S*a*-Ti*k*Z[*]

Claudio Fiandrino
`claudio.fiandrino@gmail.com`

January 29, 2014

Contents

Introduction

The S*a*-Ti*k*Z library is a tool for drawing *switching-architectures*. In particular, one of its aims, is to help students to verify the correctness of their exercises, but

---

[*] This package has version number *v*0.7a of January 29, 2014; it is released under and subject to the LaTeX Project Public License (LPPL).

it could also help teachers in preparing lecture notes. The official webpage of the project is hosted on http://cfiandra.github.io/Sa-TikZ/ and a public repository of the library can be found in https://github.com/cfiandra/Sa-TikZ.

The S*a*-Ti*k*Z library can be loaded in a LaTeX preamble 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 auxiliary Ti*k*Z libraries `calc`, `decorations.pathreplacing` and `positioning` are automatically loaded; in the latter case also the Ti*k*Z package is loaded.

The version *v*0.7a is compliant with Ti*k*Z 3.0.0[1] and provides a way to draw Clos Networks Strictly-non-Blocking (snb) and Rearrangeable (rear), Benes Networks and Banyan Networks (in particular Omega[2] and Flip Networks); moreover, the package provides the possibility to fully customize the aspect of the drawn network: the dimensions of module, their distance and the font used are some examples. Finally, S*a*-Ti*k*Z let users 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.

---

[1] In order to compile the documentation, and only for this scope, Ti*k*Z 3.0.0 is *mandatory*.

[2] Implementation of Omega Networks by João Gabriel Reis.

`/tikz/banyan omega` (no value)

Option for drawing an Banyan-Omega Network.

`/tikz/banyan flip` (no value)

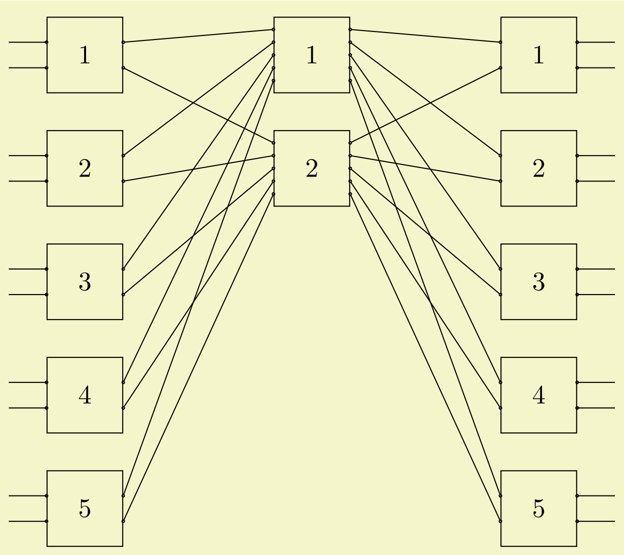
Option for drawing an Banyan-Omega Network with inverse shuffle exchange (Flip).

inside a `tikzpicture` environment:

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

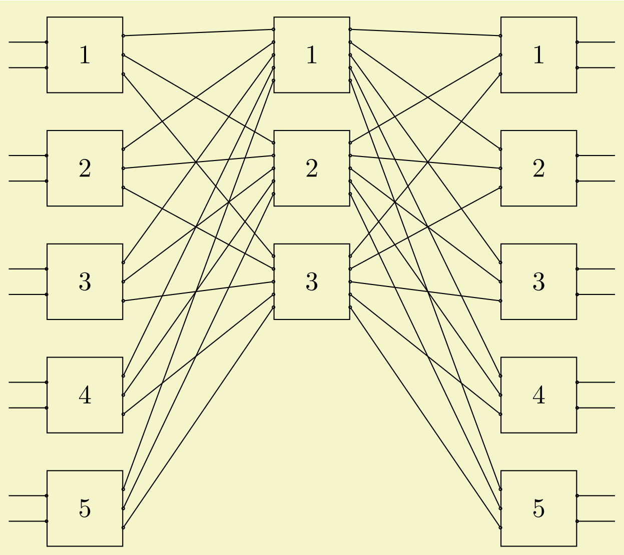
1.1 Examples of Clos Networks

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}
```

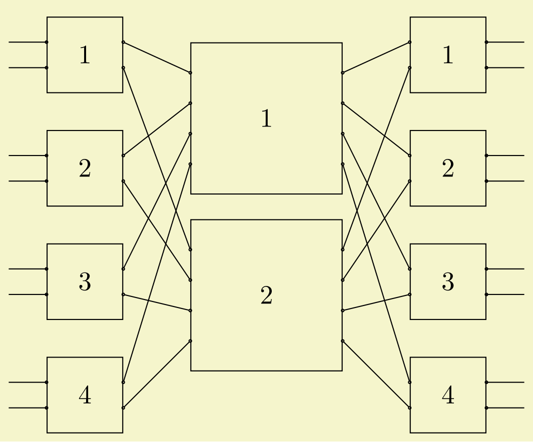
Notice from the examples that the library is able to compute *automatically* the constraints defining a Clos Network to be Strictly-non-Blocking or Rearrangeable. The network drawn is characterized by:

- 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.

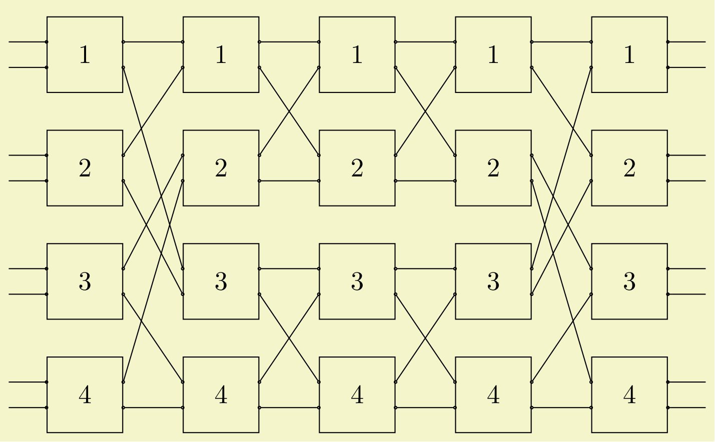
1.2 Examples of Benes Networks

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:

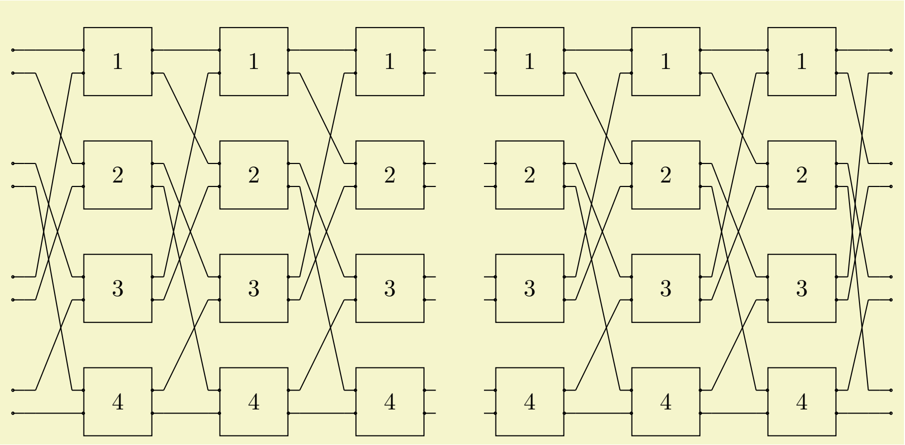


```
\begin{tikzpicture}
    \node[benes complete] {};
\end{tikzpicture}
```

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

1.3 Examples of Banyan Networks

The following examples show the two Banyan Network architectures that S*a*-Ti*k*Z can draw.



