd;
15 |  |  |  |  else
16 |  |  |  |  |  use rules ψ = 1 for starting module even;
17 |  |  |  |  end
18 |  |  |  if rep == 2 then
19 |  |  |  |  if s is odd then
20 |  |  |  |  |  use rules ψ = 2 for starting module odd;
21 |  |  |  |  else
22 |  |  |  |  |  use rules ψ = 2 for starting module even;
23 |  |  |  |  end
24 |  |  |  if rep > 2 then
25 |  |  |  |  if s is odd then
26 |  |  |  |  |  use rules ψ > 2 for starting module odd;
27 |  |  |  |  else
28 |  |  |  |  |  use rules ψ > 2 for starting module even;
29 |  |  |  |  end
30 |  |  |
31 |  |  end
32 |  end
33 end
34 from right to left;
35 for stg ← S to (I_2 − 1) do
36 |  repeat in dual mode 6 − 32;
37 end
38 complete with intermediate stages;
39 foreach stg in set (I_1, I_2) do
40 |  use rules for intermediate stages;
41 end
```

Index