0pt0.6pt

Dpgen generators User's Manual

Sophie Belloeil

1 DpgenInv

- **Name** : DpgenInv – Inverter Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenInv', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits inverter with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 1, 2, 4 or 8
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout

- **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= not ( i0 )
```

- **Example** :

```
from stratus import *

class inst_inv ( Model ) :

  def Interface ( self ) :
    self.i = SignalIn  ( "i", 54 )
    self.o = SignalOut ( "o", 54 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenInv', 'inv_54'
             , param = { 'nbit'     : 54
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'inv_54', 'inst'
                  , map = { 'i0'  : self.i
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

2 DpgenBuff

- **Name** : DpgenBuff – Buffer Macro-Generator

- **Synopsys** :

```
Generate ( 'DpgenBuff', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits inverter with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 2, 4 or 8
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= i0
```

- **Example** :

```
from stratus import *

class inst_buff ( Model ) :

  def Interface ( self ) :
    self.i = SignalIn  ( "i", 32 )
    self.o = SignalOut ( "o", 32 )
```

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

  def Netlist ( self ) :
    Generate ( 'DpgenBuff', 'buff_32'
             , param = { 'nbit'     : 32
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'buff_32', 'inst'
                  , map = { 'i0'  : self.i
                          , 'q'   : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

3 DpgenNand2

- **Name** : DpgenNand2 – Nand2 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNand2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs NAND with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)

- **i1** : input (`n` bits)
- **nq** : output (`n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    * Valid drive are : 1 or 4
    * If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= not ( i0 and i1 )
```

- **Example** :

```
from stratus import *

class inst_nand2 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 8 )
    self.in2 = SignalIn  ( "in2", 8 )
    self.o   = SignalOut (   "o", 8 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNand2', 'nand2_8'
             , param = { 'nbit'     : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nand2_8', 'inst'
                  , map = { 'i0'  : self.in1
```

```
                              , 'i1'  : self.in2
                              , 'nq'  : self.o
                              , 'vdd' : self.vdd
                              , 'vss' : self.vss
                              }
                      )

   def Layout ( self ) :
     Place ( self.I, NOSYM, Ref(0, 0) )
```

4 DpgenNand3

- **Name** : DpgenNand3 – Nand3 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNand3', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits three inputs NAND with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates

    * Valid drive are : 1 or 4
    * If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= not ( i0 and i1 and i2 )
```

- **Example** :

```
from stratus import *

class inst_nand3 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 20 )
    self.in2 = SignalIn  ( "in2", 20 )
    self.in3 = SignalIn  ( "in3", 20 )
    self.o   = SignalOut (   "o", 20 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNand3', 'nand3_20'
             , param = { 'nbit'     : 20
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nand3_20', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'i2'  : self.in3
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

5 Dpgennand4

- **Name** : DpgenNand4 – Nand4 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNand4', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits four inputs NAND with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **i3** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    * Valid drive are : 1 or 4
    * If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= not ( i0 and i1 and i2 and i3 )
```

- **Example** :

```
from stratus import *

class inst_nand4 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 9 )
    self.in2 = SignalIn  ( "in2", 9 )
    self.in3 = SignalIn  ( "in3", 9 )
    self.in4 = SignalIn  ( "in4", 9 )
    self.o   = SignalOut (   "o", 9 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNand4', 'nand4_9'
             , param = { 'nbit'     : 9
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nand4_9', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'i2'  : self.in3
                          , 'i3'  : self.in4
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

6 DpgenAnd2

- **Name** : DpgenAnd2 – And2 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenAnd2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs AND with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 2 or 4
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= i0 and i1
```

- **Example** :

```
from stratus import *

class inst_and2 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 8 )
```

```
    self.in2 = SignalIn  ( "in2", 8 )
    self.out = SignalOut (   "o", 8 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenAnd2', 'and2_8'
             , param = { 'nbit'     : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'and2_8', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'q'   : self.out
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

7 DpgenAnd3

- **Name** : DpgenAnd3 – And3 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenAnd3', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits three inputs AND with an output power of `d` named `modelname`.

- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional): Defines the output power of the gates
    - Valid drive are : 2 or 4
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False): In order to generate a layout
  - **behavioral** (optional, default value : False): In order to generate a behavior
- **Behavior** :

```
nq <= i0 and i1 and i2
```

- **Example** :

```
from stratus import *

class inst_and3 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 16 )
    self.in2 = SignalIn  ( "in2", 16 )
    self.in3 = SignalIn  ( "in3", 16 )
    self.out = SignalOut (    "o", 16 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenAnd3', "and3_16"
```

```
            , param = { 'nbit'     : 16
                      , 'physical' : True
                      }
            )
    self.I = Inst ( 'and3_16', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'i2'  : self.in3
                          , 'q'   : self.out
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref (0, 0) )
```

8 DpgenAnd4

- **Name** : DpgenAnd4 – And4 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenAnd4', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits four inputs AND with an output power of `d` named `modelname`.
- Terminal Names :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **i3** : input (`n` bits)

- **q** : output (`n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 2 or 4
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= i0 and i1 and i2 and i3
```

- **Example** :

```
from stratus import *

class inst_and4 ( Model ) :

  def Interface ( self ) :
    self.in1   = SignalIn  ( "in1", 2 )
    self.in2   = SignalIn  ( "in2", 2 )
    self.in3   = SignalIn  ( "in3", 2 )
    self.in4   = SignalIn  ( "in4", 2 )
    self.out   = SignalOut (   "o", 2 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenAnd4', 'and4_2'
             , param = { 'nbit'     : 2
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'and4_2', 'inst'
```

```
                    , map = { 'i0'  : self.in1
                            , 'i1'  : self.in2
                            , 'i2'  : self.in3
                            , 'i3'  : self.in4
                            , 'q'   : self.out
                            , 'vdd' : self.vdd
                            , 'vss' : self.vss
                            }
                    )

    def Layout ( self ) :
      Place ( self.I, NOSYM, Ref(0, 0) )
```

9 DpgenNor2

- **Name** : DpgenNor2 – Nor2 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNor2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs NOR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator

- **drive** (optional) : Defines the output power of the gates
  - Valid drive are : 1 or 4
  - If this parameter is not defined, it's value is the smallest one permitted
- **physical** (optional, default value : False) : In order to generate a layout
- **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= not ( i0 or i1 )
```

- **Example** :

