Stratus User's Manual

Sophie Belloeil

1 What's new

1.1 march 2012

- A configuration file can be used to direct Stratus.
- A Stratus's description can now be exported either in Alliance VST format or standard VHDL using the `Save` method and the configuration file.
- A `Stimuli` method permits to generate stimuli for simulation.
- Thanks to the `Stimuli` method, a testbench can now automatically be generated either in Alliance PAT format or standard VHDL.
- The `Simul` method of the class `Model` permits to simulate either with the Alliance's simulator `asimut` or a standard VHDL simulator (`ghdl`).

2 Introduction

2.1 Stratus

2.1.1 Name

Stratus – Procedural design language based upon *Python*

2.1.2 Description

Stratus is a set of *Python* methods/functions dedicated to procedural generation purposes. From a user point of view, *Stratus* is a circuit's description language that allows *Python* programming flow control, variable use, and specialized functions in order to handle vlsi objects.

Based upon the *Hurricane* data structures, the *Stratus* language gives the user the ability to describe netlist and layout views.

2.1.3 Configuration

A configuration file can be used to direct the generation process of Stratus. With this file, the user can choose the output format (vst, vhdl...), the simulator (asimut, ghdl...), the standard cell library... This configuration file named `.st_config.py` must be placed either in the HOME directory or in the current directory. This file contains a set of variables used in the process generation of Stratus, as for example :

```
format = 'vhdl'
simulator = 'ghdl'
```

The default configuration of Stratus uses the Alliance CAD system, ie `'vst'` as `format` and `'asimut'` as `simulator`.

2.1.4 Description of a cell

A cell is a hierachical structural description of a circuit in terms of ports (I/Os), signals (nets) and instances.

The description of a cell is done by creating a new class, derivating for class `Model`, with different methods :

- Method `Interface` : Description of the external ports of the cell :
  - SignalIn, SignalOut, ...
- Method `Netlist` : Description of the netlist of the cell :
  - Inst, Signal
- Method `Layout` : Description of the layout of the cell :
  - Place, PlaceTop, PlaceBottom, PlaceRight, PlaceLeft ...
- Method `Stimuli` : Description of the simulation stimuli of the cell :
  - affect, addd ...

2.1.5 Creation of the cell

After the description of a cell as a sub-class of `Model`, the cell has to be instantiated. The different methods described before have to be called.

Then different methods are provided :

- Method `View` : Opens/Refreshes the editor in order to see the created layout
- Method `Save` : Saves the created cell in the desired format thanks to the configuration file

- no argument : creation of a netlist file
- PHYSICAL : creation of a netlist file AND a layout file
- STRATUS : creation of a python/stratus file
  * FileName : optionnal argument when using Save(STRATUS) in order to choose the name of the file to be generated
  * Be careful : if one wants to create a stratus file AND a netlist, always use Save(STRATUS) before Save() !

- Method `Testbench` : Creates the testbench of the cell using the `Stimuli` method to compute the stimuli. The output format depends of the `format` variable given in the configuration file

- Method `Simul` : Runs the simulation using the simulator named in the configuration file

2.1.6 Syntax

A *Stratus* file must have a .py extension and must begin as follow :

```
#!/usr/bin/env python

from stratus import *
```

The description of a cell as a sub-class of `Model` is done as follow :

```
class myClass ( Model ) :
    ...
```

The creation of the cell is done by instantiating the previous class as follow :

```
exemple = myClass ( name, param )
```

After the different methods can be called as follow :

```
exemple.Interface()
exemple.Netlist()
exemple.Save()
    ...
```

In order to execute a *Stratus* file (named `file` for example), one has two choices :

```
python file.py
```

Or :

```
chmod u+x file.py
./file.py
```

The names used in *Stratus*, as arguments to *Stratus* functions, should be alphanumerical, including the underscore. The arguments of *Stratus* are case sensitive, so VDD is not equivalent to VDD.

Vectorized connectors or signal can be used using the [N:M] construct.

2.1.7 Syntax highlighting

When using vi, it's possible to have the right syntax highlighting :

- Commands to do when you want to change once the coloration of your file :

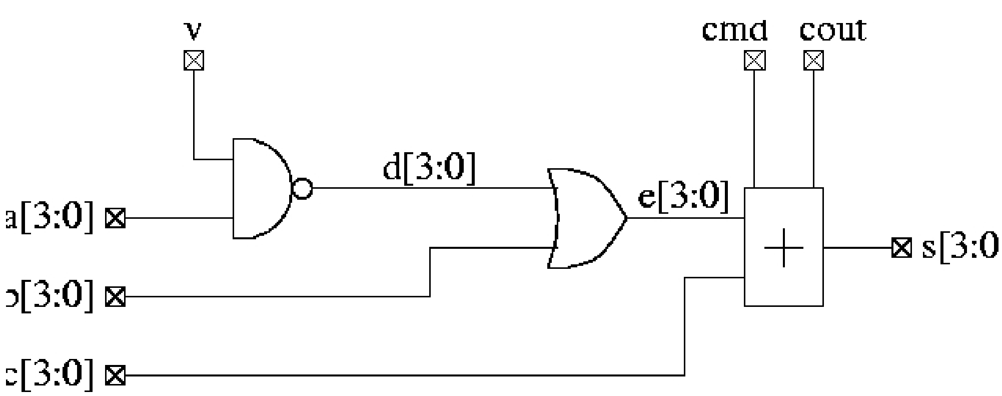
```
:syntax off
:source /asim/coriolis/share/etc/stratus.vim
```

- Modification of your .vimrc in order to have the syntax highlighting each time you open a file :

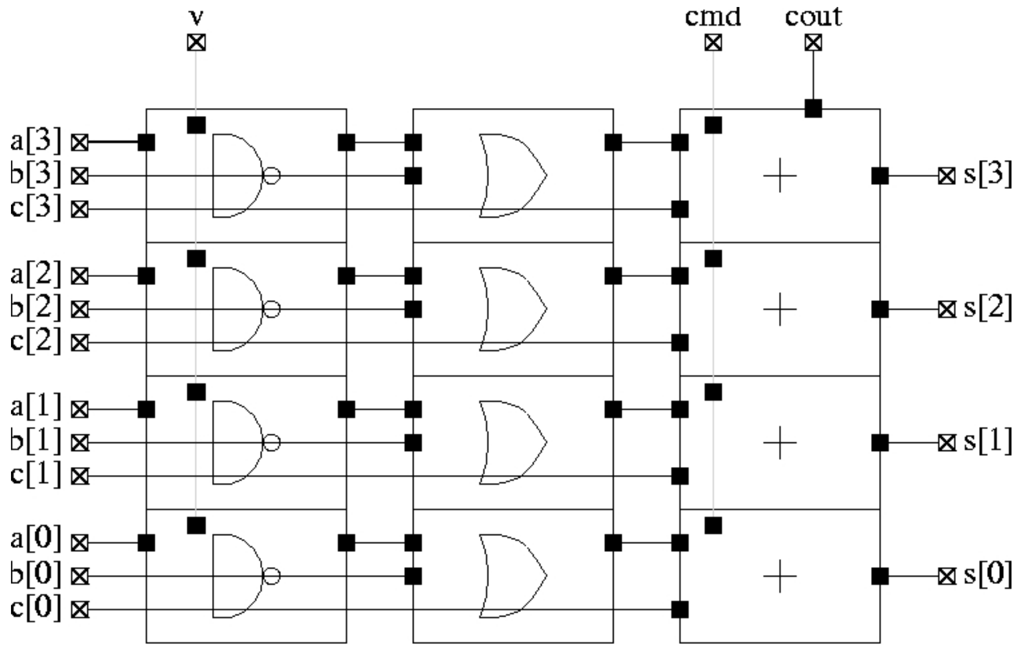
```
syntax off
autocmd BufRead,BufNewfile *.py so /asim/coriolis/share/etc/stratus.vim
syntax on
```