```
\begin{tikzpicture}
    % Omega Network on the left
    \node[banyan omega] {};
    \begin{scope}[xshift=7.25cm]
       % Flip network on the right
       \node[banyan flip]{};
    \end{scope}
\end{tikzpicture}
```

2 The options

2.1 Designing choices

This subsection provides a detailed analysis of the options and parameters that can be used to draw Clos, Benes and Omega Networks. In particular:

- Clos Networks are analysed in 2.1.1;
- Benes Networks are analysed in 2.1.2;
- Banyan Networks are analysed in 2.1.3.

Each subsection presents the keys' definitions and simple examples to help the reader.

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 customized 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 important 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 modules in the first stage.

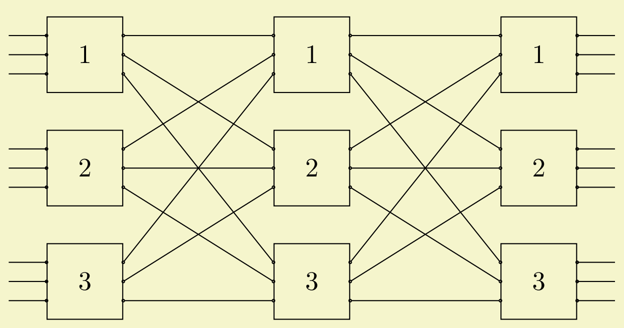
`/tikz/r3={⟨value⟩}` (no default, initially `5`)

This is the number of modules in the last stage.

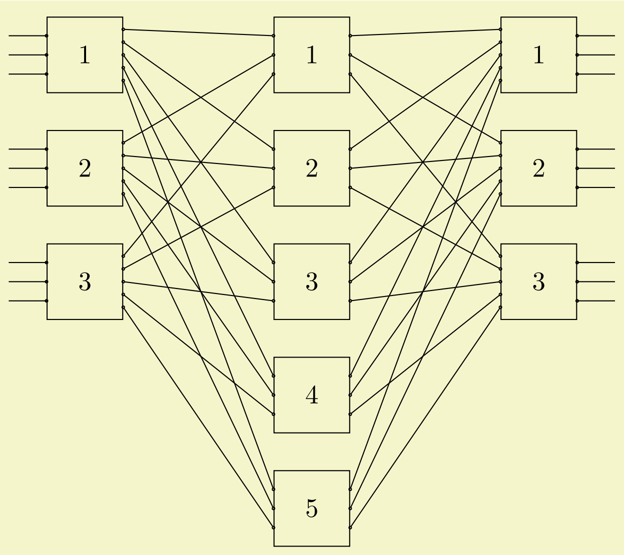
The two design parameters provide the number of ports of each module:

$$m_1 = \frac{N}{r_1} \qquad\qquad m_3 = \frac{M}{r_3}$$

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

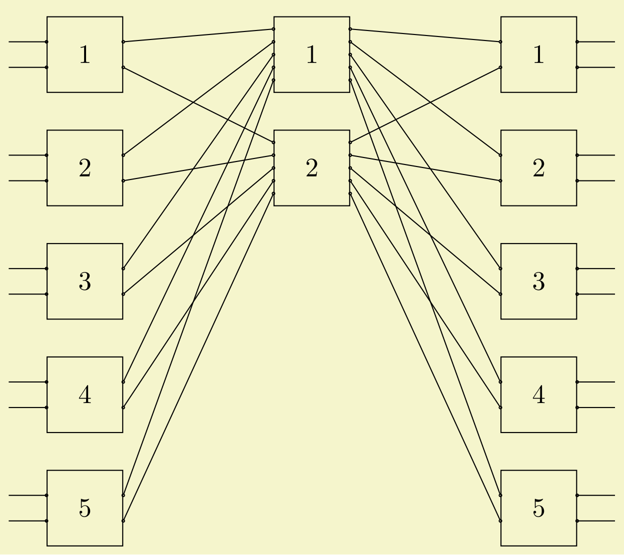


```
\begin{tikzpicture}
    \node[N=9,r1=3,M=9,r3=3,clos rear] {};
\end{tikzpicture}
```



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

Notice a very important thing: when the keys are set within the `\node`, the type of the architecture should be loaded *after* all the design parameters; if this constraint is not respected, you will end up with an architecture which has 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. Hence, as design choice, it just possible to select 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.

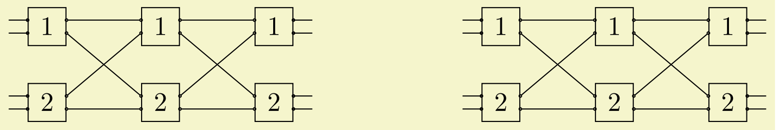
Notice that `P` could assume values

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

and the user is responsible to correctly set this parameter.

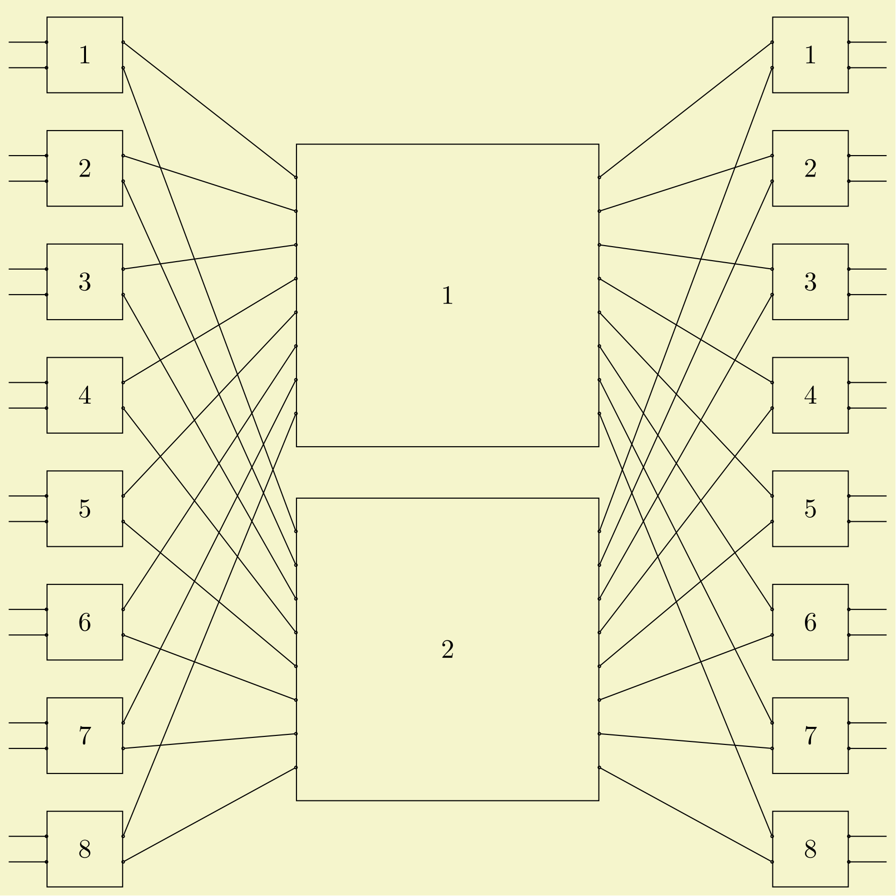
For low values of p there are no problems in visualizing the network, but as p increases, the user should take care of the modules' dimension and the modules' separation (vertical and horizontal): these parameters could be customized as explained in the subsection 2.2. Actually, for `benes complete`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

Notice that, 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:



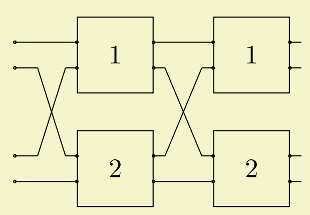
```
\begin{tikzpicture}
    \node[P=16,benes] {};
\end{tikzpicture}
```

It holds the same concept afore-mentioned for Clos Networks: set the parameter `P` before declaring the `\node` to be a Benes Network.

2.1.3 Banyan Networks

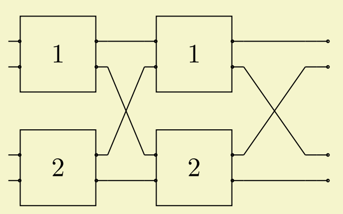
Banyan Networks are architectures based on Benes Networks: they have particular interconnections properties. As well as Benes Networks, only the number of inputs and outputs ports can be selected and it is `P`.

S*a*-Ti*k*Zis able to represent Omega and Flip Banyan Networks. Omega networks are based on João Gabriel Reis' implementation. An example of 4 × 4 `banyan omega` network:



```
\begin{tikzpicture}
    \node[P=4,banyan omega] {};
\end{tikzpicture}
```

An example of 4 × 4 `banyan flip` network:



```
\begin{tikzpicture}
    \node[P=4,banyan flip] {};
\end{tikzpicture}
```

2.2 Output customization

This subsection presents the keys devoted to customize the networks' aspect.

`/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.

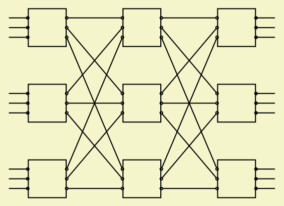
`/tikz/module font={⟨font commands⟩}` (default `\normalfont`)

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 it is set to true. Beware: this option is valid only for `clos snb`, `clos rear`, `benes` and `benes complete` networks, but it does not holds for the architectures explained in section 4.

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



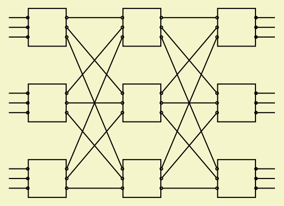
```
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
    \node[module size=0.5cm,pin length factor=0.5,
        module ysep=1, module xsep=1.25,
        clos rear={module label opacity=0}] {};
\end{tikzpicture}
```

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

`\tikzset{`⟨*options*⟩`}`

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

Therefore, the previous example can be modified into:

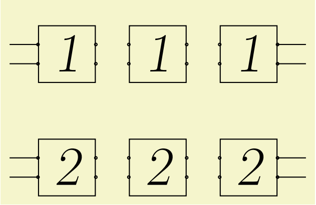