```
from stratus import *

class inst_nor2 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 8 )
    self.in2 = SignalIn  ( "in2", 8 )
    self.o   = SignalOut (   "o", 8 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNor2', 'nor2_8'
             , param = { 'nbit'     : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nor2_8', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

10 DpgenNor3

- **Name** : DpgenNor3 – Nor3 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNor3', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits three inputs NOR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 1 or 4
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= not ( i0 or i1 or i2 )
```

- **Example** :

```
from stratus import *

class inst_nor3 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 3 )
    self.in2 = SignalIn  ( "in2", 3 )
    self.in3 = SignalIn  ( "in3", 3 )
    self.o   = SignalOut ( "out", 3 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNor3', 'nor3_3'
             , param = { 'nbit'     : 3
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nor3_3', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'i2'  : self.in3
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

11 DpgenNor4

- **Name** : DpgenNor4 – Nor4 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNor4', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
```

```
                       , 'physical'   : True
                       , 'behavioral' : True
                       }
             )
```

- **Description** : Generates a `n` bits four inputs NOR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **i3** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 1 or 4
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= not ( i0 or i1 or i2 or i3 )
```

- **Example** :

```
from stratus import *

class inst_nor4 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 15 )
```

```
    self.in2 = SignalIn  ( "in2", 15 )
    self.in3 = SignalIn  ( "in3", 15 )
    self.in4 = SignalIn  ( "in4", 15 )
    self.out = SignalOut (   "o", 15 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNor4', 'nor4_15'
             , param = { 'nbit'     : 15
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nor4_15', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'i2'  : self.in3
                          , 'i3'  : self.in4
                          , 'nq'  : self.out
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )


  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

12 DpgenOr2

- **Name** : DpgenOr2 – Or2 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenOr2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
```

```
)
```

- **Description** : Generates a `n` bits two inputs OR with an output power of `drive` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the a map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    * Valid drive are : 2 or 4
    * If this parameter is not defined, the `drive` is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= i0 or i1
```

- **Example** :

```
from stratus import *

class inst_or2 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 8 )
    self.in2 = SignalIn  ( "in2", 8 )
    self.o   = SignalOut (   "o", 8 )

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

```
def Netlist ( self ) :
  Generate ( 'DpgenOr2', 'or2_8'
           , param = { 'nbit'     : 8
                     , 'physical' : True
                     }
           )
  self.I = Inst ( 'or2_8', 'inst'
                , map = { 'i0'  : self.in1
                        , 'i1'  : self.in2
                        , 'q'   : self.o
                        , 'vdd' : self.vdd
                        , 'vss' : self.vss
                        }
                )


def Layout ( self ) :
  Place ( self.I, NOSYM, Ref(0, 0) )
```

13 DpgenOr3

- **Name** : DpgenOr3 – Or3 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenOr3', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits three inputs OR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)

- **q** : output (`n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    * Valid drive are : 2 or 4
    * If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= i0 or i1 or i2
```

- **Example** :

```
from stratus import *

class inst_or3 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 5 )
    self.in2 = SignalIn  ( "in2", 5 )
    self.in3 = SignalIn  ( "in3", 5 )
    self.o   = SignalOut (   "o", 5 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenOr3', 'or3_5'
             , param = { 'nbit'     : 5
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'or3_5', 'inst'
                  , map = { 'i0'  : self.in1
```

```
                        , 'i1'  : self.in2
                        , 'i2'  : self.in3
                        , 'q'   : self.o
                        , 'vdd' : self.vdd
                        , 'vss' : self.vss
                        }
                )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

14 DpgenOr4

- **Name** : DpgenOr4 – Or4 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenOr4', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits four inputs OR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **i2** : input (`n` bits)
  - **i3** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.

- **nbit** (mandatory) : Defines the size of the generator
- **drive** (optional) : Defines the output power of the gates
  * Valid drive are : 2 or 4
  * If this parameter is not defined, it's value is the smallest one permitted
- **physical** (optional, default value : False) : In order to generate a layout
- **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= i0 or i1 or i2 or i3
```

- **Example** :

```
from stratus import *

class inst_or4 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 16 )
    self.in2 = SignalIn  ( "in2", 16 )
    self.in3 = SignalIn  ( "in3", 16 )
    self.in4 = SignalIn  ( "in4", 16 )
    self.out = SignalOut (   "o", 16 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenOr4', 'or4_16'
             , param = { 'nbit'     : 16
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'or4_16', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'i2'  : self.in3
                          , 'i3'  : self.in4
                          , 'q'   : self.out
                          , 'vdd' : self.vdd
```

```
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

15 DpgenXor2

- **Name** : DpgenXor2 – Xor2 Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenXor2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs XOR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    - Valid drive are : 2 or 4
    - If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optionnal, default value : False) : In order to generate a layout

- **behavioral** (optionnal, default value : False) : In order to generate a behavior

- **Behavior** :

```
nq <= i0 xor i1
```

- **Example** :

```
from stratus import *

class inst_xor2 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 8 )
    self.in2 = SignalIn  ( "in2", 8 )
    self.o   = SignalOut (   "o", 8 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenXor2', 'xor2_8'
             , param = { 'nbit' : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'xor2_8', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'q'   : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

16 DpgenXnor2

- **Name** : DpgenXnor2 – Xnor2 Macro-Generator

- **Synopsys** :

```
Generate ( 'DpgenXnor2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs XNOR with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    * Valid drive are : 1 or 4
    * If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= not ( i0 xor i1 )
```

- **Example** :

```
from stratus import *

class inst_xnor2 ( Model ) :
```

```
def Interface ( self ) :
  self.in1 = SignalIn  ( "in1", 8 )
  self.in2 = SignalIn  ( "in2", 8 )
  self.o   = SignalOut (   "o", 8 )

