GenLib User’s Manual

Jean-Paul Chaput
System Administrator
Pierre & Marie Curie University, LIP6
ASIM Department

Frédéric Pétrot
Professor Assistant
Pierre & Marie Curie University, LIP6
ASIM Department

GenLib User's Manual

by Jean-Paul Chaput and Frédéric Pétrot

Published June 2002

Table of Contents

I. *Alliance* - *genlib* User’s Manual

genlib

Name

`genlib` — Procedural design language based upon C.

Description

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

Based upon the *Alliance mbk* data structures, the *genlib* language gives the user the ability to describe both netlist and layout views, thus allowing both standard cell and full custom approachs.

Netlist capture

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

The function calls used to handle the netlist view are :

- GENLIB_DEF_LOFIG(3)
- GENLIB_SAVE_LOFIG(3)
- GENLIB_LOINS(3)
- GENLIB_LOCON(3)
- GENLIB_LOSIG(3)
- GENLIB_FLATTEN_LOFIG(3)

Some facilities, in order to create vectors are also available :

- GENLIB_BUS(3)
- GENLIB_ELM(3)

Standard cell placement

The following functions allows to define a placement file for a standard cell design. This file can be used by the standard cell router ocr(1) :

- GENLIB_DEF_PHSC(3)
- GENLIB_SAVE_PHSC(3)
- GENLIB_SC_PLACE(3)
- GENLIB_SC_RIGHT(3)
- GENLIB_SC_TOP(3)
- GENLIB_SC_LEFT(3)
- GENLIB_SC_BOTTOM(3)

Full custom symbolic layout

Those functions are dedicated to optimized full custom procedural layout. In order to provide some process independance, *Alliance* uses a symbolic layout approach (fixed grid without compaction).

The symbolic objects are segments (wires), vias (contacts), connectors (I/Os), references and instances. For more informations, see phseg(1), phvia(1), phcon(1), phref(1), phins(1) and alc(1).

- GENLIB_DEF_PHFIG(3)
- GENLIB_SAVE_PHFIG(3)
- GENLIB_DEF_AB(3)
- GENLIB_DEF_PHINS(3)
- GENLIB_PHCON(3)
- GENLIB_COPY_UP_CON(3)
- GENLIB_COPY_UP_CON_FACE(3)
- GENLIB_COPY_UP_ALL_CON(3)
- GENLIB_PHSEG(3)
- GENLIB_COPY_UP_SEG(3)
- GENLIB_THRU_H(3)
- GENLIB_THRU_V(3)
- GENLIB_THRU_CON_H(3)
- GENLIB_THRU_CON_V(3)
- GENLIB_WIRE1(3)
- GENLIB_WIRE2(3)
- GENLIB_WIRE3(3)
- GENLIB_PHVIA(3)
- GENLIB_PLACE(3)
- GENLIB_PLACE_RIGHT(3)
- GENLIB_PLACE_TOP(3)
- GENLIB_PLACE_LEFT(3)
- GENLIB_PLACE_BOTTOM(3)
- GENLIB_PLACE_ON(3)
- GENLIB_PHREF(3)
- GENLIB_COPY_UP_REF(3)
- GENLIB_COPY_UP_ALL_REF(3)
- GENLIB_PLACE_VIA_REF(3)
- GENLIB_PLACE_CON_REF(3)
- GENLIB_PLACE_SEG_REF(3)
- GENLIB_FLATTEN_PHFIG(3)
- GENLIB_GET_REF_X(3)
- GENLIB_GET_REF_Y(3)
- GENLIB_GET_CON_X(3)
- GENLIB_GET_CON_Y(3)

- GENLIB_HEIGHT(3)
- GENLIB_WIDTH(3)

In order to have information about each of these functions, use the online documentation with man(1), as in `man function-name`.

It is strongly recommended to read some books on C programming, in order to take full advantage of the C flow control possibilities, as it may greatly reduce the size of a *genlib* source code.

ENVIRONMENT VARIABLES

- MBK_IN_LO(1), default value : `al`
- MBK_OUT_LO(1), default value : `al`
- MBK_IN_PH(1), default value : `ap`
- MBK_OUT_LO(1), default value : `ap`
- MBK_CATA_LIB(1), default value : `.`
- MBK_WORK_LIB(1), default value : `.`
- MBK_CATAL_NAME(1), default value : `CATAL`

See the corresponding manual pages for further informations.