```
\tikzset{module size=0.5cm,pin length factor=0.5,
         module ysep=1, module xsep=1.25}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
    \node[clos rear={module label opacity=0}] {};
\end{tikzpicture}
```

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:

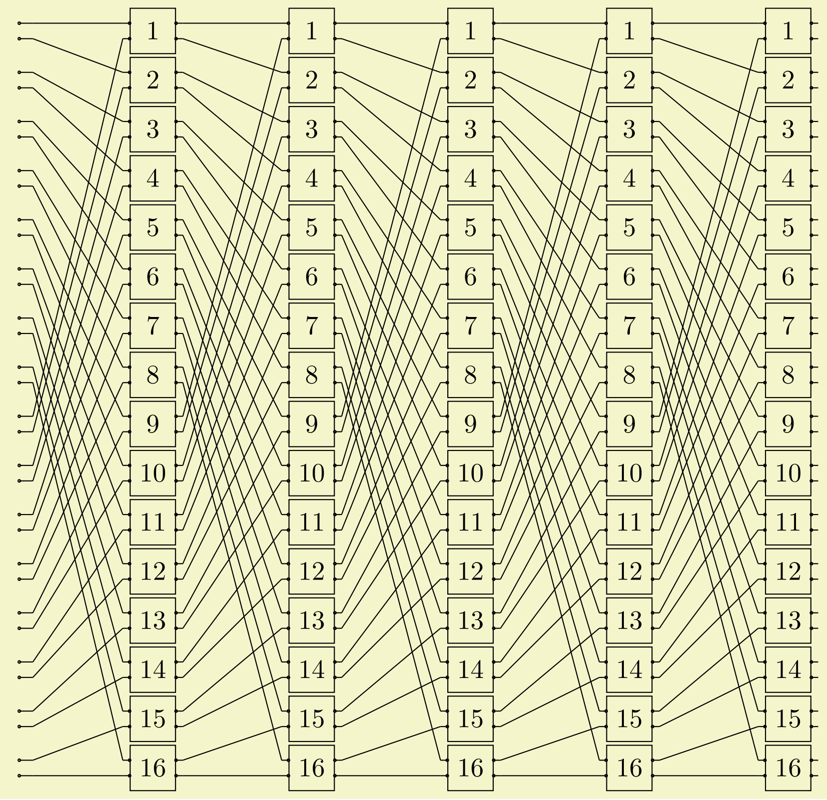
```
\tikzset{module size definition/.style={
  module size=0.75cm,
  pin length factor=0.75,
  module xsep=2,
  module ysep=2,
  }
}
\tikzset{module size definition,
 P=16,
}
\begin{tikzpicture}
    \node[benes] {};
\end{tikzpicture}
```

Here is a Benes Network 4 × 4 which presents an extremely large font size for the module labels and its interconnection paths are disabled:



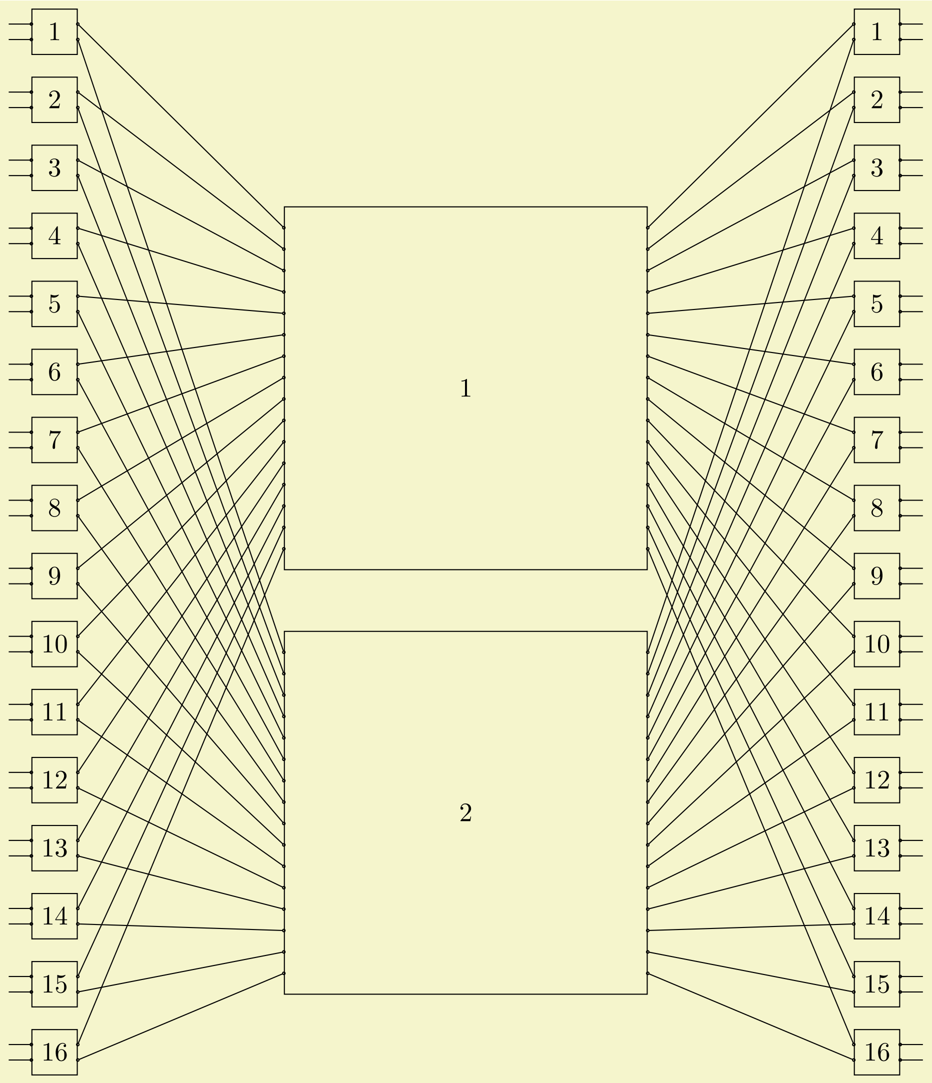
```
\tikzset{my style/.style={
  module size=0.75cm,
  pin length factor=0.75,
  module xsep=2,
  }
}
\tikzset{my style, P=4,
 module font=\huge\slshape,
 connections disabled=true
}
\begin{tikzpicture}
    \node[benes complete] {};
\end{tikzpicture}
```

The following example is a 16 × 16 Omega Network.



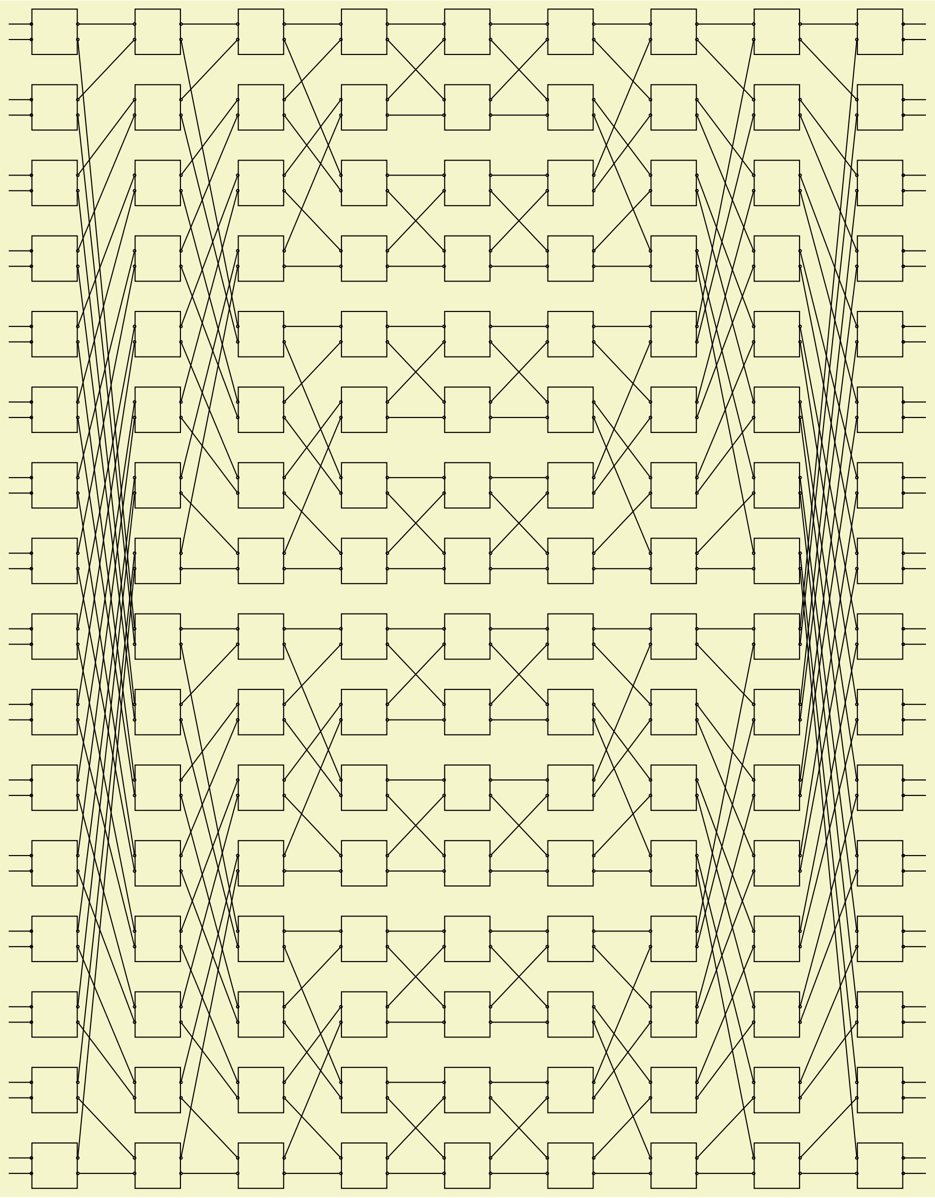
```
\tikzset{module size=0.6cm,pin length factor=0.6,
         module ysep=0.65, module xsep=3.5}
\begin{tikzpicture}[P=32]
    \node[banyan omega] {};
\end{tikzpicture}
```

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

This section presents some more advanced examples. More in detail, it is described how to add elements to the basic architecture; the elements can be:

- labels for the input and output ports;
- paths interconnecting input and output ports.

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:

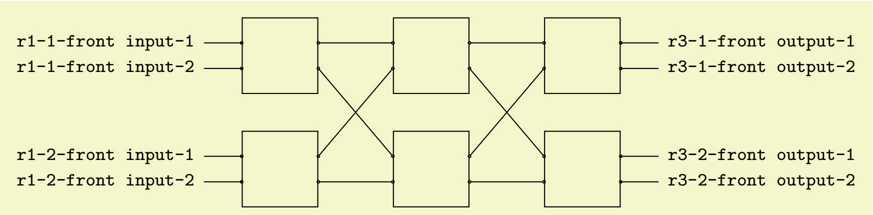
`r1-module number-front input-port number`; example:
`r1-1-front input-1`;

Each front output port could be accessed by means of:

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

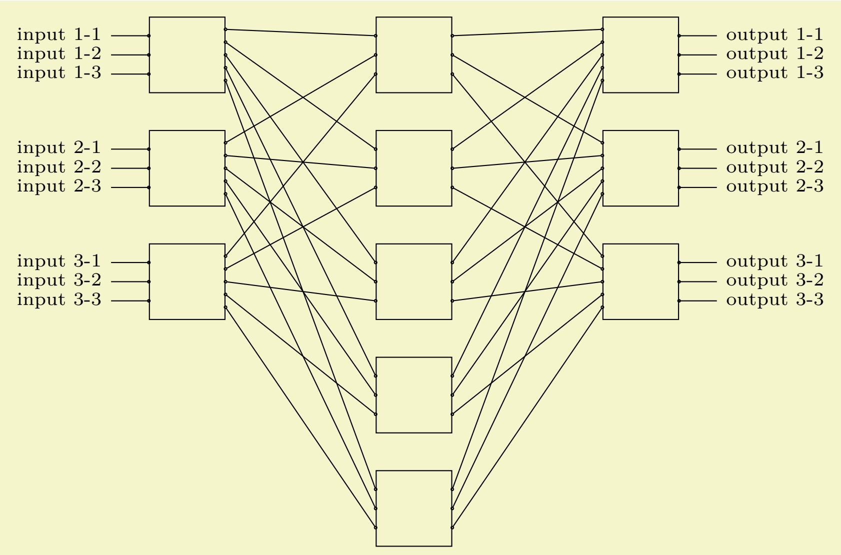
Noticed that the first stage is always 1, but the last stage may be different from 3 in case the `benes complete` network is drawn. Errors will occur in case the last stage number is not correct and the user is responsible for the correct setting.

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}
```