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

def Netlist ( self ) :
  Generate ( 'DpgenXnor2', 'xnor2_8'
           , param = { 'nbit'     : 8
                     , 'physical' : True
                     }
           )
  self.I = Inst ( 'xnor2_8', 'inst'
                , map = { 'i0'  : self.in1
                        , 'i1'  : self.in2
                        , 'nq'  : self.o
                        , 'vdd' : self.vdd
                        , 'vss' : self.vss
                        }
                )

def Layout ( self ) :
  Place ( self.I, NOSYM, Ref(0, 0) )
```

17 DpgenNmux2

- **Name** : DpgenNmux2 – Multiplexer Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNmux2', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs multiplexer named `modelname`.

- **Terminal Names** :
  - **cmd** : select (1 bit)
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= WITH cmd SELECT not i0 WHEN '0',
                      not i1 WHEN '1';
```

- **Example** :

```
from stratus import *

class inst_nmux2 ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  (  "in1", 5 )
    self.in2 = SignalIn  (  "in2", 5 )
    self.cmd = SignalIn  (  "cmd", 1 )
    self.o   = SignalOut (    "o", 5 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNmux2', 'nmux2_5'
             , param = { 'nbit'     : 5
                       , 'physical' : True
                       }
```

```
            )
      self.I = Inst ( 'nmux2_5', 'inst'
                    , map = { 'i0'  : self.in1
                            , 'i1'  : self.in2
                            , 'cmd' : self.cmd
                            , 'nq'  : self.o
                            , 'vdd' : self.vdd
                            , 'vss' : self.vss
                            }
                    )

    def Layout ( self ) :
      Place ( self.I, NOSYM, Ref(0, 0) )
```

18 DpgenMux2

- **Name** : DpgenMux2 – Multiplexer Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenMux2', modelname
         , param = { 'nbit'       : n
                   , 'drive'      : d
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits two inputs multiplexer with an output power of `d` named `modelname`.
- **Terminal Names** :
  - **cmd** : select (1 bit)
  - **i0** : input (`n` bits)
  - **i1** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **nbit_cmd** (mandatory) : Defines the size of the generator
  - **drive** (optional) : Defines the output power of the gates
    * Valid drive are : 2 or 4
    * If this parameter is not defined, it's value is the smallest one permitted
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nq <= WITH cmd SELECT i0 WHEN '0',
                      i1 WHEN '1';
```

- **Example** :

```
from stratus import *

class inst_mux2 ( Model ) :

  def Interface ( self ) :
    self.in1  = SignalIn  ( "in1", 8 )
    self.in2  = SignalIn  ( "in2", 8 )
    self.cmd  = SignalIn  ( "cmd", 1 )
    self.o    = SignalOut (   "o", 8 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenMux2', 'mux2_8'
             , param = { 'nbit'     : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'mux2_8', 'inst'
                  , map = { 'i0'  : self.in1
                          , 'i1'  : self.in2
                          , 'cmd' : self.cmd
```

```
                         , 'q'   : self.o
                         , 'vdd' : self.vdd
                         , 'vss' : self.vss
                         }
                )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

19 DpgenNbuse

- **Name** : DpgenNbuse – Tristate Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNbuse', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : true
                   , 'behavioral' : true
                   }
         )
```

- **Description** : Generates a `n` bits tristate with an complemented output named `modelname`.
- **Terminal Names** :
  - **cmd** : select (1 bit)
  - **i0** : input (`n` bits)
  - **nq** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nts:BLOCK(cmd = '1') BEGIN
    nq <= GUARDED not(i0);
END
```

- **Example** :

```
from stratus import *

class inst_nbuse ( Model ) :

  def Interface ( self ) :
    self.i   = SignalIn  (   "i", 29 )
    self.cmd = SignalIn  ( "cmd", 1  )
    self.o   = SignalOut (   "o", 29 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenNbuse', 'nbuse29'
             , param = { 'nbit'     : 29
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nbuse29', 'inst'
                  , map = { 'i0'  : self.i
                          , 'cmd' : self.cmd
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

20 DpgenBuse

- **Name** : DpgenBuse – Tristate Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenBuse', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits tristate named `modelname`.
- **Terminal Names** :
  - **cmd** : select (1 bit)
  - **i0** : input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **Behavior** :

```
nts:BLOCK(cmd = '1') BEGIN
    q <= GUARDED i0;
END
```

- **Example** :

```
from stratus import *

class inst_buse ( Model ) :

  def Interface ( self ) :
    self.i   = SignalIn  (   "i", 8 )
    self.cmd = SignalIn  ( "cmd", 1 )
    self.o   = SignalOut (   "o", 8 )

    self.vdd = VddIn ( "vdd" )
```

```
    self.vss = VssIn ( "vss" )

  def Netlist ( self ) :
    Generate ( 'DpgenBuse', 'buse_8'
             , param = { 'nbit'     : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'buse_8', 'inst'
                  , map = { 'i0'  : self.i
                          , 'cmd' : self.cmd
                          , 'q'   : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

21 DpgenNand2mask

- **Name** : DpgenNand2mask – Programmable Mask Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNand2mask', modelname
         , param = { 'nbit'       : n
                   , 'const'      : constVal
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits conditionnal NAND mask named `modelname`.
- **Terminal Names** :
  - **cmd** : mask control (1 bit)
  - **i0** : input (`n` bits)
  - **nq** : output (`n` bits)

- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **const** (mandatory) : Defines the constant (string beginning with 0b, 0x or 0o functions of the basis)
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **How it works** :
  - If the `cmd` signal is set to `zero`, the mask is NOT applied, so the whole operator behaves like an inverter.
  - If the `cmd` signal is set to `one`, the mask is applied, the output is the *complemented* result of the input value *ANDed* with the mask (suplied by `constVal`).
  - The constant `constVal` is given to the macro-generator call, therefore the value cannot be changed afterward : it's hard wired in the operator.
  - A common error is to give a real constant for the `constVal` argument. Be aware that it is a character string.