In order to compile and execute a *genlib* file, one has to call *genlib* with one argument, that is the *genlib* source file. The source file must have a .c extension, but the extension should not be mentionned on the command line.

The names used in genlib, as arguments to genlib functions, should be alphanumerical, including the underscore. They also are not case sensitive, so `VDD` is equivalent to `vdd`. Vectorized connectors or signal can be declareds using the `[n:m]` construct.

Synopsis

genlib [`-cklmnv`] [--no-rm-core] [--keep-makefile] [--keep-exec] [--keep-log] [--no-exec] [--verbose] {program} [-e ***`program_args`***]

Options

- <`program`> : the name of the C file containing the *genlib* program, whitout extention. Mandatory argument.
- `[--no-rm-core|-c]` : in case of core dump, do not remove the generated core file. This option must be used with `[--keep-exec|-k]`.
- `[--keep-makefile|-m]` : do not erase the generated makefile after execution.
- `[--keep-exec|-k]` : keep the generated executable after the *genlib* run.
- `[--keep-log|-l]` : do not erase the log file after a successfull completion (the log is keeped after a faulty run).
- `[--no-exec|-n]` : do not run the generated program. Should be used with `[--keep-exec|-k]`.
- `[--no-verbose|-v]` : self explanatory.
- `[-e]` : all the following arguments are handled to the compiled program.

Examples

Compile and run a file `amd2901.c` :

```
genlib -v amd2901
```

See Also

mbk(1),

Diagnostic

Many errors may occur while executing the source file, so refer to the proper genlib function manual for more. When an error occur, *genlib* left a log file <`program`>`.grr`. As <`program`>`.c` is a C program, all syntatic C error can occurs...

All genlib functions are listed below alphabetically sorted.

GENLIB_MACRO

Name

`GENLIB_MACRO` — Interface with all MACRO generators.

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(long function, char *modelname, long flags, long N,
long drive, long regNumber, char *constVal);
```

Description

The *GENLIB_MACRO()* fonction call is the generic interface to all *genlib* macro generators. As all generators do not have the sames arguments it takes a variable number of arguments.

Arguments are of two kinds :

- Mandatory arguments : those arguments must always be supplied. They represents the minimal set of common arguments for all macro-generators. Mandatory arguments comme firts in the arguments list.

- Optional arguments : arguments specifics to a macro-generator or a class of macro-generators.

Arguments

1. `long` *`function`* : (mandatory) specify which macro-generator is to be invoked. For example, to create a two entry multiplexer gives `GENLIB_MUX2`.
2. `char*` *`modelname`* : (mandatory) specify the name of the model to be generated. For example : `"model_mux2_32bits"`.
3. `long` *`flags`* : (mandatory) tells which views are to be generated. It is a logical combination of the following values : (*ORed*)
   a. `F_PLACE` : generate physical view (placement only).
   b. `F_BEHAV` : generate behavioral view (*VHDL* view).
   c. `F_LSB_FIRST` : the terminal of index zero is matched to the LSB, thus a 32 bits vector will be written : `"signal[31:0]"`.
   d. `F_MSB_FIRST` : the terminal of index zero is matched to the MSB, thus a 32 bits vector will be written : `"signal[0:31]"` (aka. IBM notation).
4. `long` *`N`* : (mandatory) width of the model's bus to be generated.
5. `long` *`drive`* : (optional) output power drive.
6. `long` *`regNumber`* : (optional) number of registers or depth of for a FIFO macro-generator.
7. `char*` *`constval`* : (optional) a character string holding a constant. For example : `"0xFFFF0000"`.

Available Macro-Generators

- Inverters and Buffers :
  - DPGEN_INV(3)
  - DPGEN_BUFF(3)
- Boolean Operators :
  - DPGEN_NAND2(3)
  - DPGEN_NAND3(3)
  - DPGEN_NAND4(3)
  - DPGEN_AND2(3)
  - DPGEN_AND3(3)
  - DPGEN_AND4(3)
  - DPGEN_NOR2(3)
  - DPGEN_NOR3(3)
  - DPGEN_NOR4(3)
  - DPGEN_OR2(3)
  - DPGEN_OR3(3)

  - DPGEN_OR4(3)
  - DPGEN_XOR2(3)
  - DPGEN_XNOR2(3)
- Multiplexers and Tristates :
  - DPGEN_NMUX2(3)
  - DPGEN_MUX2(3)
  - DPGEN_NBUSE(3)
  - DPGEN_BUSE(3)
- Programmable Masks :
  - DPGEN_NAND2MASK(3)
  - DPGEN_NOR2MASK(3)
  - DPGEN_XNOR2MASK(3)
- Arithmetics Operators :
  - DPGEN_ADSB2F(3)
  - DPGEN_SHIFT(3)
- Miscellaneous :
  - DPGEN_NUL(3)
  - DPGEN_CONST(3)
  - DPGEN_ROM2(3)
  - DPGEN_ROM4(3)
- Registers :
  - DPGEN_RF1(3)
  - DPGEN_RF1R0(3)
  - DPGEN_FIFO(3)
  - DPGEN_RF1D(3)
  - DPGEN_RF1DR0(3)
  - DPGEN_DFF(3)
  - DPGEN_DFFT(3)
  - DPGEN_SFF(3)
  - DPGEN_SFFT(3)

See Also

mbk(1),

DPGEN_INV

Name

DPGEN_INV — Inverter Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_INV, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *N* bits inverter with an output power of *drive* named *modelname*.

Valid *drive* are : 1, 2, 3 or 4.

Terminal Names

1. i0 : input.
2. nq : output.
3. vdd : power.
4. vss : ground.

Behavior

nq <= not(i0)

Example