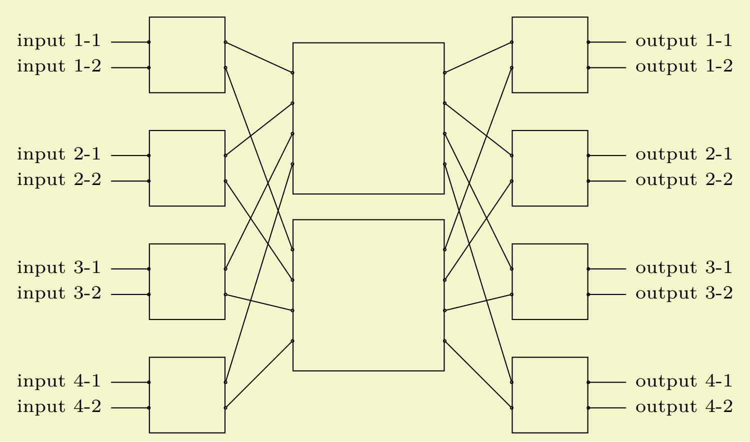
The following is a Strictly-non-Blocking Clos network of 9 input and output ports in which the first and last stage have 3 modules each one.



```
\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}
```

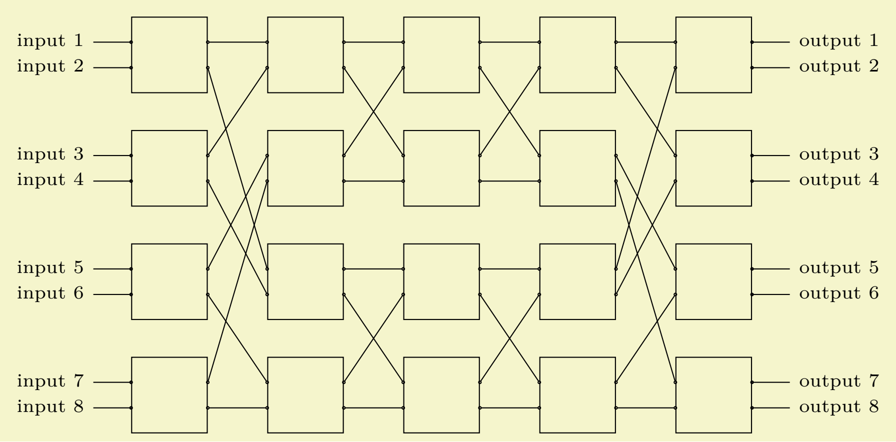
The same applies also for Benes Networks:



```
\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}
```

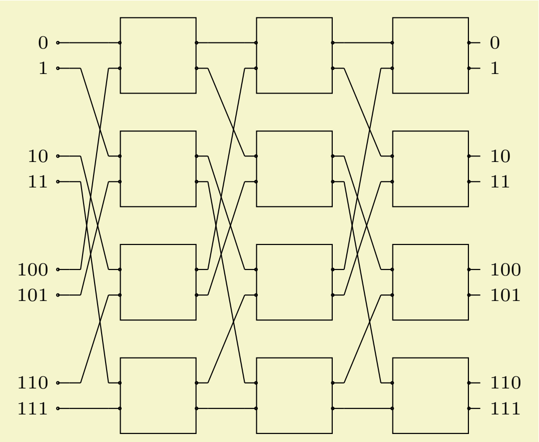
and to the correspondent complete form:



```
\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}
```

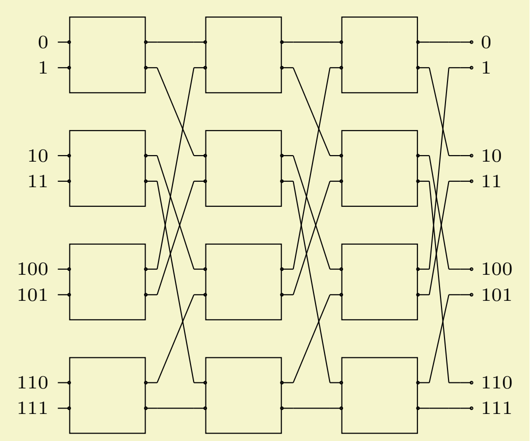
Notice that in this case to access the `front output` ports, the stage number correct is 5 and not 3 as usual.

TikZ has very useful `bin`(x) function: it converts x (it is assumed to be a 10 base integer) into its binary representation. Exploiting this function for Omega or Flip Networks is very convenient. An example of Omega Network:



```
\begin{tikzpicture}
\node[banyan omega={module label opacity=0}] {};
\newcounter{porta}
\setcounter{porta}{0}
\foreach \module in {1,...,4}{
\foreach \port in {1,...,2}{
\stepcounter{porta}
\pgfmathbin{\theporta-1}% get the binary value of the counter
\node[left] at (r0-\module-front input-\port)
 {\scriptsize{\pgfmathresult}};
\node[right] at (r3-\module-front output-\port)
 {\scriptsize{\pgfmathresult}};
}
}
\end{tikzpicture}
```

An example of Flip Network:



```
\begin{tikzpicture}
\node[banyan flip={module label opacity=0}] {};
\newcounter{portb}
\setcounter{portb}{0}
\foreach \module in {1,...,4}{
\foreach \port in {1,...,2}{
\stepcounter{portb}
\pgfmathbin{\theportb-1}
\node[left] at (r0-\module-front input-\port)
 {\scriptsize{\pgfmathresult}};
\node[right] at (r3-\module-front output-\port)
 {\scriptsize{\pgfmathresult}};
}
}
\end{tikzpicture}
```

Notice that for Banyan Networks the first module is characterized by number 0 and not 1.

3.2 Identifying input/output ports per module

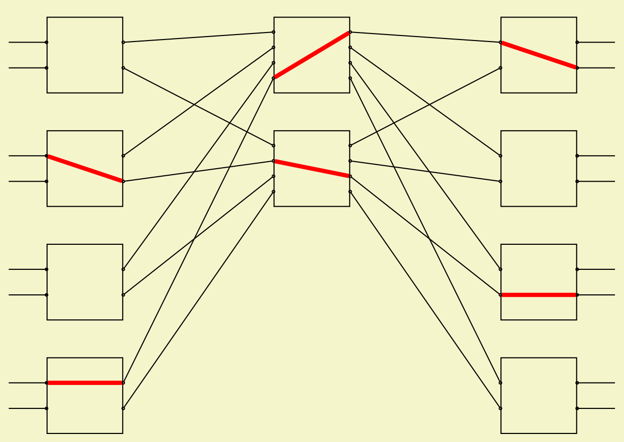
It is also possible to access, for each module of each stage, its input and output ports. The syntax is similar to the one used for the front input and output ports; each input port could be accessed by means of:

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

Each output port could be accessed by means of:

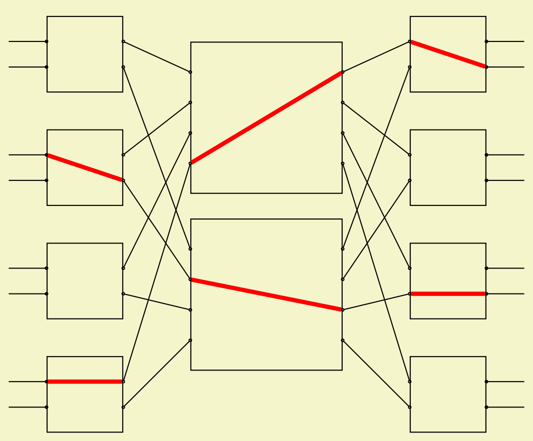
`rstage number-module number-front output-port number`; example: `r2-1-output-1`;