2.2 Example

2.2.1 The addaccu circuit



2.2.2 The data-path



2.2.3 Description of the circuit with *Stratus* : file addaccu.py

```python
#!/usr/bin/env python

from stratus import *

class addaccu ( Model ) :

  def Interface ( self ) :
    self.nbit = self._param['nbit']

    self.a    = SignalIn ( "a", self.nbit )
    self.b    = SignalIn ( "b", self.nbit )
    self.c    = SignalIn ( "c", self.nbit )
    self.v    = SignalIn ( "v", 1 )
    self.cmd  = SignalIn ( "cmd", 1 )

    self.cout = SignalOut ( "cout", 1 )
    self.s    = SignalOut ( "s", self.nbit )

    self.vdd = VddIn ( "vdd" )
    self.vss = VssIn ( "vss" )

  def Netlist ( self ) :
    d_aux = Signal ( "d_aux", self.nbit )
    e_aux = Signal ( "e_aux", self.nbit )
    ovr   = Signal ( "ovr", 1  )

    Generate ( "DpgenNand2", "nand2%d" % self.nbit
             , param = { 'nbit'     : self.nbit
                       , 'physical' : True
                       }
             )
    self.instNand2 = Inst ( "nand2%d" % self.nbit, "instance_nand2"
                          , map   = { 'i0'   : Cat ( self.v, self.v, self.v, self.v )
                                    , 'i1'   : self.a
                                    , 'nq'   : d_aux
                                    , 'vdd'  : self.vdd
                                    , 'vss'  : self.vss
                                    }
                           )

    Generate ( "DpgenOr2", "or2%d" % self.nbit
             , param = { 'nbit'     : self.nbit
                       , 'physical' : True
                       }
             )
    self.instOr2   = Inst ( "or2%d" % self.nbit, "instance_or2"
                          , map   = { 'i0'   : d_aux
                                    , 'i1'   : self.b
                                    , 'q'    : e_aux
                                    , 'vdd'  : self.vdd
                                    , 'vss'  : self.vss
                                    }
                          )
```

```

    Generate ( "DpgenAdsb2f", "adder%d" % self.nbit
             , param = { 'nbit'     : self.nbit
                       , 'physical' : True
                       }
             )
    self.instAdd2  = Inst ( "adder%d" % self.nbit, "instance_add2"
                          , map   = { 'i0'      : e_aux
                                    , 'i1'      : self.c
                                    , 'q'       : self.s
                                    , 'add_sub' : self.cmd
                                    , 'c31'     : self.cout
                                    , 'c30'     : ovr
                                    , 'vdd'     : self.vdd
                                    , 'vss'     : self.vss
                                    }
                          )

  def Layout ( self ) :
    Place      ( self.instNand2, NOSYM, XY ( 0, 0 ) )
    PlaceRight ( self.instOr2,   NOSYM )
    PlaceRight ( self.instAdd2,  NOSYM )
```

2.2.4 Creation of the circuit : file test.py

```
#!/usr/bin/env python

from stratus import *
from addaccu import addaccu

nbit = Param ( "n" )

dict = { 'nbit' : nbit }

inst_addaccu = addaccu ( "inst_addaccu", dict )

inst_addaccu.Interface()
inst_addaccu.Netlist()
inst_addaccu.Layout()
inst_addaccu.View()

inst_addaccu.Save ( PHYSICAL )
```

2.2.5 How to execute the file

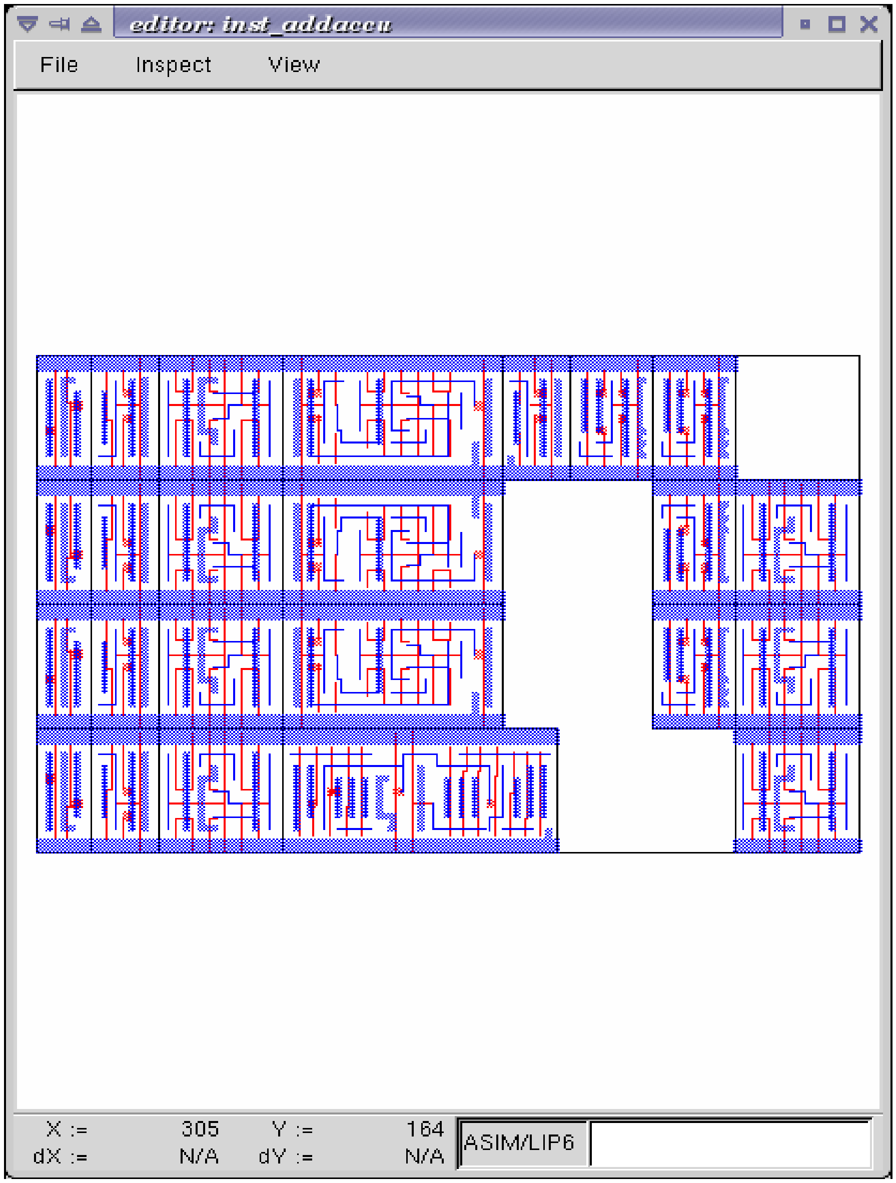
```
python test.py -n 4
```

or :

```
chmod u+x test.py
./test -n 4
```

2.2.6 The editor

The method `View` permits to open an editor in which one can see the cell being created as shown in the picture below.



2.2.7 Function Param

This function allows the user to give parameters when creating a cell.

When one wants to give values to two parameters, one can type on the shell :

```
python test.py -n 4 -w 8
```

The file `test.py` has then to contain :

```
nbit, nword = Param ( "n", "w" )
```

The letters typed on the shell must be the ones given as parameters of function `Param`.

2.2.8 How to instanciate your generator in another generator

One can create a generator and instantiate it in another generator.

To do that, the model name of the generator must have the form : "file_name.class_name".

Note that if the two generators are not in the same directory, the directory of the generator to be instantiated has to be added in the CRL_CATA_LIB environment variable.

For example, in order to instanciate the addaccu created above in a cell :

```
n = 4
Generate ( "addaccu.addaccu", "my_addaccu_%dbits" % n
         , param = { 'nbit' : n } )

Inst ( "my_addaccu_%dbits" % n
     , map = { 'a'    : self.netA
             , 'b'    : self.netB
             , 'c'    : self.netC
             , 'v'    : self.netV
             , 'cmd'  : self.netCmd
             , 'cout' : self.netCout
             , 's'    : self.netS
             , 'vdd'  : self.vdd
             , 'vss'  : self.vss
             }
     )
```

3 Description of a netlist

3.1 Nets

3.1.1 Name

SignalIn, SignalOut ... – Creation of nets

3.1.2 Synopsys

```
netA = SignalIn ( "a", 4 )
```

3.1.3 Description

How to create and use nets.

3.1.4 Nets

Differents kind of nets are listed below :

- `SignalIn` : Creation of an input port
- `SignalOut` : Creation of an output port
- `SignalInOut` : Creation of an inout port
- `SignalUnknown` : Creation of an input/output port which direction is not defined
- `TriState` : Creation of a tristate port
- `CkIn` : Creation of a clock port
- `VddIn` : Creation of the vdd alimentation
- `VssIn` : Creation of the vss alimentation
- `Signal` : Creation of an internal net

3.1.5 Parameters

All kind of constructors have the same parameters :

- `name` : the name of the net (mandatory argument)
- `arity` : the arity of the net (mandatory argument)
- `indice` : for bit vectors only : the LSB bit (optional argument : set to 0 by default)

Only `CkIn`, `VddIn` and `VssIn` do not have the same parameters : there is only the `name` parameter (they are 1 bit nets).