- **Behavior** :

```
nq <= WITH cmd SELECT not(i0)                WHEN '0',
                      not(i0 and constVal) WHEN '1';
```

- **Example** :

```
from stratus import *

class inst_nand2mask ( Model ) :

  def Interface ( self ) :
    self.i   = SignalIn  (   "i", 32 )
    self.cmd = SignalIn  ( "cmd",  1 )
    self.o   = SignalOut (   "o", 32 )

    self.vdd = VddIn ( "vdd" )
```

```
    self.vss = VssIn ( "vss" )

  def Netlist ( self ) :
    Generate ( 'DpgenNand2mask', 'nand2mask_0x0000ffff'
             , param = { 'nbit'     : 32
                       , 'const'    : "0x0000FFFF"
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nand2mask_0x0000ffff', 'inst'
                  , map = { 'i0'  : self.i
                          , 'cmd' : self.cmd
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

22 DpgenNor2mask

- **Name** : DpgenNor2mask – Programmable Mask Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNor2mask', modelname
         , param = { 'nbit'       : n
                   , 'const'      : constVal
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits conditionnal NOR mask named `modelname`.
- **Terminal Names** :
  - **cmd** : mask control (1 bit)
  - **i0** : input (`n` bits)

- **nq** : output (`n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **const** (mandatory) : Defines the constant (string beginning with 0b, 0x or 0o functions of the basis)
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **How it works** :
  - If the `cmd` signal is set to `zero`, the mask is NOT applied, so the whole operator behaves like an inverter.
  - If the `cmd` signal is set to `one`, the mask is applied, the output is the *complemented* result of the input value *ORed* with the mask (suplied by `constVal`).
  - The constant `constVal` is given to the macro-generator call, therefore the value cannot be changed afterward : it's hard wired in the operator.
  - A common error is to give a real constant for the `constVal` argument. Be aware that it is a character string.

- **Behavior** :

```
nq <= WITH cmd SELECT not(i0)              WHEN '0',
                      not(i0 or constVal) WHEN '1';
```

- **Example** :

```
from stratus import *

class inst_nor2mask ( Model ) :

  def Interface ( self ) :
    self.i   = SignalIn  (   "i", 8 )
    self.cmd = SignalIn  ( "cmd", 1 )
    self.o   = SignalOut (   "o", 8 )

    self.vdd = VddIn ( "vdd" )
```

```
    self.vss = VssIn ( "vss" )

  def Netlist ( self ) :
    Generate ( 'DpgenNor2mask', 'nor2mask_000111'
             , param = { 'nbit'     : 8
                       , 'const'    : "0b000111"
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nor2mask_000111', 'inst'
                  , map = { 'i0'  : self.i
                          , 'cmd' : self.cmd
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

23 DpgenXnor2mask

- **Name** : DpgenXnor2mask – Programmable Mask Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenXnor2mask', modelname
         , param = { 'nbit'       : n
                   , 'const'      : constVal
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits conditionnal XNOR mask named `modelname`.
- **Terminal Names** :
  - **cmd** : mask control (1 bit)
  - **i0** : input (`n` bits)

- **nq** : output (`n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **const** (mandatory) : Defines the constant (string beginning with 0b, 0x or 0o functions of the basis)
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **How it works** :
  - If the `cmd` signal is set to `zero`, the mask is NOT applied, so the whole operator behaves like an inverter.
  - If the `cmd` signal is set to `one`, the mask is applied, the output is the *complemented* result of the input value *XORed* with the mask (suplied by `constVal`).
  - The constant `constVal` is given to the macro-generator call, therefore the value cannot be changed afterward : it's hard wired in the operator.
  - A common error is to give a real constant for the `constVal` argument. Be aware that it is a character string.

- **Behavior** :

```
nq <= WITH cmd SELECT not(i0)                WHEN '0',
                      not(i0 xor constVal) WHEN '1';
```

- **Example** :

```
from stratus import *

class inst_xnor2mask ( Model ) :

  def Interface ( self ) :
    self.i   = SignalIn  (   "i", 8 )
    self.cmd = SignalIn  ( "cmd", 1 )
    self.o   = SignalOut (   "o", 8 )
```

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

  def Netlist ( self ) :
    Generate ( 'DpgenXnor2mask', 'xnor2mask_0b000111'
             , param = { 'nbit'     : 8
                       , 'const'    : "0b000111"
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'xnor2mask_0b000111', 'inst'
                  , map = { 'i0'  : self.i
                          , 'cmd' : self.cmd
                          , 'nq'  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

24 DpgenAdsb2f

- **Name** : DpgenAdsb2f – Adder/Substractor Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenAdsb2f', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits adder/substractor named `modelname`.
- **Terminal Names** :
  - **i0** : First operand (input, `n` bits)
  - **i1** : Second operand (input, `n` bits)

- **q** : Output operand (ouput, `n` bits)
- **add_sub** : Select addition or substraction (input, 1 bit)
- **c31** : Sarry out. In unsigned mode, this is the overflow (output, 1 bit)
- **c30** : Used to compute overflow in signed mode : `overflow = c31 xor c30` (output, 1 bit)
- **vdd** : power
- **vss** : ground

• **Parameters** : Parameters are given in the map `param`.

- **nbit** (mandatory) : Defines the size of the generator
- **physical** (optional, default value : False) : In order to generate a layout
- **behavioral** (optional, default value : False) : In order to generate a behavior

• **How it works** :

- If the `add_sub` signal is set to `zero`, an addition is performed, otherwise it's a substraction.
- Operation can be either signed or unsigned. In unsigned mode `c31` is the overflow ; in signed mode you have to compute overflow by *XORing* `c31` and `c30`

• **Example** :

```
from stratus import *

class inst_ADSB2F ( Model ) :

  def Interface ( self ) :
    self.in1 = SignalIn  ( "in1", 8 )
    self.in2 = SignalIn  ( "in2", 8 )
    self.out = SignalOut (   "o", 8 )
    self.as  = SignalIn  (  "as", 1 )
    self.c0  = SignalOut (  "c0", 1 )
    self.c1  = SignalOut (  "c1", 1 )

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

  def Netlist ( self ) :
```

```
    Generate ( 'DpgenAdsb2f', 'adder_8'
             , param = { 'nbit'     : 8
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'adder_8', 'inst'
                  , map = { 'i0'      : self.in1
                          , 'i1'      : self.in2
                          , 'add_sub' : self.as
                          , 'q'       : self.out
                          , 'c30'     : self.c0
                          , 'c31'     : self.c1
                          , 'vdd'     : self.vdd
                          , 'vss'     : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

25 DpgenShift

- **Name** : DpgenShift – Shifter Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenShift', modelname
         , param = { 'nbit'     : n
                   , 'physical' : True
                   }
         )
```

- **Description** : Generates a `n` bits shifter named `modelname`.
- **Terminal Names** :
  - **op** : select the kind of shift (input, 2 bits)
  - **shamt** : the shift amount (input, `Y` bits)
  - **i** : value to shift (input, `n` bits)
  - **o** : output (`n` bits)

- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout

- **How it works** :
  - If the `op[0]` signal is set to `one`, performs a right shift, performs a left shift otherwise.
  - If the `op[1]` signal is set to `one`, performs an arithmetic shift (only meaningful in case of a right shift).
  - shamt : specifies the shift amount. The width of this signal (`Y`) is computed from the operator's width : `Y = ceil(log2(n))` - 1

- **Example** :

```
from stratus import *

class inst_shifter ( Model ) :

  def Interface ( self ) :
    self.instop    = SignalIn  (    "instop", 2 )
    self.instshamt = SignalIn  ( "instshamt", 2 )
    self.insti     = SignalIn  (     "insti", 4 )
    self.insto     = SignalOut (     "insto", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenShifter', 'shifter_4'
             , param = { 'nbit'     : 4
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'shifter_4', 'inst'
                  , map = { 'op'    : self.instop
                          , 'shamt' : self.instshamt
```

```
                             , 'i'     : self.insti
                             , 'o'     : self.insto
                             , 'vdd'   : self.vdd
                             , 'vss'   : self.vss
                             }
                     )

    def Layout ( self ) :
      Place ( self.I, NOSYM, Ref(0, 0) )
```

26 DpgenShrot

- **Name** : DpgenShrot – Shift/Rotation Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenShrot', modelname
         , param = { 'nbit'     : n
                   , 'physical' : True
                   }
         )
```

- **Description** : Generates a `n` bits shift/rotation operator named `modelname`.
- **Terminal Names** :
  - **op** : select the kind of shift/rotation (input, 3 bits)
  - **shamt** : the shift amount (input, `Y` bits)
  - **i** : value to shift (input, `n` bits)
  - **o** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
- **How it works** :

- If the `op[0]` signal is set to `one`, performs a right shift/rotation , otherwise left shift/rotation occurs.
- If the `op[1]` signal is set to `one`, performs an arithmetic shift (only meaningful in case of a right shift).
- If the `op[2]` signal is set to `one`, performs a rotation, otherwise performs a shift..
- `shamt` specifies the shift amount. The width of this signal (`Y`) is computed from the operator's width : `Y = ceil(log2(n))` - 1

- **Example** :

```
from stratus import *

class inst_shrot ( Model ) :

  def Interface ( self ) :
    self.rotop     = SignalIn  (     "rotop", 3 )
    self.instshamt = SignalIn  ( "instshamt", 2 )
    self.insti     = SignalIn  (     "insti", 4 )
    self.insto     = SignalOut (     "insto", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenShrot', 'shrot_4'
             , param = { 'nbit'     : 4
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'shrot_4', 'inst'
                  , map = { 'op'    : self.rotop
                          , 'shamt' : self.instshamt
                          , 'i'     : self.insti
                          , 'o'     : self.insto
                          , 'vdd'   : self.vdd
                          , 'vss'   : self.vss
                          }
                  )

  def Layout ( self ) :
```

```
    Place ( self.I, NOSYM, Ref(0, 0) )
```

27 DpgenNul

- **Name** : DpgenNul – Zero Detector Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenNul', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   }
         )
```

- **Description** : Generates a `n` bits zero detector named `modelname`.
- **Terminal Names** :
  - **i0** : value to check (input, `n` bits)
  - **q** : null flag (1 bit)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
- **Behavior** :

```
q <= '1' WHEN ( i0 = X"00000000" ) ELSE '0';
```

- **Example** :

```
from stratus import *

class inst_nul ( Model ) :

  def Interface ( self ) :
    self.i = SignalIn  ( "i", 4 )
    self.o = SignalOut ( "o", 1 )
```

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

  def Netlist ( self ) :
    Generate ( 'DpgenNul', 'nul_4'
             , param = { 'nbit'     : 4
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'nul_4', 'inst'
                   , map = { 'i0'  : self.i
                           , 'nul' : self.o
                           , 'vdd' : self.vdd
                           , 'vss' : self.vss
                           }
                   )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

28 DpgenConst

- **Name** : DpgenConst – Constant Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenConst', modelname
         , param = { 'nbit'       : n
                   , 'const'      : constVal
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a `n` bits constant named `modelname`.
- **Terminal Names** :
  - **q** : the constant (output, `n` bit)
  - **vdd** : power

- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **const** (mandatory) : Defines the constant (string beginning with 0b, 0x or 0o functions of the basis)
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior

- **Behavior** :

```
q <= constVal
```

- **Example** :

```
from stratus import *

class inst_const ( Model ) :

  def Interface ( self ) :
    self.o = SignalOut ( "o", 32 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenConst', 'const_0x0000ffff'
             , param = { 'nbit'     : 32
                       , 'const'    : "0x0000FFFF"
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'const_0x0000ffff', 'inst'
                  , map = { 'q'   : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

29 DpgenRom2

- **Name** : DpgenRom2 – 2 words ROM Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenRom2', modelname
         , param = { 'nbit'      : n
                   , 'val0'      : constVal0
                   , 'val1'      : constVal1
                   , 'physical'  : True
                   }
         )
```

- **Description** : Generates a `n` bits 2 words optimized ROM named `modelname`.
- **Terminal Names** :
  - **sel0** : address of the value (input, 1 bit)
  - **q** : the selected word (output, `n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **val0** (mandatory) : Defines the first word
  - **val1** (mandatory) : Defines the second word
  - **physical** (optional, default value : False) : In order to generate a layout
- **Behavior** :

```
q <= WITH sel0 SELECT
     constVal0  WHEN B"0",
     constVal1  WHEN B"1";
```

- **Example** :

```
from stratus import *

class inst_rom2 ( Model ) :
```

```
  def Interface ( self ) :
    self.sel0 = SignalIn  (    "sel0", 1 )
    self.q    = SignalOut ( "dataout", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenRom2', 'rom2_0b1010_0b1100'
             , param = { 'nbit'     : 4
                       , 'val0'     : "0b1010"
                       , 'val1'     : "0b1100"
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'rom2_0b1010_0b1100', 'inst'
                  , map = { 'sel0' : self.sel0
                          , 'q'    : self.q
                          , 'vdd'  : self.vdd
                          , 'vss'  : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

30 DpgenRom4

- **Name** : DpgenRom4 – 4 words ROM Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenRom4', modelname
         , param = { 'nbit'     : n
                   , 'val0'     : constVal0
                   , 'val1'     : constVal1
                   , 'val2'     : constVal2
                   , 'val3'     : constVal3
                   , 'physical' : True
                   }
```

```
)
```