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



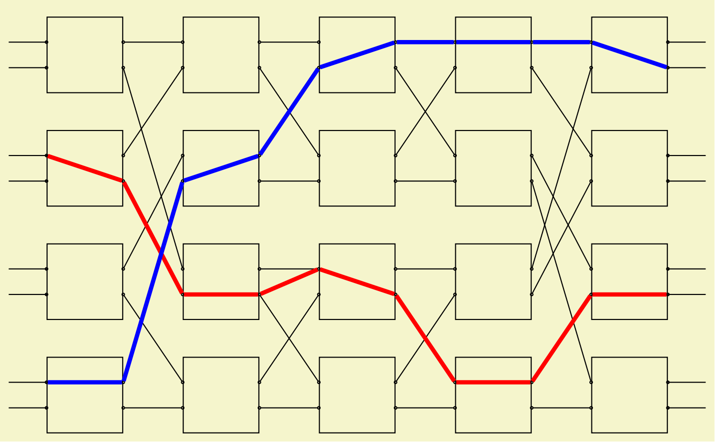
```
\begin{tikzpicture}
    \node[N=8,r1=4,M=8,r3=4,clos rear={module label opacity=0}] {};
    \draw[red,ultra thick](r1-2-input-1)-(r1-2-output-2)
    (r2-2-input-2)-(r2-2-output-3)
    (r3-3-input-2)-(r3-3-output-2);
   \draw[red,ultra thick](r1-4-input-1)-(r1-4-output-1)
    (r2-1-input-4)-(r2-1-output-1)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

Similarly, an example in a Benes Network:



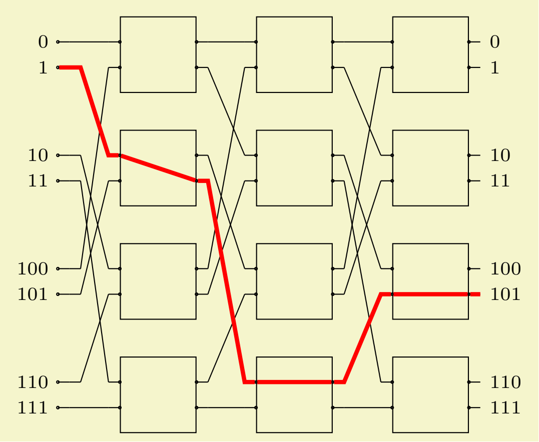
```
\begin{tikzpicture}
    \node[benes={module label opacity=0}] {};
    \draw[red,ultra thick](r1-2-input-1)-(r1-2-output-2)
    (r2-2-input-2)-(r2-2-output-3)
    (r3-3-input-2)-(r3-3-output-2);
   \draw[red,ultra thick](r1-4-input-1)-(r1-4-output-1)
    (r2-1-input-4)-(r2-1-output-1)
    (r3-1-input-1)-(r3-1-output-2);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

For Banyan Networks, due to the way in which the interconnections are established, it is advised to proceed as follows:



```
\begin{tikzpicture}
\node[banyan omega={module label opacity=0}] {};
\newcounter{portc}
\setcounter{portc}{0}
\foreach \module in {1,...,4}{
\foreach \port in {1,...,2}{
\stepcounter{portc}
\pgfmathbin{\theportc-1}
\node[left] at (r0-\module-front input-\port)
 {\scriptsize{\pgfmathresult}};
\node[right] at (r3-\module-front output-\port)
 {\scriptsize{\pgfmathresult}};
}
}
\draw[red,ultra thick]
(r0-1-front input-2)-(r0-1-front output-2)-
(r1-2-front input-1)-(r1-2-input-1)-
(r1-2-output-2)-(r1-2-front output-2)-
(r2-4-front input-1)-(r2-4-input-1)-
(r2-4-output-1)-(r2-4-front output-1)-
(r3-3-front input-2)- (r3-3-input-2)-
(r3-3-output-2)-(r3-3-front output-2);
\end{tikzpicture}
```

4 Architectures for didactic purposes

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

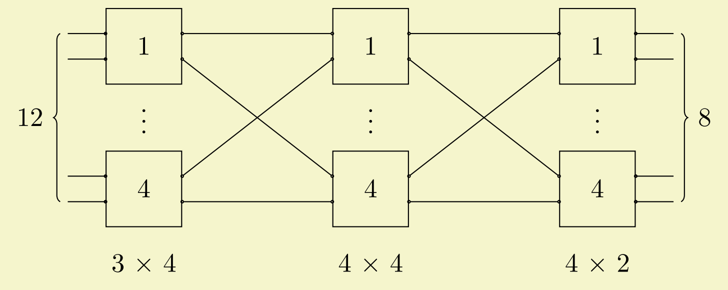
`/tikz/clos snb example` (no value)

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

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

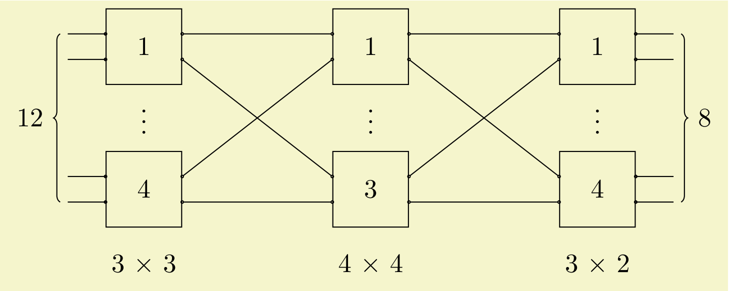
Option for quickly drawing a Clos Network Rearrangeable.

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=12,r1=4,M=8,r3=4]
    \node[clos snb example] {};
\end{tikzpicture}
```

Similarly, an example of a Rearrangeable Clos Network:



```
\begin{tikzpicture}[N=12,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.

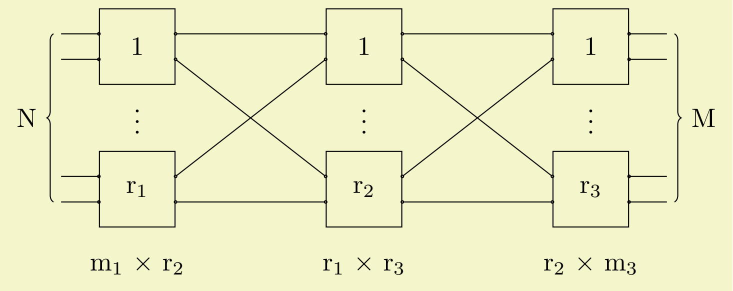
`/tikz/r3 label={⟨value⟩}` (default $\mathtt{r}_3$)

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

`/tikz/m3 label={⟨value⟩}` (default $\texttt{m}_3$)

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

An example with the default values for the labels:



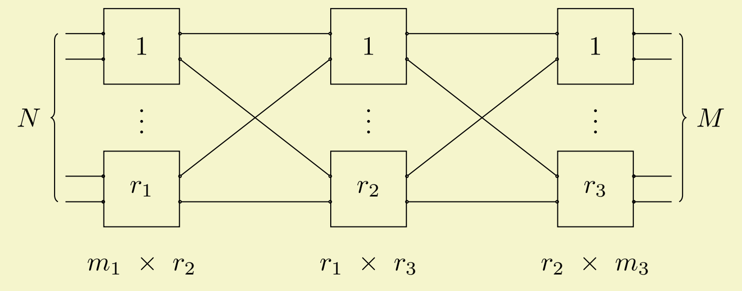
```
%\tikzset{N=8,r1=4,M=8,r3=4} % here, setting the parameters is useless
\begin{tikzpicture}
    \node[clos example with labels] {};
\end{tikzpicture}
```

To have automatically all the labels in math mode, you can use:

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

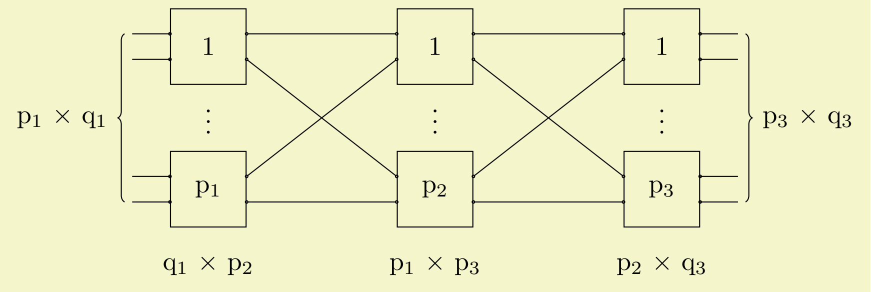
This option is normally disabled `set math mode labels/.default=false`; to ensure labels be set completely in math mode is sufficient set `set math mode labels=true` before the type of the network.