3.1.6 Functions and methods

Some functions/methods are provided in order to handle nets :

- function `Cat` : Concatenation of nets, beginning with the MSB

```
Inst ( 'DpgenInv'
     , map = { 'i0'  : Cat ( A, B )
             , 'nq'  : S
             , 'vdd' : vdd
             , 'vss' : vss
             }
     )
```

Or :

```
tab = []
tab.append ( A )
tab.append ( B )

Inst ( 'DpgenInv'
     , map = { 'i0'  : Cat ( tab )
             , 'nq'  : S
             , 'vdd' : vdd
             , 'vss' : vss
             }
     )
```

If A and B are 2 bits nets, the net `myNet` will be such as :

```
myNet[3] = A[1]
myNet[2] = A[0]
myNet[1] = B[1]
myNet[0] = B[0]
```

- function `Extend` : Creation of a net which is an extension of the net which it is applied to

```
temp    = Signal (     "temp", 5 )
tempExt = Signal ( "temp_ext", 8 )

tempExt <= temp.Extand ( 8, 'one' )
```

- method `Alias` : Creation of an alias name for a net

```
cin.Alias  ( c_temp[0] )
cout.Alias ( c_temp[4] )
for i in range ( 4 ) :
  Inst ( "Fulladder"
       , map = { 'a'    : a[i]
               , 'b'    : b[i]
               , 'cin'  : c_temp[i]
               , 'sout' : sout[i]
               , 'cout' : c_temp[i+1]
               , 'vdd'  : vdd
               , 'vss'  : vss
               }
       )
```

3.1.7 Errors

Some errors may occur :

- `Error in SignalIn :`
  `the lenght of the net must be a positive value.`
  One can not create a net with a negative lenght.

3.2 Instances

3.2.1 Name

Inst – Creation of instances

3.2.2 Synopsys

```
Inst ( model
     , name
     , map = connectmap
     )
```

3.2.3 Description

Instantiation of an instance. The type of the instance is given by the `model` parameter. The connexions are made thanks to the `connectmap` parameters.

3.2.4 Parameters

- `Model` : Name of the mastercell of the instance to create (mandatory argument)
- `name` : Name of the instance (optional)
  When this argument is not defined, the instance has a name created by default. This argument is usefull when one wants to create a layout as well. Indeed, the placement of the instances is much easier when the conceptor has chosen himself the name f the instances.</para>
- `connectmap` : Connexions in order to make the netlist

`param` and `map` are dictionnaries as shown in the example below.

3.2.5 Example

```
Inst ( 'a2_x2'
     , map = { 'i0'  : in0
             , 'i1'  : in1
```

```
            , 'q'   : out
            , 'vdd' : vdd
            , 'vss' : vss
            }
    )
```

3.2.6 Errors

Some errors may occur :

- `Error in Inst : the model Model does not exist.`
  `Check CRL_CATA_LIB.`
  Either one has made a mistake in the name of the model, either the environment variable is not correct.

- `Error in Inst : port does not exist in model Model.`
  One port in map is not correct.

- `Error in Inst : one input net is not dimensionned.`
  The size of the output nets is automatically calculated bus the input nets must be dimensionned before being connected.

3.3 Generators

3.3.1 Name

Generate – Interface with the generators

3.3.2 Synopsys

```
Generate ( model, modelname, param = dict )
```

3.3.3 Description

The `Generate` function call is the generic interface to all generators.

3.3.4 Arguments

- `model` : Specifies which generator is to be invoked
  - If the generator belongs to the Dpgen library provided by Stratus, the model name of the generator is simply the name of the class of the generator.
  - If the generator is created by the user, the model name of the generator must have the form : "file_name.class_name". (Note that if the the generator is not in the working directory, the directory of the generator to be instantiated has to be added in the CRL_CATA_LIB environment variable)

- `modelname` : Specifies the name of the model to be generated
- `dict` : Specifies the parameters of the generator

3.3.5 Parameters

Every generator has it's own parameters. They must be described in the map `dict`.

Every generator provides a netlist view. Two other views can be generated, if they are provided by the generator. Two parameters have to be given, in order to choose those views :

- 'physical' : True/False, generation of the physical view (optionnal, False by default)
- 'behavioral' : True/False, generation of the behavioral view (optionnal, False by default)

3.3.6 Errors

Some errors may occur :

- `[Stratus ERROR] Generate : the model must be described in a string.`

4 Description of a layout

4.1 Place

4.1.1 Name

Place – Places an instance

4.1.2 Synopsys

```
Place ( ins, sym, point )
```

4.1.3 Description

Placement of an instance.

The instance has to be instantiated in the method `Netlist`, in order to use the `Place` function.

4.1.4 Parameters

- `ins` : Instance to place.

- `sym` : Geometrical operation to be performed on the instance before beeing placed.
  The `sym` argument can take eight legal values :
  - `NOSYM` : no geometrical operation is performed
  - `SYM_Y` : Y becomes -Y, that means toward X axe symetry
  - `SYM_X` : X becomes -X, that means toward Y axe symetry
  - `SYMXY` : X becomes -X, Y becomes -Y
  - `ROT_P` : a positive 90 degrees rotation takes place
  - `ROT_M` : a negative 90 degrees rotation takes place
  - `SY_RP` : Y becomes -Y, and then a positive 90 degrees rotation takes place
  - `SY_RM` : Y becomes -Y, and then a negative 90 degrees rotation takes place
- `point` : coordinates of the lower left corner of the abutment box of the instance in the current figure.

4.1.5 Example

```
Place ( myInst, NOSYM, XY ( 0, 0 ) )
```

4.1.6 Errors

Some errors may occur :

- `[Stratus ERROR] Placement : the instance doesn't exist.`
  The instance must be instanciated in order to be placed.
- `[Stratus ERROR] Placement : the first argument is not an instance.`
- `[Stratus ERROR] Placement : the instance is already placed.`
  One can not place an instance twice
- `[Stratus ERROR] Place : wrong argument for placement type.`
  The symetry given as argument is not correct.
- `[Stratus ERROR] Place : wrong argument for placement, the coordinates must be put in a XY object.`
  The coordinates are not descrobed the bood way.

4.2 PlaceTop

4.2.1 Name

PlaceTop – Places an instance at the top of the "reference instance"

4.2.2 Synopsys

```
PlaceTop ( ins, sym, offsetX, offsetY )
```

4.2.3 Description

Placement of an instance.

The instance has to be instantiated in the method `Netlist` in order to use the `PlaceTop` function.

The bottom left corner of the abutment box of the instance is placed, after beeing symetrized and/or rotated, toward the top left corner of the abutment box of the "reference instance". The newly placed instance becomes the "reference instance".

4.2.4 Parameters

- `ins` : Instance to place.
- `sym` : Geometrical operation to be performed on the instance before beeing placed.
  The `sym` argument can take eight legal values :
  - `NOSYM` : no geometrical operation is performed
  - `SYM_Y` : Y becomes -Y, that means toward X axe symetry
  - `SYM_X` : X becomes -X, that means toward Y axe symetry
  - `SYMXY` : X becomes -X, Y becomes -Y
  - `ROT_P` : a positive 90 degrees rotation takes place
  - `ROT_M` : a negative 90 degrees rotation takes place
  - `SY_RP` : Y becomes -Y, and then a positive 90 degrees rotation takes place
  - `SY_RM` : Y becomes -Y, and then a negative 90 degrees rotation takes place
- `offsetX` (optionnal) : An offset is put horizontally. The value given as argument must be a multiple of PITCH
- `offsetY` (optionnal) : An offset is put vertically. The value given as argument must be a multiple of SLICE

4.2.5 Example

```
Place    ( myInst1, NOSYM, 0, 0 )
PlaceTop ( myInst2, SYM_Y )
```

4.2.6 Errors

Some errors may occur :

- `[Stratus ERROR] Placement : the instance doesn't exist.`
  The instance must be instanciated in order to be placed.
- `[Stratus ERROR] Placement : the first argument is not an instance.`
- `[Stratus ERROR] Placement : the instance is already placed.`
  One can not place an instance twice
- `[Stratus ERROR] PlaceTop : no previous instance.`
  One can use `PlaceTop` only if a reference instance exist. Use a `Place` call before.
- `[Stratus ERROR] PlaceTop : wrong argument for placement type.`
  The symetry given as argument is not correct.

4.3 PlaceBottom

4.3.1 Name

PlaceBottom – Places an instance below the "reference instance"

4.3.2 Synopsys

```
PlaceBottom ( ins, sym, offsetX, offsetY )
```

4.3.3 Description

Placement of an instance.

The instance has to be instantiated in the method `Netlist` in order to use the `PlaceTop` function.

The top left corner of the abutment box of the instance is placed, after beeing symetrized and/or rotated, toward the bottom left corner of the abutment box of the "reference instance". The newly placed instance becomes the "reference instance".