```
GENLIB_MACRO(DPGEN_INV, "model_inv_32"
                      , F_BEHAV|F_PLACE
                      , 32
                      , 1
                      );

GENLIB_LOINS( "model_inv_32"
            , "instance1_inv_32"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_BUFF

Name

`DPGEN_BUFF` — Buffer Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_BUFF, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits buffer with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 2, 3 or 4.

Terminal Names

1. `i0` : input.
2. `q` : output.
3. `vdd` : power.
4. `vss` : ground.

Behavior

q <= i0

Example

```
GENLIB_MACRO( DPGEN_BUFF
            , "model_buff_32"
            , F_BEHAV|F_PLACE
            , 32
            , 4
            );

GENLIB_LOINS( "model_buff_32"
            , "instance1_buff_32"
            , "i0[31:0]", "q[31:0]"
```

```
          , "vdd", "vss", NULL
          );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NAND2

Name

`DPGEN_NAND2` — NAND2 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NAND2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits two inputs NAND with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i1` : input.
2. `i0` : input.
3. `nq` : output.
4. `vdd` : power.
5. `vss` : ground.

Behavior

nq <= not(i0 and i1)

Example

```
GENLIB_MACRO(DPGEN_NAND2, "model_nand2_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        , 1
                        );

GENLIB_LOINS( "model_nand2_32"
            , "instance1_nand2_32"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NAND3

Name

`DPGEN_NAND3` — NAND3 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NAND3, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits three inputs NAND with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i2` : input.
2. `i1` : input.
3. `i0` : input.
4. `nq` : output.

5. `vdd` : power.
6. `vss` : ground.

Behavior

nq <= not(i0 and i1 and i2)

Example

```
GENLIB_MACRO(DPGEN_NAND3, "model_nand3_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        , 1
                        );

GENLIB_LOINS( "model_nand3_32"
            , "instance1_nand3_32"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NAND4

Name

`DPGEN_NAND4` — NAND4 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NAND4, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits four inputs NAND with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i3` : input.
2. `i2` : input.
3. `i1` : input.
4. `i0` : input.
5. `nq` : output.
6. `vdd` : power.
7. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_NAND4, "model_nand4_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        , 1
                        );

GENLIB_LOINS( "model_nand4_32"
            , "instance1_nand4_32"
            , "i3[31:0]"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_AND2

Name

`DPGEN_AND2` — AND2 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_AND2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits two inputs AND with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 2 or 4.

Terminal Names

1. `i1` : input.
2. `i0` : input.
3. `q` : output.
4. `vdd` : power.
5. `vss` : ground.

Behavior

q <= i0 and i1

Example

```
GENLIB_MACRO(DPGEN_AND2, "model_and2_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 2
                       );

GENLIB_LOINS( "model_and2_32"
            , "instance1_and2_32"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_AND3

Name

`DPGEN_AND3` — AND3 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_AND3, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits three inputs AND with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 2 or 4.

Terminal Names

1. `i2` : input.
2. `i1` : input.
3. `i0` : input.
4. `q` : output.
5. `vdd` : power.
6. `vss` : ground.