An example:



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

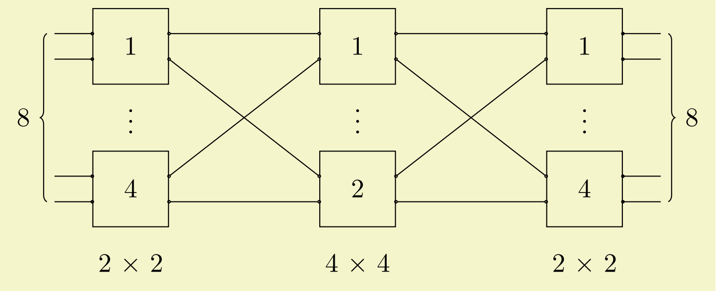
This example, instead, represents a `clos example with labels` network with custom labels introduced by means of the `\tikzset` syntax.



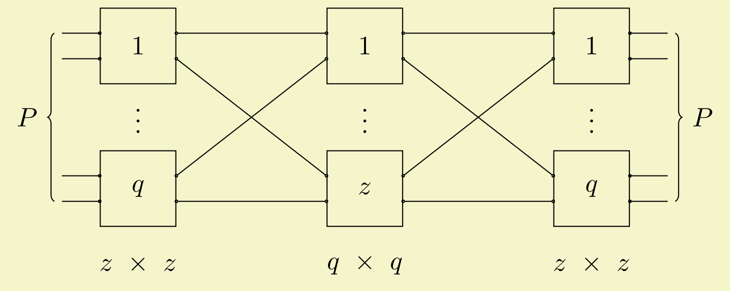
```
\tikzset{N label={p$_1$ $\times$ q$_1$},M label={p$_3$ $\times$ q$_3$},
r1 label=p$_1$, m1 label=q$_1$, r2 label=p$_2$,r3 label=p$_3$, m3 label=q$_3$}
\begin{tikzpicture}
    \node[clos example with labels] {};
\end{tikzpicture}
```

Notice that it does not exist an equivalent of `clos example with labels` or `clos rear example` for Benes Networks: this because Benes Networks are a particular type of Rearrangeable Clos Networks where `P`=`N`=`M` and $m1$=$m3$=z=2, thus `r1`=`r3`=$q = P/z$.

For example:



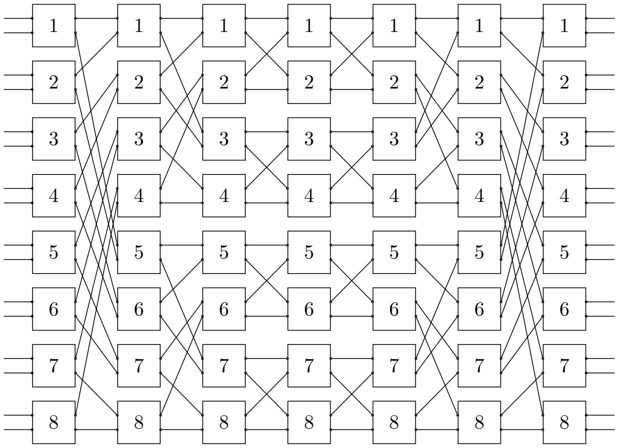
```
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
    \node[clos rear example] {};
\end{tikzpicture}
```



```
\tikzset{N label={P},M label={P},
r1 label=q, m1 label=z, r2 label=z,r3 label=q, m3 label=z}
\begin{tikzpicture}
    \node[set math mode labels=true,clos example with labels] {};
\end{tikzpicture}
```

A Algorithm for drawing connections in Benes Networks

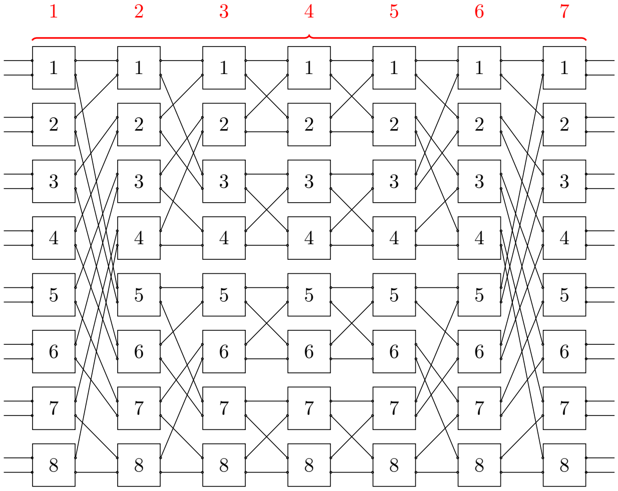
To explain how the connections of the `benes complete` networks are drawn, the following reference example will be considered:



The network is 16 × 16 (`P`=16), thus the number of stages $\mathcal{S}$ is:

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

Indeed:

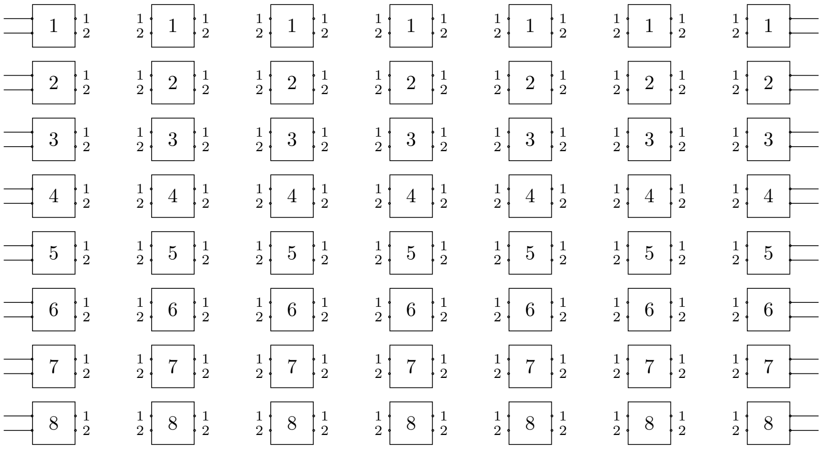


This parameter, therefore, allows to correctly draw all the modules of the network. As it will be pointed out later better, knowing `P` is also important to define the range of applicability of the algorithm among the stages. Notice the network symmetry: the connections from stage 1 to stage 4 are exactly identical to the connections from stage 7 to stage 4.