4.3.4 Parameters

- `ins` : Instance to place.
- `sym` : Geometrical operation to be performed on the instance before beeing placed.
  The `sym` argument can take eight legal values :
  - `NOSYM` : no geometrical operation is performed

- SYM_Y : Y becomes -Y, that means toward X axe symetry
- SYM_X : X becomes -X, that means toward Y axe symetry
- SYMXY : X becomes -X, Y becomes -Y
- ROT_P : a positive 90 degrees rotation takes place
- ROT_M : a negative 90 degrees rotation takes place
- SY_RP : Y becomes -Y, and then a positive 90 degrees rotation takes place
- SY_RM : Y becomes -Y, and then a negative 90 degrees rotation takes place

- `offsetX` (optionnal) : An offset is put horizontally. The value given as argument must be a multiple of PITCH
- `offsetY` (optionnal) : An offset is put vertically. The value given as argument must be a multiple of SLICE

4.3.5 Example

```
Place       ( myInst1, NOSYM, 0, 0 )
PlaceBottom ( myInst2, SYM_Y       )
```

4.3.6 Errors

Some errors may occur :

- `[Stratus ERROR] Placement : the instance doesn't exist.`
  The instance must be instanciated in order to be placed.
- `[Stratus ERROR] Placement : the first argument is not an instance.`
- `[Stratus ERROR] Placement : the instance is already placed.`
  One can not place an instance twice
- `[Stratus ERROR] PlaceBottom : no previous instance.`
  One can use `PlaceBottom` only if a reference instance exist. Use a `Place` call before.
- `[Stratus ERROR] PlaceBottom : wrong argument for placement type.`
  The symetry given as argument is not correct.

4.4 PlaceRight

4.4.1 Name

PlaceRight – Places an instance at the right of the "reference instance"

4.4.2 Synopsys

```
PlaceRight ( ins, sym, offsetX, offsetY )
```

4.4.3 Description

Placement of an instance.

The instance has to be instantiated in the method `Netlist` in order to use the `PlaceTop` function.

The bottom left corner of the abutment box of the instance is placed, after beeing symetrized and/or rotated, toward the bottom right corner of the abutment box of the "reference instance". The newly placed instance becomes the "reference instance".

4.4.4 Parameters

- `ins` : Instance to place.
- `sym` : Geometrical operation to be performed on the instance before beeing placed.
  The `sym` argument can take eight legal values :
  - `NOSYM` : no geometrical operation is performed
  - `SYM_Y` : Y becomes -Y, that means toward X axe symetry
  - `SYM_X` : X becomes -X, that means toward Y axe symetry
  - `SYMXY` : X becomes -X, Y becomes -Y
  - `ROT_P` : a positive 90 degrees rotation takes place
  - `ROT_M` : a negative 90 degrees rotation takes place
  - `SY_RP` : Y becomes -Y, and then a positive 90 degrees rotation takes place
  - `SY_RM` : Y becomes -Y, and then a negative 90 degrees rotation takes place
- `offsetX` (optionnal) : An offset is put horizontally. The value given as argument must be a multiple of PITCH
- `offsetY` (optionnal) : An offset is put vertically. The value given as argument must be a multiple of SLICE

4.4.5 Example

```
Place      ( myInst1, NOSYM, 0, 0 )
PlaceRight ( myInst2, NOSYM )
```

4.4.6 Errors

Some errors may occur :

- `[Stratus ERROR] Placement : the instance doesn't exist.`
  The instance must be instanciated in order to be placed.
- `[Stratus ERROR] Placement : the first argument is not an instance.`
- `[Stratus ERROR] Placement : the instance is already placed.`
  One can not place an instance twice
- `[Stratus ERROR] PlaceRight : no previous instance.`
  One can use `PlaceRight` only if a reference instance exist. Use a `Place` call before.
- `[Stratus ERROR] PlaceRight : wrong argument for placement type.`
  The symetry given as argument is not correct.

4.5 PlaceLeft

4.5.1 Name

PlaceLeft – Places an instance at the left of the "reference instance"

4.5.2 Synopsys

```
PlaceLeft ( ins, sym, offsetX, offsetY )
```

4.5.3 Description

Placement of an instance.

The instance has to be instantiated in the method `Netlist` in order to use the `PlaceTop` function.

The bottom right corner of the abutment box of the instance is placed, after beeing symetrized and/or rotated, toward the bottom left corner of the abutment box of the "reference instance". The newly placed instance becomes the "reference instance".

4.5.4 Parameters

- `ins` : Instance to place.
- `sym` : Geometrical operation to be performed on the instance before beeing placed.
  The `sym` argument can take eight legal values :
  - `NOSYM` : no geometrical operation is performed

– `SYM_Y` : Y becomes -Y, that means toward X axe symetry
– `SYM_X` : X becomes -X, that means toward Y axe symetry
– `SYMXY` : X becomes -X, Y becomes -Y
– `ROT_P` : a positive 90 degrees rotation takes place
– `ROT_M` : a negative 90 degrees rotation takes place
– `SY_RP` : Y becomes -Y, and then a positive 90 degrees rotation takes place
– `SY_RM` : Y becomes -Y, and then a negative 90 degrees rotation takes place

- `offsetX` (optionnal) : An offset is put horizontally. The value given as argument must be a multiple of PITCH
- `offsetY` (optionnal) : An offset is put vertically. The value given as argument must be a multiple of SLICE

4.5.5 Example

```
Place     ( myInst1, NOSYM, 0, 0 )
PlaceLeft ( myInst2, NOSYM )
```

4.5.6 Errors

Some errors may occur :

- `[Stratus ERROR] Placement : the instance doesn't exist.`
  The instance must be instanciated in order to be placed.
- `[Stratus ERROR] Placement : the first argument is not an instance.`
- `[Stratus ERROR] Placement : the instance is already placed.`
  One can not place an instance twice
- `[Stratus ERROR] PlaceLeft : no previous instance.`
  One can use `PlaceLeft` only if a reference instance exist. Use a `Place` call before.
- `[Stratus ERROR] PlaceLeft : wrong argument for placement type.`
  The symetry given as argument is not correct.

4.6 SetRefIns

4.6.1 Name

SetRefIns – Defines the new "reference instance" for placement

4.6.2 Synopsys

```
SetRefIns ( ins )
```

4.6.3 Description

This function defines the new "reference instance", used as starting point in the relative placement functions.

It's regarding the abutmentbox of the instance `ins` that the next instance is going to be placed, if using the appropriate functions.

Note that the more recently placed instance becomes automaticaly the "reference instance", if SetRefIns isn't called.

4.6.4 Parameters

- `ins` : defines the new "reference instance"

4.6.5 Example

```
Place      ( myInst1, NOSYM, 0, 0 )
PlaceRight ( myInst2, NOSYM       )

SetRefIns  ( myInst1 )
PlaceTop   ( myInst3, SYM_Y       )
```

`myInst3` is on top of `myInst1` instead of `myInst2`.

4.6.6 Errors

Some errors may occur :

- `[Stratus ERROR] SetRefIns : the instance doesn't exist.`
  If the instance has not been instanciated, it is impossible do to any placement from it.
- `[Stratus ERROR] SetRefIns : the instance ...is not placed.`
  If the instance has not been placed, it is impossible do to any placement from it.

4.7 DefAb

4.7.1 Name

DefAb – Creates the abutment box of the current cell

4.7.2 Synopsys

```
DefAb ( point1, point2 )
```

4.7.3 Description

This function creates the abutment box of the current cell.

Note that one does not have to call this function before saving in order to create the abutment box. The abutment box is created nevertheless (given to placed instances). This function is usefull if one wants to create an abutment before placing the instances.

4.7.4 Parameters

- `point1` : coordinates of the bottom left corner of the created abutment box.
- `point2` : coordinates of the top right corner of the created abutment box.

4.7.5 Example

```
DefAb ( XY(0, 0), XY(500, 100) )

Place ( self.inst, NOSYM, XY(0, 0) )
```

4.7.6 Errors

Some errors may occur :

- `[Stratus ERROR] DefAb : an abutment box already exists.`
  `Maybe you should use ResizeAb function.`
  One has called DefAb but the current cell already has an abutment box.
  In order to modify the current abutment box, the function to call is ResizeAb.
- `[Stratus ERROR] DefAb : wrong argument,`
  `the coordinates must be put in a XY object.`
  The type of one of the arguments is not correct. Coordinates must be put in a `XY` object.
- `[Stratus ERROR] DefAb :`
  `Coordinates of an abutment Box in y must be multiple of the slice.`
  `Coordinates of an abutment Box in x must be multiple of the pitch.`
  One has called DefAb with non authorized values.

4.8 ResizeAb

4.8.1 Name

ResizeAb – Modifies the abutment box of the current cell

4.8.2 Synopsys

```
ResizeAb ( dx1, dy1, dx2, dy2 )
```

4.8.3 Description

This function modifies the abutment box of the current cell.

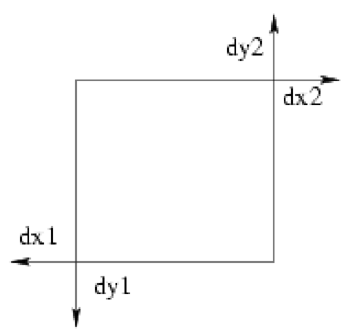
The coordinates of the abutment box are the coordinates of the envelop of the abutment boxes of each instance plus the delta values given as argument.

Note that one can not call this function in order to create the abutment box. This fonction only modifies the already created abutment box.