- **Description** : Generates a `n` bits 4 words optimized ROM named `modelname`.
- **Terminal Names** :
  - **sel1** : upper bit of the address of the value (input, 1 bit)
  - **sel0** : lower bit of the address of the value (input, 1 bit)
  - **q** : the selected word (output, `n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **val0** (mandatory) : Defines the first word
  - **val1** (mandatory) : Defines the second word
  - **val2** (mandatory) : Defines the third word
  - **val3** (mandatory) : Defines the fourth word
  - **physical** (optional, default value : False) : In order to generate a layout
- **Behavior** :

```
q <= WITH sel1 & sel0 SELECT constVal0  WHEN B"00",
                             constVal1  WHEN B"01",
                             constVal2  WHEN B"10",
                             constVal3  WHEN B"11";
```

- **Example** :

```
from stratus import *

class inst_rom4 ( Model ) :

  def Interface ( self ) :
    self.sel0 = SignalIn  (    "sel0", 1 )
    self.sel1 = SignalIn  (    "sel1", 1 )
    self.q    = SignalOut ( "dataout", 4 )

    self.vdd = VddIn ( "vdd" )
```

```
    self.vss = VssIn ( "vss" )

  def Netlist ( self ) :
    Generate ( 'DpgenRom4', 'rom4_0b1010_0b1100_0b1111_0b0001'
             , param = { 'nbit'     : 4
                       , 'val0'     : "0b1010"
                       , 'val1'     : "0b1100"
                       , 'val2'     : "0b1111"
                       , 'val3'     : "0b0001"
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'rom4_0b1010_0b1100_0b1111_0b0001', 'inst'
                  , map = { 'sel0' : self.sel0
                          , 'sel1' : self.sel1
                          , 'q'    : self.q
                          , 'vdd'  : self.vdd
                          , 'vss'  : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

31 DpgenRam

- **Name** : DpgenRam – RAM Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenRam', modelname
         , param = { 'nbit'     : n
                   , 'nword'    : regNumber
                   , 'physical' : True
                   }
         )
```

- **Description** : Generates a RAM of `regNumber` words of `n` bits named `modelname`.
- **Terminal Names** :

- **ck** : clock signal (input, 1 bit)
- **w** : write requested (input, 1 bit)
- **selram** : select the write bus (input, 1 bit)
- **ad** : the address (input, `Y` bits)
- **datain** : write bus (input, `n` bits)
- **dataout** : read bus (output, `n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **nword** (mandatory) : Defines the size of the words
  - **physical** (optional, default value : False) : In order to generate a layout
- **Example** :

```
from stratus import *

class inst_ram ( Model ) :

  def Interface ( self ) :
    self.ck      = SignalIn  (      "ck",  1 )
    self.w       = SignalIn  (       "w",  1 )
    self.selram  = SignalIn  (  "selram",  1 )
    self.ad      = SignalIn  (      "ad",  5 )
    self.datain  = SignalIn  (  "datain", 32 )
    self.dataout = TriState ( "dataout", 32 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenRam', 'ram_32_32'
             , param = { 'nbit'     : 32
                       , 'nword'    : 32
                       , 'physical' : True
                       }
             )
```

```
    self.I = Inst ( 'ram_32_32', 'inst'
                  , map = { 'ck'      : self.ck
                          , 'w'       : self.w
                          , 'selram'  : self.selram
                          , 'ad'      : self.ad
                          , 'datain'  : self.datain
                          , 'dataout' : self.dataout
                          , 'vdd'     : self.vdd
                          , 'vss'     : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

32 DpgenRf1

- **Name** : DpgenRf1, DpgenRf1r0 – Register File Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenRf1', modelname
         , param = { 'nbit'     : n
                   , 'nword'    : regNumber
                   , 'physical' : True
                   }
         )
```

- **Description** : Generates a register file of `regNumber` words of `n` bits without decoder named `modelname`.
- **Terminal Names** :
  - **ckok** : clock signal (input, 1 bit)
  - **sel** : select the write bus (input, 1 bit)
  - **selr** : the decoded read address (input, `regNumber` bits)
  - **selw** : the decoded write address (input, `regNumber` bits)
  - **datain0** : first write bus (input, `n` bits)
  - **datain1** : second write bus (input, `n` bits)

- **dataout** : read bus (output, `n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the words (even, between 2 and 64)
  - **nword** (mandatory) : Defines the number of the words (even, between 4 and 32)
  - **physical** (optional, default value : False) : In order to generate a layout

- **How it works** :
  - datain0 and datain1 are the two write busses. Only one is used to actually write the register word, it is selected by the sel signal.
  - When sel is set to zero datain0 is used to write the register word, otherwise it will be datain1
  - selr, selw : this register file have no decoder, so selr have a bus width equal to `regNumber`. One bit for each word
  - The DpgenRf1r0 variant differs from the DpgenRf1 in that the register of address zero is stuck to zero. You can write into it, it will not change the value. When read, it will always return zero

- **Example** :

```
from stratus import *

class inst_rf1 ( Model ) :

  def Interface ( self ) :
    self.ck       = SignalIn    (       "ck",  1 )
    self.sel      = SignalIn    (      "sel",  1 )
    self.selr     = SignalIn    (     "selr", 16 )
    self.selw     = SignalIn    (     "selw", 16 )
    self.datain0  = SignalIn    (  "datain0",  4 )
    self.datain1  = SignalIn    (  "datain1",  4 )
    self.dataout  = SignalOut   (  "dataout",  4 )

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

```
def Netlist ( self ) :
  Generate ( 'DpgenRf1', 'rf1_4_16'
           , param = { 'nbit'     : 4
                     , 'nword'    : 16
                     , 'physical' : True
                     }
           )
  self.I = Inst ( 'rf1_4_16', 'inst'
                , map =  { 'ck'       : self.ck
                         , 'sel'      : self.sel
                         , 'selr'     : self.selr
                         , 'selw'     : self.selw
                         , 'datain0'  : self.datain0
                         , 'datain1'  : self.datain1
                         , 'dataout'  : self.dataout
                         , 'vdd'      : self.vdd
                         , 'vss'      : self.vss
                         }
                )

def Layout ( self ) :
  Place ( self.I, NOSYM, Ref(0, 0) )
```

33 DpgenRf1d

- **Name** : DpgenRf1d, DpgenRf1dr0 – Register File with Decoder Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenRf1d', modelname
         , param = { 'nbit'     : n
                   , 'nword'    : regNumber
                   , 'physical' : True
                   }
         )
```