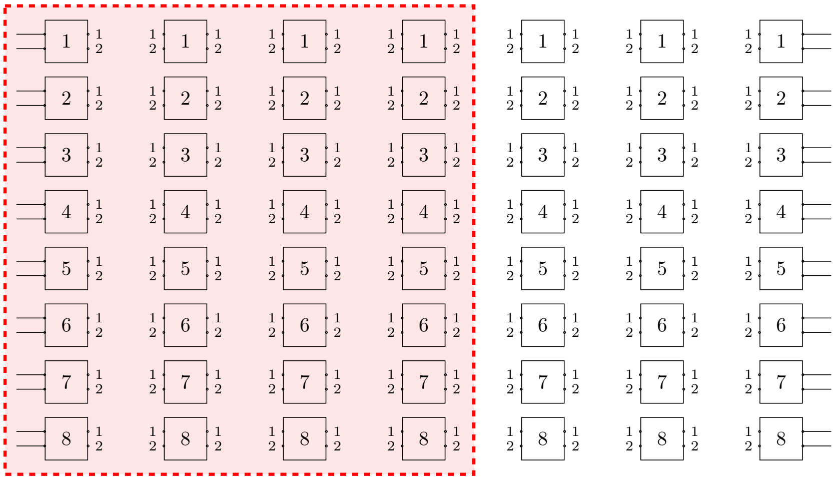
The first step is *labelling* modules and ports. S*a*-Ti*k*Z uses this rationale:

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

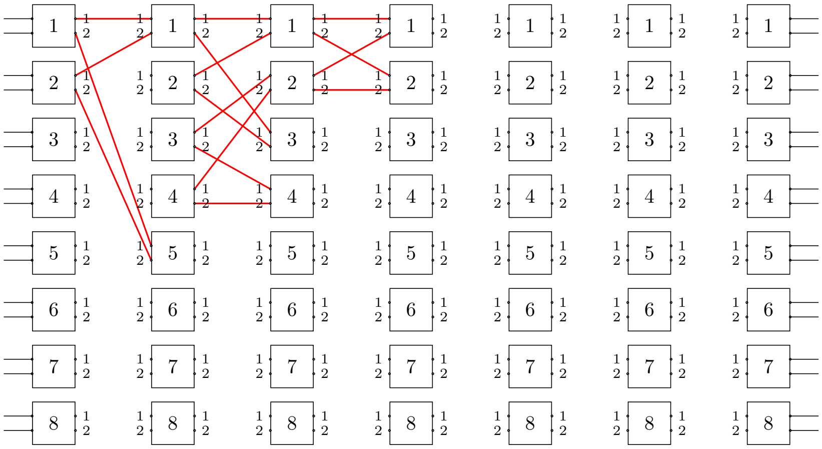
Thus:



Due to the network symmetry, the attention will be focused only on the left side of the network initially. On the right part of the network, 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

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

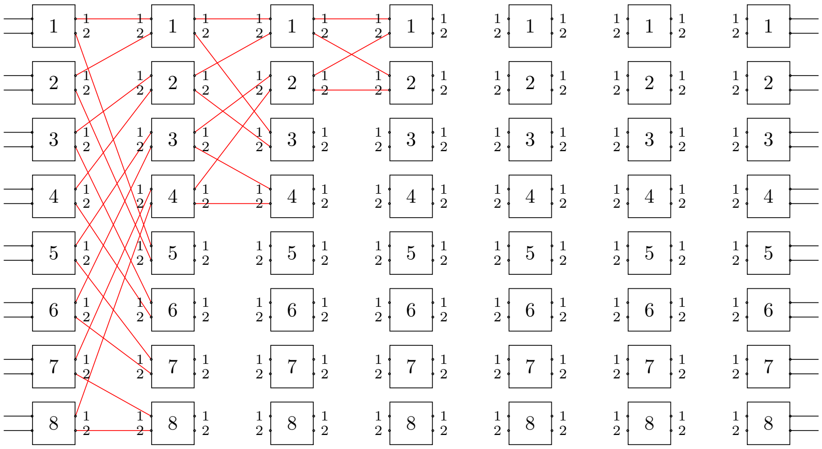
What is the term γ? It is a corrective term that depends on the starting stage. Consider, indeed, the connections of the output port 2 of module 1 in the first and in the second starting stages:

`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 $\gamma = 8$ and $\gamma = 4$ in the first case and in the second case respectively. This suggest that γ is related in some sense to the stage of the start module: in our example `P`=16, hence the relation is

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

However, following this strategy 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.

Since in the first stage $\gamma = 8$ $(P/2^1)$, in the second stage $\gamma = 4$ $(P/2^2)$ and in the third stage $\gamma = 2$ $(P/2^3)$: this means that γ defines the *module applicability range*.

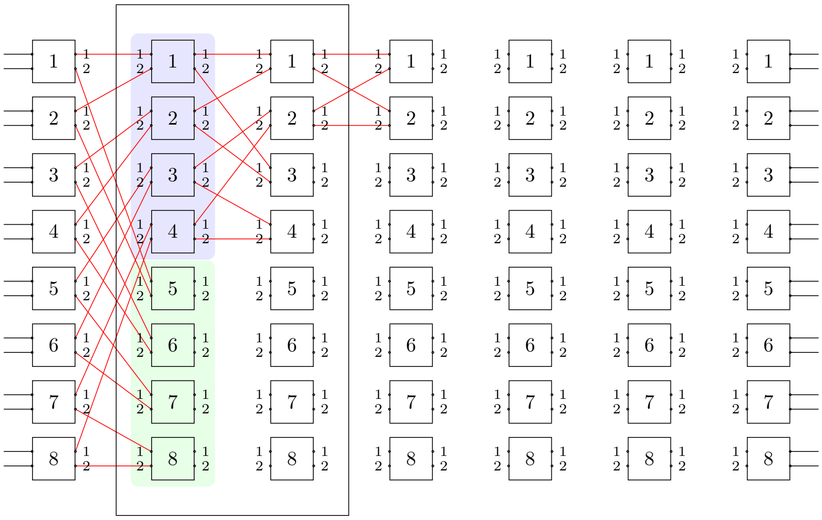
Notice now, that for the second stage and the third stage, the algorithm should be simply repeated:

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

The repetition ψ depends on the stage according to the following 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 $\psi = 2 \implies \{1, 2\}$, hence 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, the knowledge of the starting and ending modules of each stage is not sufficient to reach the goal: the reason is that the algorithm works and draws the connections in a correct manner when the module labels start with 1. However, during the repetitions the new starting module labels are different, thus 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 it is 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 afore-mentioned description.

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

```
 1 compute $\mathcal{S} = 2\log_2 P - 1$;
 2 compute $I_1 = \lfloor \mathcal{S} \div 2 \rfloor$;
 3 compute $I_2 = \mathcal{S} - (I_1 - 1)$;
 4 from left to right;
 5 for $stg \leftarrow 1$ to $(I_1 - 1)$ do
 6     compute $\gamma = P \div 2^{stg}$;
 7     compute $\psi = 2^{stg-1}$;
 8     for $t \leftarrow 1$ to $\psi$ do
 9         compute starting point $x = 1 + ((t-1) \cdot \gamma)$;
10         compute ending point $y = (x + \gamma) - 1$;
11         foreach start module $s$ in set $(x, y)$ do
12             if $t == 1$ then
13                 if $s$ is odd then
14                     use rules $\psi = 1$ for starting module odd;
15                 else
16                     use rules $\psi = 1$ for starting module even;
17                 end
18             if $t == 2$ then
19                 if $s$ is odd then
20                     use rules $\psi = 2$ for starting module odd;
21                 else
22                     use rules $\psi = 2$ for starting module even;
23                 end
24             if $t > 2$ then
25                 if $s$ is odd then
26                     use rules $\psi > 2$ for starting module odd;
27                 else
28                     use rules $\psi > 2$ for starting module even;
29                 end
30         end
31     end
32 end
33 from right to left;
34 for $stg \leftarrow \mathcal{S}$ to $(I_2 - 1)$ do
35     repeat in dual mode $6 - 32$;
36 end
37 complete with intermediate stages;
38 foreach $stg$ in set $(I_1, I_2)$ do
39     use rules for intermediate stages;
40 end
```

Index