4.8.4 Parameters

- `(dx1, dy1)` : Values to be substracted to the lower left corner of the previous abutment box.
- `(dx2, dy2)` : Values to be added to the upper right corner of the previous abutment box.

The Values are used as follow :



4.8.5 Example

```
% Expansion of the abutment box at the top and the bottom
ResizeAb ( 0, 100, 0, 100 )
```

4.8.6 Errors

Some errors may occur :

- `[Stratus ERROR] ResizeAb :`
  `Coordinates of an abutment Box in y must be multiple of the slice.`
  `Coordinates of an abutment Box in x must be multiple of the pitch.`
  One has called ResizeAb with non authorized values

- `[Stratus ERROR] ResizeAb :`
  `one of the values of dx1 or dx2 (dy1 or dy2)  is incompatible with the size of the abutment box.`
  `Coordinates of an abutment Box in x must be multiple of the pitch.`
  One has called ResizeAb with a value which deteriorates the abtument box

5 Description of the stimuli

The stimuli used for the simulation are described in a `Stimuli` method. This method is a Python function generator that is automatically called by the `Testbench` method to generate all the stimuli. As a Python function generator, the `yield` instruction have to be used at the end of each stimuli computation.

5.0.1 Affect value to signals

The method `affect` permits to affect a value to a given signal as follow

```
self._stim.affect(self.Ck,0)
```

5.0.2 Add stimuli

The method `add` permits to finish a step of simulation by add all the values to the current stimuli

```
self._stim.add()
```

6 Place and Route

6.1 PlaceSegment

6.1.1 Name

PlaceSegment – Places a segment

6.1.2 Synopsys

```
PlaceSegment ( net, layer, point1, point2, width )
```

6.1.3 Description

Placement of a segment.

The segment is created between `point1` and `point2` on the layer `layer` and with width `width`. It belongs to the net `net`.

Note that the segment must be horizontal or vertival.

6.1.4 Parameters

- `net` : Net which the segment belongs to
- `layer` : Layer of the segment.
  The `layer` argument is a string wich can take different values, thanks to the technology (file described in HUR_TECHNO_NAME)
  - NWELL, PWELL, ptie, ntie, pdif, ndif, ntrans, ptrans, poly, ALU1, ALU2, ALU3, ALU4, ALU5, ALU6, VIA1, VIA2, VIA3, VIA4, VIA5, TEXT, UNDEF, SPL1, TALU1, TALU2, TALU3, TALU4, TALU5, TALU6, POLY, NTIE, PTIE, NDIF, PDIF, PTRANS, NTRANS, CALU1, CALU2, CALU3, CALU4, CALU5, CALU6, CONT_POLY, CONT_DIF_N, CONT_DIF_P, CONT_BODY_N, CONT_BODY_P, via12, via23, via34, via45, via56, via24, via25, via26, via35, via36, via46, CONT_TURN1, CONT_TURN2, CONT_TURN3, CONT_TURN4, CONT_TURN5, CONT_TURN6
- `point1,point2` : The segment is created between those two points

6.1.5 Example

```
PlaceSegment ( myNet, "ALU3", XY (10, 0), XY (10, 100), 2 )
```

6.1.6 Errors

Some errors may occur :

- `[Stratus ERROR] PlaceSegment : Argument layer must be a string.`
- `[Stratus ERROR] PlaceSegment : Wrong argument,`
  `the coordinates of the segment must be put in XY objects.`
- `[Stratus ERROR] PlaceSegment : Segments are vertical or horizontal.`
  The two references given as argument do not describe a vertical or horizontal segment. Wether coordinate x or y of the references must be identical.

6.2 PlaceContact

6.2.1 Name

PlaceContact – Places a contact

6.2.2 Synopsys

```
PlaceContact ( net, layer, point, width, height )
```

6.2.3 Description

Placement of a contact.

The contact is located at the coodinates of `point`, on the layer `layer` and has a size of 1 per 1. It belongs to the net `net`.

Note that the segment must be horizontal or vertival.

6.2.4 Parameters

- `net` : Net which the contact belongs to
- `layer` : Layer of the segment.
  The `layer` argument is a string wich can take different values, thanks to the technology (file described in HUR_TECHNO_NAME)
  - NWELL, PWELL, ptie, ntie, pdif, ndif, ntrans, ptrans, poly, ALU1, ALU2, ALU3, ALU4, ALU5, ALU6, VIA1, VIA2, VIA3, VIA4, VIA5, TEXT, UNDEF, SPL1, TALU1, TALU2, TALU3, TALU4, TALU5, TALU6, POLY, NTIE, PTIE, NDIF, PDIF, PTRANS, NTRANS, CALU1, CALU2, CALU3, CALU4, CALU5, CALU6, CONT_POLY, CONT_DIF_N, CONT_DIF_P, CONT_BODY_N, CONT_BODY_P, via12, via23, via34, via45, via56, via24, via25, via26, via35, via36, via46, CONT_TURN1, CONT_TURN2, CONT_TURN3, CONT_TURN4, CONT_TURN5, CONT_TURN6
- `point` : Coodinates of the contact
- `width` : Width of the contact
- `height` : Height of the contact

6.2.5 Example

```
PlaceContact ( myNet, "ALU2", XY (10, 0), 2, 2 )
```

6.2.6 Errors

Some errors may occur :

- `[Stratus ERROR] PlaceContact : Argument layer must be a string.`
- `[Stratus ERROR] PlaceContact : Wrong argument,`
  `the coordinates of the contact must be put in a XY object.`

6.3 PlacePin

6.3.1 Name

PlacePin – Places a pin

6.3.2 Synopsys

```
PlacePin ( net, layer, direction, point, width, height )
```

6.3.3 Description

Placement of a pin.

The pin is located at the coodinates of `point`, on the layer `layer`, has a a direction of `direction` and size of 1 per 1. It belongs to the net `net`.

6.3.4 Parameters

- `net` : Net which the pin belongs to
- `layer` : Layer of the segment.
  The `layer` argument is a string wich can take different values, thanks to the technology (file described in HUR_TECHNO_NAME)
  - NWELL, PWELL, ptie, ntie, pdif, ndif, ntrans, ptrans, poly, ALU1, ALU2, ALU3, ALU4, ALU5, ALU6, VIA1, VIA2, VIA3, VIA4, VIA5, TEXT, UNDEF, SPL1, TALU1, TALU2, TALU3, TALU4, TALU5, TALU6, POLY, NTIE, PTIE, NDIF, PDIF, PTRANS, NTRANS, CALU1, CALU2, CALU3, CALU4, CALU5, CALU6, CONT_POLY, CONT_DIF_N, CONT_DIF_P, CONT_BODY_N, CONT_BODY_P, via12, via23, via34, via45, via56, via24, via25, via26, via35, via36, via46, CONT_TURN1, CONT_TURN2, CONT_TURN3, CONT_TURN4, CONT_TURN5, CONT_TURN6
- `direction` : Direction of the pin
  - UNDEFINED, NORTH, SOUTH, EAST, WEST
- `point` : Coodinates of the pin
- `width` : Width of the pin
- `height` : Height of the pin

6.3.5 Example

```
PlacePin ( myNet, "ALU2", NORTH, XY (10, 0), 2, 2 )
```

6.3.6 Errors

Some errors may occur :

- `[Stratus ERROR] PlacePin : Argument layer must be a string.`

- `[Stratus ERROR] PlacePin : Illegal pin access direction.`
  `The values are : UNDEFINED, NORTH, SOUTH, EAST, WEST.`
- `[Stratus ERROR] PlacePin : Wrong argument,`
  `the coordinates of the pin must be put in a XY object.`

6.4 PlaceRef

6.4.1 Name

PlaceRef – Places a reference

6.4.2 Synopsys

```
PlaceRef ( point, name )
```

6.4.3 Description

Placement of a reference.
The reference is located at the coordinates of `point`, with name `name`.

6.4.4 Parameters

- `point` : Coodinates of the reference
- `name` : Name of the reference

6.4.5 Example

```
PlaceRef ( XY (10, 0), "myref" )
```

6.4.6 Errors

Some errors may occur :

- `[Stratus ERROR] PlaceRef : Wrong argument,`
  `the coordinates of the reference must be put in a XY object.`
- `[Stratus ERROR] PlaceRef : Argument layer must be a string.`

6.5 GetRefXY

6.5.1 Name

GetRefXY – Returns the coordinates of a reference

6.5.2 Synopsys

```
GetRefXY ( pathname, refname )
```

6.5.3 Description

Computation of coordinates.

The point returned (object XY) represents the location of the reference of name `refname` within the coodinates system of the top cell. The reference `refname` is instanciated in an instance found thanks to `pathname` which represents an ordered sequence of instances through the hierarchy.