- **Description** : Generates a register file of `regNumber` words of `n` bits with decoder named `modelname`.
- **Terminal Names** :

- **ck** : clock signal (input, 1 bit)
- **sel** : select the write bus (input, 1 bit)
- **wen** : write enable (input, 1 bit)
- **ren** : read enable (input, 1 bit)
- **adr** : the read address (input, `Y` bits)
- **adw** : the write address (input, `Y` bits)
- **datain0** : first write bus (input, `n` bits)
- **datain1** : second write bus (input, `n` bits)
- **dataout** : read bus (output, `n` bits)
- **vdd** : power
- **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the words (even, between 2 and 64)
  - **nword** (mandatory) : Defines the number of the words (even, between 6 and 32)
  - **physical** (optional, default value : False) : In order to generate a layout

- **How it works** :
  - datain0 and datain1 are the two write busses. Only one is used to actually write the register word, it is selected by the sel signal.
  - When sel is set to zero datain0 is used to write the register word, otherwise it will be datain1
  - adr, adw : the width (Y) of those signals is computed from regNumber : `Y = log2(regNumber)`
  - wen and ren : write enable and read enable, allows reading and writing when sets to `one`
  - The DpgenRf1dr0 variant differs from the DpgenRf1d in that the register of address zero is stuck to zero. You can write into it, it will not change the value. When read, it will always return zero

- **Example** :

```
from stratus import *
```

```
class inst_rf1d ( Model ) :

  def Interface ( self ) :
    self.ck      = SignalIn  (      "ck", 1 )
    self.sel     = SignalIn  (     "sel", 1 )
    self.wen     = SignalIn  (     "wen", 1 )
    self.ren     = SignalIn  (     "ren", 1 )
    self.adr     = SignalIn  (     "adr", 4 )
    self.adw     = SignalIn  (     "adw", 4 )
    self.datain0 = SignalIn  ( "datain0", 4 )
    self.datain1 = SignalIn  ( "datain1", 4 )
    self.dataout = SignalOut ( "dataout", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenRf1d', 'rf1d_4_16'
             , param = { 'nbit'     : 4
                       , 'nword'    : 16
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'rf1d_4_16', 'inst'
                  , map = { 'ck'      : self.ck
                          , 'sel'     : self.sel
                          , 'wen'     : self.wen
                          , 'ren'     : self.ren
                          , 'adr'     : self.adr
                          , 'adw'     : self.adw
                          , 'datain0' : self.datain0
                          , 'datain1' : self.datain1
                          , 'dataout' : self.dataout
                          , 'vdd'     : self.vdd
                          , 'vss'     : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

34 DpgenFifo

- **Name** : DpgenFifo – Fifo Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenFifo', modelname
         , param = { 'nbit'        : n
                   , 'nword'       : regNumber
                   , 'physical'    : True
                   }
         )
```

- **Description** : Generates a FIFO of `regNumber` words of `n` bits named `modelname`.
- **Terminal Names** :
  - **ck** : clock signal (input, 1 bit)
  - **reset** : reset signal (input, 1 bit)
  - **r** : read requested (input, 1 bit)
  - **w** : write requested (input, 1 bit)
  - **rok** : read acknowledge (output, 1 bit)
  - **wok** : write acknowledge (output, 1 bit)
  - **sel** : select the write bus (input, 1 bit)
  - **datain0** : first write bus (input, `n` bits)
  - **datain1** : second write bus (input, `n` bits)
  - **dataout** : read bus (output, `n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the words (even, between 2 and 64)
  - **nword** (mandatory) : Defines the number of words (even, between 4 and 32)
  - **physical** (optional, default value : False) : In order to generate a layout
- **How it works** :

- datain0 and datain1 : the two write busses. Only one is used to actually write the FIFO, it is selected by the sel signal.
- sel : when set to `zero` the datain0 is used to write the register word, otherwise it will be datain1.
- r, rok : set r when a word is requested, rok tells that a word has effectively been popped (rok == not empty).
- w, wok : set w when a word is pushed, wok tells that the word has effectively been pushed (wok == not full).

- **Example** :

```
from stratus import *

class inst_fifo ( Model ) :

  def Interface ( self ) :
    self.ck       = SignalIn    (       "ck", 1 )
    self.reset    = SignalIn    (    "reset", 1 )
    self.r        = SignalIn    (        "r", 1 )
    self.w        = SignalIn    (        "w", 1 )
    self.rok      = SignalInOut (      "rok", 1 )
    self.wok      = SignalInOut (      "wok", 1 )
    self.sel      = SignalIn    (      "sel", 1 )
    self.datain0 = SignalIn     (  "datain0", 4 )
    self.datain1 = SignalIn     (  "datain1", 4 )
    self.dataout = SignalOut    (  "dataout", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenFifo', 'fifo_4_16'
             , param = { 'nbit'     : 4
                       , 'nword'    : 16
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'fifo_4_16', 'inst'
                  , map = { 'ck'      : self.ck
                          , 'reset'   : self.reset
                          , 'r'       : self.r
```

```
                                      , 'w'        : self.w
                                      , 'rok'      : self.rok
                                      , 'wok'      : self.wok
                                      , 'sel'      : self.sel
                                      , 'datain0'  : self.datain0
                                      , 'datain1'  : self.datain1
                                      , 'dataout'  : self.dataout
                                      , 'vdd'      : self.vdd
                                      , 'vss'      : self.vss
                                      }
                              )

      def Layout ( self ) :
        Place ( self.I, NOSYM, Ref(0, 0) )
```

35 DpgenDff

- **Name** : DpgenDff – Dynamic Flip-Flop Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenDff', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a n bits dynamic flip-flop named `modelname`. The two latches of this flip-flop are dynamic, i.e. the data is stored in a capacitor.
- **Terminal Names** :
  - **wen** : write enable (1 bit)
  - **ck** : clock signal (1 bit)
  - **i0** : data input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground

- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **How it works** :
  - When wen is set to `one`, enables the writing of the flip-flop
- **Example** :

```
from stratus import *

class inst_dff ( Model ) :

  def Interface ( self ) :
    self.ck  = SignalIn  (  "ck", 1 )
    self.wen = SignalIn  ( "wen", 1 )
    self.i   = SignalIn  (   "i", 4 )
    self.o   = SignalOut (   "o", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenDff', 'dff_4'
             , param = { 'nbit'     : 4
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'dff_4', 'inst'
                  , map = { "wen" : self.wen
                          , "ck"  : self.ck
                          , "i0"  : self.i
                          ,  "q"  : self.o
                          , 'vdd' : self.vdd
                          , 'vss' : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

36 DpgenDfft

- **Name** : DpgenDfft – Dynamic Flip-Flop with Scan-Path Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenDfft', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a n bits dynamic flip-flop with scan-path named `modelname`. The two latches of this flip-flop are dynamic, i.e. the data is stored in a capacitor.
- **Terminal Names** :
  - **scan** : scan-path mode (input, 1 bit)
  - **scin** : scan path in (input, 1 bit)
  - **wen** : write enable (1 bit)
  - **ck** : clock signal (1 bit)
  - **i0** : data input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **How it works** :
  - When scan is set to `one`, it enables the scan-path mode. Note that in scan-path mode, the wen signal is not effective
  - scin is the input of the scan-path. This terminal is different from `i0[0]`. The scout is q[N-1] (in the following example this is `q[31]`)

- When wen is set to `one` enables the writing of the flip-flop

- **Example** :

```
from stratus import *

class inst_dfft ( Model ) :

  def Interface ( self ) :
    self.scan = SignalIn  ( "scin", 1 )
    self.scin = SignalIn  ( "scan", 1 )
    self.ck   = SignalIn  (   "ck", 1 )
    self.wen  = SignalIn  (  "wen", 1 )
    self.i    = SignalIn  (    "i", 4 )
    self.o    = SignalOut (    "o", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenDfft', 'dfft_4'
             , param = { 'nbit' : 4
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'dfft_4', 'inst'
                  , map = { "wen"  : self.wen
                          , "ck"   : self.ck
                          , "scan" : self.scan
                          , "scin" : self.scin
                          , "i0"   : self.i
                          ,  "q"   : self.o
                          , 'vdd'  : self.vdd
                          , 'vss'  : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```

37 DpgenSff

- **Name** : DpgenSff – Static Flip-Flop Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenSff', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a n bits static flip-flop named `modelname`. The two latches of this flip-flop are static, i.e. each one is made of two interters looped together.
- **Terminal Names** :
  - **wen** : write enable (1 bit)
  - **ck** : clock signal (1 bit)
  - **i0** : data input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **How it works** :
  - When wen is set to `one`, enables the writing of the flip-flop
- **Example** :

```
from stratus import *

class inst_sff ( Model ) :
```

```
def Interface ( self ) :
  self.ck  = SignalIn  (  "ck", 1 )
  self.wen = SignalIn  ( "wen", 1 )
  self.i   = SignalIn  (   "i", 4 )
  self.o   = SignalOut (   "o", 4 )

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

def Netlist ( self ) :
  Generate ( 'DpgenSff', 'sff_4'
           , param = { 'nbit'     : 4
                     , 'physical' : True
                     }
           )
  self.I = Inst ( 'sff_4', 'inst'
                , map = { "wen" : self.wen
                        , "ck"  : self.ck
                        , "i0"  : self.i
                        ,  "q"  : self.o
                        , 'vdd' : self.vdd
                        , 'vss' : self.vss
                        }
                )

def Layout ( self ) :
  Place ( self.I, NOSYM, Ref(0, 0) )
```

38 DpgenSfft

- **Name** : DpgenSfft – Static Flip-Flop with Scan-Path Macro-Generator
- **Synopsys** :

```
Generate ( 'DpgenSfft', modelname
         , param = { 'nbit'       : n
                   , 'physical'   : True
                   , 'behavioral' : True
                   }
         )
```

- **Description** : Generates a n bits static flip-flop with scan-path named `modelname`. The two latches of this flip-flop are static i.e. each one is made of two interters looped togethers.
- **Terminal Names** :
  - **scan** : scan-path mode (input, 1 bit)
  - **scin** : scan path in (input, 1 bit)
  - **wen** : write enable (1 bit)
  - **ck** : clock signal (1 bit)
  - **i0** : data input (`n` bits)
  - **q** : output (`n` bits)
  - **vdd** : power
  - **vss** : ground
- **Parameters** : Parameters are given in the a map `param`.
  - **nbit** (mandatory) : Defines the size of the generator
  - **physical** (optional, default value : False) : In order to generate a layout
  - **behavioral** (optional, default value : False) : In order to generate a behavior
- **How it works** :
  - When scan is set to `one`, it enables the scan-path mode. Note that in scan-path mode, the wen signal is not effective
  - scin : the input of the scan-path. This terminal is different from `i0[0]`. The scout is `q[N-1]` (in the following example this is `q[3]`)
  - When wen is set to `one`, it enables the writing of the flip-flop
- **Example** :

```
from stratus import *

class inst_sfft ( Model ) :

  def Interface ( self ) :
    self.scan = SignalIn  ( "scin", 1 )
    self.scin = SignalIn  ( "scan", 1 )
    self.ck   = SignalIn  (   "ck", 1 )
    self.wen  = SignalIn  (  "wen", 1 )
```

```
    self.i    = SignalIn  (   "in", 4 )
    self.o    = SignalOut (  "out", 4 )

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

  def Netlist ( self ) :
    Generate ( 'DpgenSfft', 'sfft_4'
             , param = { 'nbit'     : 4
                       , 'physical' : True
                       }
             )
    self.I = Inst ( 'sfft_4', 'inst'
                  , map = { "wen"  : self.wen
                          , "ck"   : self.ck
                          , "scan" : self.scan
                          , "scin" : self.scin
                          , "i0"   : self.i
                          ,  "q"   : self.o
                          , 'vdd'  : self.vdd
                          , 'vss'  : self.vss
                          }
                  )

  def Layout ( self ) :
    Place ( self.I, NOSYM, Ref(0, 0) )
```