6.5.4 Parameters

- `pathname` : The path in order to obtain, from the top cell, the instance the reference `refname` belongs to
- `refname` : The name of the reference

6.5.5 Example

The cell which is being created (the top cell), instanciates a generator with instance name "my_dpgen_and2". This generator instanciates an instance called "cell_1" which the reference "i0_20" belongs to.

```
GetRefXY ( "my_dpgen_and2.cell_1", "i0_20" )
```

6.5.6 Errors

Some errors may occur :

- `[Stratus ERROR] GetRefXY :`
  `The instance's path must be put with a string.`
- `[Stratus ERROR] GetRefXY :`
  `The reference must be done with it's name : a string.`
- `[Stratus ERROR] GetRefXY :`
  `No reference found with name ... in masterCell ...`

6.6 CopyUpSegment

6.6.1 Name

CopyUpSegment – Copies the segment of an instance in the current cell

6.6.2 Synopsys

```
CopyUpSegment ( pathname, netname, newnet  )
```

6.6.3 Description

Duplication of a segment.

The segment is created with the same cordinates and layer as the segment corresponding to the net `netname` in the instance found thanks to `pathname`. It belongs to the net `newnet`.

Note that if several segments correspond to the net, they are all going to be copied.

6.6.4 Parameters

- `pathname` : The path in order to obtain, from the top cell, the instance the net `netname` belongs to
- `netname` : The name of the net which the segment belongs to
- `net` : The net which the top cell segment os going to belong to

6.6.5 Example

```
CopuUpSegment ( "my_dpgen_and2.cell_1", "i0", myNet )
```

6.6.6 Errors

Some errors may occur :

- `[Stratus ERROR] CopyUpSegment :`
  `The instance's path must be put with a string.`
- `[Stratus ERROR] CopyUpSegment :`
  `The segment must be done with it's name : a string.`
- `[Stratus ERROR] CopyUpSegment :`
  `No net found with name ... in masterCell ...`
  There is no net with name `netname` in the instance found thanks to the path `pathname`.
- `[Stratus ERROR] CopyUpSegment :`
  `No segment found with net ... in masterCell ...`
  The net with name `netname` has no segment. So the copy of segment can not be done.

- `[Stratus ERROR] CopyUpSegment :`
  `the segment of net ... are not of type CALU.`
  In other words, the net is not an external net. The copy can be done only with external nets.

6.7 PlaceCentric

6.7.1 Name

PlaceCentric – Placement of an instance in the middle of an abutment box

6.7.2 Synopsys

```
PlaceCentric ( ins )
```

6.7.3 Description

This function places an instance in the middle of and abutment box.

The instance has to be instantiated in the method `Netlist` in order to use this function.

6.7.4 Parameters

- `ins` : Instance to place

6.7.5 Errors

Some errors may occur :

- `[Stratus ERROR] PlaceCentric: the instance does not exist.`
  The instance must be instanciated in order to be placed.

- `[Stratus ERROR] PlaceCentric :`
  `the instance's size is greater than this model.`
  The instance must fit in the abutment box. The abutment box may not be big enough.

6.8 PlaceGlu

6.8.1 Name

PlaceGlue – Automatic placement of non placed instances

6.8.2 Synopsys

```
PlaceGlue ( cell )
```

6.8.3 Description

This function places, thanks to the automatic placer Mistral of Coriolis, all the non placed instances of the cell.

6.8.4 Parameters

- `cell` : the cell which the fonction is applied to

6.9 FillCell

6.9.1 Name

FillCell – Automatic placement of ties.

6.9.2 Synopsys

```
FillCell ( cell )
```

6.9.3 Description

This function places automatically ties.

6.9.4 Parameters

- `cell` : the cell which the fonction is applied to

6.9.5 Errors

Some errors may occur :

- `[Stratus ERROR] FillCell : Given cell doesn't exist.`
  The argument is wrong. Check if one has created the cell correctly.

6.10 Pads

6.10.1 Name

PadNorth, PadSouth, PadEast, PasWest – Placement of pads at the periphery of the cell

6.10.2 Synopsys

```
PadNorth ( args )
```

6.10.3 Description

These functions place the pads given as arguments at the given side of the cell (PadNorth : up north, PadSouth : down south ...). Pads are placed from bottom to top for PadNorth and PadSouth and from left to right for PadWest and PasEast.

6.10.4 Parameters

- `args` : List of pads to be placed

6.10.5 Example

```
PadSouth ( self.p_cin, self.p_np, self.p_ng, self.p_vssick0
         , self.p_vddeck0, self.p_vsseck1, self.p_vddeck1, self.p_cout
         , self.p_y[0], self.p_y[1], self.p_y[2]
         )
```

6.10.6 Errors

Some errors may occur :

- `[Stratus ERROR] PadNorth : not enough space for all pads.`
  The abutment box is not big enough in order to place all the pads. Maybe one could put pads on other faces of the cell.
- `[Stratus ERROR] PadNorth : one instance doesn't exist.`
  One of the pads given as arguments does not exist
- `[Stratus ERROR] PadNorth : one argument is not an instance.`
  One of the pads is not one of the pads of the cell.
- `[Stratus ERROR] PadNorth : the instance ins is already placed.`
  One is trying to place a pad twice.
- `[Stratus ERROR] PadNorth : pad ins must be closer to the center.`
  The pad name ins must be put closer to the center in order to route the cell

6.11 Alimentation rails

6.11.1 Name

AlimVerticalRail, AlimHorizontalRail – Placement of a vertical/horizontal alimentation call back

6.11.2 Synopsys

```
AlimVerticalRail ( nb )
```

6.11.3 Description

These functions place a vertical/horizontal alimentation call back. It's position is given by the parameter given.

6.11.4 Parameters

- `nb` : coordinate of the rail
  - For AlimVerticalRail, `nb` is in pitches i.e. 5 lambdas
  - For AlimHorizontalRail, `nb` is in slices i.e. 50 lambdas

6.11.5 Example

```
AlimVerticalRail   (  50 )
AlimVerticalRail   ( 150 )

AlimHorizontalRail (  10 )
```

6.11.6 Errors

Some errors may occur :

- `[Stratus ERROR] AlimHorizontalRail :`
  `Illegal argument y, y must be between ... and ...`
  The argument given is wrong : the call back would not be in the abutment box.
- `[Stratus ERROR] Placement of cells :`
  `please check your file of layout with DRUC.`
  The placement of the cell needs to be correct in order to place a call back. Check the errors of placement.

6.12 Alimentation connectors

6.12.1 Name

AlimConnectors – Creation of connectors at the periphery of the core of a circuit

6.12.2 Synopsys

```
AlimConnectors()
```

6.12.3 Description

This function creates the connectors in Alu 1 at the periphery of the core.

6.13 PowerRing

6.13.1 Name

PowerRing – Placement of power rings.

6.13.2 Synopsys

```
PowerRing ( nb )
```

6.13.3 Description

This function places power rings around the core and around the plots.

6.13.4 Parameters

- `nb` : Number of pair of rings vdd/vss

6.13.5 Example

```
PowerRing ( 3 )
```

6.13.6 Errors

Some errors may occur :

- `[Stratus ERROR] PowerRing : Pads in the north haven't been placed.` The pads of the 4 sides of the chip must be placed before calling function PowerRing.
- `[Stratus ERROR] PowerRing : too many rings, not enough space.` Wether The argument of PowerRing is to big, or the abutment box of the chip is to small. There's no space to put the rings.

6.14 RouteCk

6.14.1 Name

RouteCk – Routing of signal Ck to standard cells

6.14.2 Synopsys

```
RouteCk ( net )
```

6.14.3 Description

This function routes signal Ck to standard cells.

6.14.4 Parameters

- `net` : the net which the fonction is applied to

6.14.5 Errors

Some errors may occur :

- `[Stratus ERROR] RouteCk : Pads in the north haven't been placed`
  The pads must be placed before calling RoutageCk.

7 Instanciation facilities

7.1 Buffer

7.1.1 Name

Buffer – Easy way to instantiate a buffer

7.1.2 Synopsys

```
netOut <= netIn.Buffer()
```

7.1.3 Description

This method is a method of net. The net which this method is applied to is the input net of the buffer. The method returns a net : the output net.

Note that it is possible to change the generator instanciated with the `SetBuff` method.

7.1.4 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.A = SignalIn  ( "a", 4 )

    self.S = SignalOut ( "s", 4 )

    self.Vdd = VddIn  ( "vdd" )
    self.Vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.S <= self.A.Buffer()
```

7.2 Multiplexor

7.2.1 Name

Mux – Easy way to instantiate a multiplexor

7.2.2 Synopsys

```
netOut <= netCmd.Mux ( arg )
```

7.2.3 Description

This method is a method of net. The net which this method is applied to is the command of the multiplexor. The nets given as parameters are all the input nets. This method returns a net : the output net.
There are two ways to describe the multiplexor : the argument `arg` can be a list or a dictionnary.

Note that it is possible to change the generator instanciated with the `SetMux` method.

7.2.4 Parameters

- List :
  For each value of the command, the corresponding net is specified. All values must be specified.
  For example :

  ```
  out <= cmd.Mux ( [in0, in1, in2, in3] )
  ```

  The net out is then initialised like this :

  ```
  if cmd == 0 : out <= in0
  if cmd == 1 : out <= in1
  if cmd == 2 : out <= in2
  if cmd == 3 : out <= in3
  ```

- Dictionnary :
  A dictionnary makes the correspondance between a value of the command and the corresponding net.
  For example :

  ```
  out <= cmd.Mux ( {"0" : in0, "1" : in1, "2" : in2, "3" : in3} )
  ```

This initialisation corresponds to the one before. Thanks to the use of a dictionnary, the connections can be clearer :

- `'default'`: This key of the dictionnary corresponds to all the nets that are not specified
  For example :

  ```
  out <= cmd.Mux ( {"0" : in0, "default" : in1} )
  ```

  This notation corresponds to :

  ```
  if cmd == 0 : out <= in0
  else        : out <= in1
  ```

  Note that if there is no `'default'` key specified and that not all the nets are specified, the non specified nets are set to 0.
- `#` and `?` : When a key of the dictionnary begins with `#`, the number after the `#` has to be binary and each ? in the number means that this bit is not precised
  For example :

  ```
  out <= cmd.Mux ( {"#01?" : in0, "default" : in1} )
  ```

  This notation corresponds to :

  ```
  if cmd in ( 2, 3 ) : out <= in0
  else               : out <= in1
  ```

- `,` and `-` : When keys contains thoses symbols, it permits to enumerate intervals
  For example :

  ```
  out <= cmd.Mux ( {"0,4" : in0, "1-3,5" : in1} )
  ```

  This notation corresponds to :

  ```
  if   cmd in ( 0, 4 )       : out <= in0
  elif cmd in ( 1, 2, 3, 5) : out <= in1
  else                       : out <= 0
  ```

7.2.5 Example

```
class essai ( Model ) :
```

```
def Interface ( self ) :
  self.A    = SignalIn  (    "a", 4 )
  self.B    = SignalIn  (    "b", 4 )
  self.C    = SignalIn  (    "c", 4 )
  self.D    = SignalIn  (    "d", 4 )

  self.Cmd1 = SignalIn  ( "cmd1", 2 )
  self.Cmd2 = SignalIn  ( "cmd2", 4 )

  self.S1   = SignalOut (   "s1", 4 )
  self.S2   = SignalOut (   "s2", 4 )

  self.Vdd = VddIn  ( "vdd" )
  self.Vss = VssIn  ( "vss" )

def Netlist ( self ) :

  self.S1 <= self.Cmd1.Mux ( [sefl.A, self.B, self.C, self.D] )

  self.S2 <= self.Cmd2.Mux ( { "0"       : self.A
                             , "1,5-7"   : self.B
                             , "#1?1?"   : self.C
                             , "default" : self.D
                           } )
```

7.2.6 Errors

Some errors may occur :

- `[Stratus ERROR] Mux : all the nets must have the same lenght.`
  All the input nets pust have the same lenght.
- `[Stratus ERROR] Mux : there are no input nets.`
  The input nets seem to have been forgotten.
- `[Stratus ERROR] Mux : wrong argument type.`
  The connections of the buses are not described by a list nor a dictionnary.
- `[Stratus ERROR] Mux :`
  `the number of nets does not match with the lenght of the command.`
  When using a list, the number of nets has to correspond to the number of possible values of the command.
- `[Stratus ERROR] Mux : wrong key.`
  One of the key of the dictionnary is not un number, neither a list or an interval.

- `[Stratus ERROR] Mux :`
  `when an interval is specified, the second number of the interval must be greater than the first one.`
  When creating an interval with "-", the second number has to be greater than the first one.

- `[Stratus ERROR] Mux :`
  `the binary number does not match with the lenght of the command.`
  When using the `#` notation, each digit of the binary number corresponds to a wire of the cmd. The leghts have to correspond.

- `[Stratus ERROR] Mux : after #, the number has to be binary.`
  When using the `#` notation, the number has to be binary : one can use 0, 1 or ?.

7.3 Shifter

7.3.1 Name

Shift – Easy way to instantiate a shifter

7.3.2 Synopsys

```
netOut <= netCmd.Shift ( netIn, direction, type )
```

7.3.3 Description

This method is a method of net. The net which this method is applied to is the command of the shifter, it's the one which defines the number of bits to shift. The net given as parameter is the input net. The other arguments set the different patameters. The method returns a net : the output net.

Note that it is possible to change the generator instanciated with the `SetShift` method.

7.3.4 Parameters

- `netIn` : the net which is going to be shifted
- `direction` : this string represents the direction of the shift :
  - "left"
  - "right"
- `type` : this string represents the type of the shift :
  - "logical" : only "zeros" are put in the net

– "arith" : meaningful for "right" shift, the values put in the nets are an extension of the MSB
– "circular" : the values put in the nets are the ones which have just been taken off

7.3.5 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.A = SignalIn ( "a", 4 )

    self.Cmd = SignalIn ( "cmd", 2 )

    self.S1 = SignalOut ( "s1", 4 )
    self.S2 = SignalOut ( "s2", 4 )
    self.S3 = SignalOut ( "s3", 4 )

    self.Vdd = VddIn  ( "vdd" )
    self.Vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.S1 <= self.Cmd.Shift ( self.A, "right", "logical" )
    self.S2 <= self.Cmd.Shift ( self.A, "right", "arith"   )

    self.S3 <= self.Cmd.Shift ( self.A, "left", "circular" )
```

If the value of "a" is "0b1001" and the value of "cmd" is "0b10", we will have :

- "s1" : "0b0010"
- "s2" : "0b1110"
- "s3" : "0b0110"

7.3.6 Errors

Some errors may occur :

- `[Stratus ERROR] Shift :`
  `The input net does not have a positive arity.`
  The net which is going to be shifted must have a positive arity.
- `[Stratus ERROR] Shift :`
  `The direction parameter must be "left" or "right".`
  The "direction" argument is not correct.

- `[Stratus ERROR] Shift :`
  `The type parameter must be "logical" or "arith" or "circular".`
  The "type" argument is not correct.

7.4 Register

7.4.1 Name

Reg – Easy way to instantiate a register

7.4.2 Synopsys

```
netOut <= netCk.Reg ( netIn )
```

7.4.3 Description

This method is a method of net. The net which this method is applied to is the clock of the register. The net given as parameter is the input net. The method returns a net : the output net.

Note that it is possible to change the generator instanciated with the `SetReg` method.

7.4.4 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.A  = SignalIn  (  "a", 4 )
    self.S  = SignalOut (  "s", 4 )

    self.Ck = CkIn ( "ck" )

    self.Vdd = VddIn  ( "vdd" )
    self.Vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.S <= self.Ck.Reg ( self.A )
```

7.4.5 Errors

Some errors may occur :

- `[Stratus ERROR] Reg : The input net does not have a positive arity.`
  The input net must have a positive arity.

- `[Stratus ERROR] Reg : The clock does not have a positive arity.` The clock must have a positive arity.

7.5 Constants

7.5.1 Name

Constant – Easy way to instantiate constants

7.5.2 Synopsys

```
netOne  <=  One ( 2 )

net8    <= "8"
```

7.5.3 Description

These functions simplify the way to instanciate constants.

- The functions `One` and`Zero` permits to initialise all the bits of a net to 'one' or 'zero'.
- The instanciation of a constant thanks to a string can be done in decimal, hecadecimal or binary.

7.5.4 Parameters

- For `One` and `Zero` :
  - `n` : the arity of the net
- For the instanciation of a constant :
  - the constant given must be a string representing :
    * A decimal number
    * A binary number : the string must begin with "0b"
    * An hexadecimal number : the string must begin with "0x"

7.5.5 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.Ones   = SignalOut (   "ones", 2 )
    self.Zeros  = SignalOut (  "zeros", 4 )
```

```
    self.Eight  = SignalOut (  "eight", 4 )
    self.Twentu = SignalOut ( "twenty", 5 )
    self.Two    = SignalOut (    "two", 5 )

    self.Vdd = VddIn  ( "vdd" )
    self.Vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.Ones  <=  One ( 2 )
    self.Zero  <= Zero ( 4 )

    self.Eight   <= "8"
    self.Twenty  <= "0x14"
    self.Two     <= "0b10"
```

7.5.6 Errors

Some errors may occur :

- `[Stratus ERROR] Const :`
  `the argument must be a string representing a number in decimal, binary (0b) or hexa (0x).`
  The string given as argument does not have the right form.

7.6 Boolean operations

7.6.1 Description

Most common boolean operators can be instantiated without the `Inst` constructor.

7.6.2 List

Boolean operators are listed below :

- `And2` : `q <= i0 & i1`
- `Or2` : `q <= i0 | i1`
- `Xor2` : `q <= i0 ^ i1`
- `Inv` : `q <= ~i0`

7.6.3 Generators to instantiate

One can choose the generator to be used. Some methods are applied to the cell and set the generator used when using `&`, `|`, `^` and `~`. The generators used by default are the ones from the virtual library.

Methods are :

- `SetAnd`
- `SetOr`
- `SetXor`
- `SetNot`

7.6.4 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.A = SignalIn  ( "a", 4 )
    self.B = SignalIn  ( "b", 4 )
    self.B = SignalIn  ( "c", 4 )

    self.S = SignalOut ( "s", 4 )

    self.vdd = VddIn  ( "vdd" )
    self.vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.S <= ( ~self.A & self.B ) | self.C
```

7.6.5 Errors

Some errors may occur :

- `[Stratus ERROR] & : the nets must have the same lenght.`
  When one uses boolean expressions, one has to check that the sizes of both nets are equivalent.
- `[Stratus ERROR] : there is no alim.`
  The cell being created does not have the alimentation nets. The instanciation is impossible.

7.7 Arithmetical operations

7.7.1 Description

Most common arithmetic operators can be instantiated without the `Inst` constructor.

7.7.2 List

Arithmetical operators are listed below :

- `Addition` : `q <= i0 + i1`
- `Substraction` : `q <= i0 - i1`
- `Multiplication` : `q <= i0 * i1`
- `Division` : `q <= i0 / i1`

7.7.3 Generators to instantiate

One can choose the generator to be used. Some methods are applied to the cell and set the generator used when using overloard. Methods are :

- `SetAdd` (for addition and substraction)
- `SetMult`
- `SetDiv`

The generators used by default are :

- `Addition` : Slansky adder
- `Substraction` : Slansky adder + inversor + cin = '1'
- `Multiplication` : CA2 multiplier (signed, modified booth/Wallace tree)
- `Division` : not available yet

7.7.4 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.A = SignalIn  ( "a", 4 )
    self.B = SignalIn  ( "b", 4 )

    self.S = SignalOut ( "s", 4 )
```

```
    self.T = SignalOut ( "t", 8 )

    self.vdd = VddIn  ( "vdd" )
    self.vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.S <= self.A + self.B

    self.T <= self.A * self.B
```

7.7.5 Errors

Some errors may occur :

- `[Stratus ERROR] + : the nets must have the same lenght.`
  When one uses arithmetic expressions, one has to check that the sizes of both nets are equivalent.
- `[Stratus ERROR] : there is no alim.`
  The cell being created does not have the alimentation nets. The instanciation is impossible.

7.8 Comparison operations

7.8.1 Name

Eq/Ne : Easy way to test the value of the nets

7.8.2 Synopsys

```
netOut <= net.Eq ( "n" )
```

7.8.3 Description

Comparaison functions are listed below :

- `Eq` : returns `true` if the value of the net is equal to `n`.
- `Ne` : returns `true` if the value of the net is different from `n`.

Note that it is possible to change the generator instanciated with the `SetComp` method.

7.8.4 Parameters

The constant given as argument must be a string representing :

- A decimal number
- A binary number : the string must begin with "0b"
- An hexadecimal number : the string must begin with "0x"

7.8.5 Example

```
class essai ( Model ) :

  def Interface ( self ) :
    self.A = SignalIn  ( "a", 4 )

    self.S = SignalOut ( "s", 1 )
    self.T = SignalOut ( "t", 1 )

    self.vdd = VddIn  ( "vdd" )
    self.vss = VssIn  ( "vss" )

  def Netlist ( self ) :

    self.S <= self.A.Eq ( "4" )

    self.T <= self.A.Ne ( "1" )
```

7.8.6 Errors

Some errors may occur :

- `[Stratus ERROR] Eq :`
  `the number does not match with the net's lenght.`
  When one uses comparaison functions on one net, one has to check that the number corresponds to the size of the net.
- `[Stratus ERROR] Eq :`
  `the argument must be a string representing a number in decimal, binary (0b) or hexa (0x).`
  The string given as argument does not have the right form.

8 Virtual library

8.0.1 Description

The virtual library permits to create a cell and map it to different libraries without having to change it.

8.0.2 List of the generators provided

- `a2 : q <= i0 & i1`
- `a3 : q <= i0 & i1 & i2`
- `a4 : q <= i0 & i1 & i2 & i3`
- `na2 : nq <= ~ ( i0 & i1 )`
- `na3 : nq <= ~ ( i0 & i1 & i2 )`
- `na4 : nq <= ~ ( i0 & i1 & i2 & i3 )`
- `o2 : q <= i0 & i1`
- `o3 : q <= i0 & i1 & i2`
- `o4 : q <= i0 & i1 & i2 & i3`
- `no2 : nq <= ~ ( i0 & i1 )`
- `no3 : nq <= ~ ( i0 & i1 & i2 )`
- `no4 : nq <= ~ ( i0 & i1 & i2 & i3 )`
- `inv : nq <= ~ i`
- `buf : q <= i`
- `xr2 : q <= i0 ^ i1`
- `nxr2 : nq <= ~ ( i0 ^ i1 )`
- `zero : nq <= '0'`
- `one : q <= '1'`
- `halfadder : sout <= a ^ b` and `cout <= a & b`
- `fulladder : sout <= a ^ b ^ cin`
  and `cout <= ( a & b ) | ( a & cin ) | ( b & cin )`
- `mx2 : q <= (i0 & ~cmd) | (i1 & cmd)`

- `nmx2:nq <= ~( (i0 & ~cmd) | (i1 & cmd) )`
- `sff:if RISE ( ck ) : q <= i`
- `sff2:if RISE ( ck ) : q <= (i0 & ~cmd) | (i1 & cmd)`
- `sff3:if RISE ( ck ) :`
  `q <= (i0 & ~cmd0) | (((i1 & cmd1)|(i2&~cmd1)) & cmd0)`
- `ts:if cmd : q <= i`
- `nts:if cmd : nq <= ~i`

8.0.3 Mapping file

The virtual library is mapped to the sxlib library. A piece of the corresponding mapping file is shown below.

In order to map the virtual library to another library, on has to write a .xml file which makes correspond models and interfaces.

Note that the interfaces of the cells must be the same (except for the names of the ports). Otherwise, one has to create .vst file in order to make the interfaces match.

The environment variable used to point the right file is `STRATUS_MAPPING_NAME`.

```
<?xml version="1.0" encoding='us-ascii'?>

<technology name="sxlib">
 <model name="a2"   realcell="a2_x2"   i0="i0" i1="i1"  q="q"  vdd="vdd" vss="vss"></model>
 <model name="na2"  realcell="na2_x1"  i0="i0" i1="i1" nq="nq" vdd="vdd" vss="vss"></model>
 <model name="o2"   realcell="o2_x2"   i0="i0" i1="i1"  q="q"  vdd="vdd" vss="vss"></model>
 <model name="no2"  realcell="no2_x1"  i0="i0" i1="i1" nq="nq" vdd="vdd" vss="vss"></model>
 <model name="xr2"  realcell="xr2_x1"  i0="i0" i1="i1"  q="q"  vdd="vdd" vss="vss"></model>
 <model name="nxr2" realcell="nxr2_x1" i0="i0" i1="i1" nq="nq" vdd="vdd" vss="vss"></model>
 <model name="inv"  realcell="inv_x1"   i="i"          nq="nq" vdd="vdd" vss="vss"></model>
 <model name="buf"  realcell="buf_x2"   i="i"           q="q"  vdd="vdd" vss="vss"></model>
</technology>
```

8.0.4 Generators

Some generators are also provided in order to use the cells of the library with nets of more than 1 bit. One has to upper the first letter of the model name in order to user those generators. What is simply done is a for loop with the bits of the nets. The parameter `'nbit'` gives the size of the generator.

8.0.5 Example

- Direct instanciation of a cell

```
for i in range ( 4 ) :
  Inst ( 'a2'
       , map = { 'i0'  : neti0[i]
               , 'i1'  : neti1[i]
               , 'q'   : netq[i]
               , 'vdd' : netvdd
               , 'vss' : netvss
               }
       )
```

- Instanciation of a generator

```
Generate ( 'A2', "my_and2_4bits", param = { 'nbit' : 4 } )
Inst ( 'my_and2_4bits'
     , map  = { 'i0'  : neti0
              , 'i1'  : neti1
              , 'q'   : netq
              , 'vdd' : vdd
              , 'vss' : vss
              }
     )
```

8.0.6 Errors

Some errors may occur :

- `[Stratus ERROR] Inst : the model ... does not exist.`
  `Check CRL_CATA_LIB.`
  The model of the cell has not been found. One has to check the environment variable.
- `[Stratus ERROR] Virtual library : No file found in order to parse.`
  `Check STRATUS_MAPPING_NAME.`
  The mapping file is not given in the environment variable.