Behavior

q <= not(i0 and i1 and i2)

Example

```
GENLIB_MACRO(DPGEN_AND3, "model_and3_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 2
                       );

GENLIB_LOINS( "model_and3_32"
            , "instance1_and3_32"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_AND4

Name

`DPGEN_AND4` — AND4 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_AND4, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits four inputs AND with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 2 or 4.

Terminal Names

1. `i3` : input.
2. `i2` : input.
3. `i1` : input.
4. `i0` : input.

5. q : output.
6. vdd : power.
7. vss : ground.

Behavior

q <= i0 and i1 and i2 and i3

Example

```
GENLIB_MACRO(DPGEN_AND4, "model_and4_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 2
                       );

GENLIB_LOINS( "model_and4_32"
            , "instance1_and4_32"
            , "i3[31:0]"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NOR2

Name

`DPGEN_NOR2` — NOR2 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NOR2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits two inputs NOR with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i1` : input.
2. `i0` : input.
3. `nq` : output.
4. `vdd` : power.
5. `vss` : ground.

Behavior

nq <= not(i0 or i1)

Example

```
GENLIB_MACRO(DPGEN_NOR2, "model_nor2_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 1
                       );

GENLIB_LOINS( "model_nor2_32"
            , "instance1_nor2_32"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NOR3

Name

`DPGEN_NOR3` — NOR3 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NOR3, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *N* bits three inputs NOR with an output power of *drive* named *modelname*.

Valid *drive* are : 1 or 4.

Terminal Names

1. `i2` : input.
2. `i1` : input.
3. `i0` : input.
4. `nq` : output.
5. `vdd` : power.
6. `vss` : ground.

Behavior

nq <= not(i0 or i1 or i2)

Example

```
GENLIB_MACRO(DPGEN_NOR3, "model_nor3_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 1
                       );

GENLIB_LOINS( "model_nor3_32"
            , "instance1_nor3_32"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NOR4

Name

`DPGEN_NOR4` — NOR4 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NOR4, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits four inputs NOR with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i3` : input.
2. `i2` : input.
3. `i1` : input.
4. `i0` : input.
5. `nq` : output.
6. `vdd` : power.
7. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_NOR4, "model_nor4_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 1
```

```
                         );

GENLIB_LOINS( "model_nor4_32"
            , "instance1_nor4_32"
            , "i3[31:0]"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_OR2

Name

`DPGEN_OR2` — OR2 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_OR2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits two inputs OR with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 2 or 4.

Terminal Names

1. `i1` : input.
2. `i0` : input.
3. `q` : output.
4. `vdd` : power.
5. `vss` : ground.

Behavior

q <= i0 or i1

Example

```
GENLIB_MACRO(DPGEN_OR2, "model_or2_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 2
                       );

GENLIB_LOINS( "model_or2_32"
            , "instance1_or2_32"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_OR3

Name

`DPGEN_OR3` — OR3 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_OR3, char *modelname, long flags, long N,
long drive);
```

Description

Generate a `N` bits three inputs OR with an output power of `drive` named `modelname`.

Valid `drive` are : 2 or 4.

Terminal Names

1. `i2` : input.
2. `i1` : input.
3. `i0` : input.
4. `q` : output.
5. `vdd` : power.
6. `vss` : ground.

Behavior

q <= i0 or i1 or i2

Example

```
GENLIB_MACRO(DPGEN_OR3, "model_or3_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 2
                       );

GENLIB_LOINS( "model_or3_32"
            , "instance1_or3_32"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_OR4

Name

`DPGEN_OR4` — OR4 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_OR4, char *modelname, long flags, long N,
long drive);
```

Description

Generate a `N` bits four inputs OR with an output power of `drive` named `modelname`. Valid `drive` are : 2 or 4.

Terminal Names

1. `i3` : input.
2. `i2` : input.
3. `i1` : input.
4. `i0` : input.
5. `q` : output.
6. `vdd` : power.
7. `vss` : ground.

Behavior

q <= i0 or i1 or i2 or i3

Example

```
GENLIB_MACRO(DPGEN_OR4, "model_or4_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        , 2
                        );

GENLIB_LOINS( "model_or4_32"
            , "instance1_or4_32"
            , "i3[31:0]"
            , "i2[31:0]"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_XNOR2

Name

`DPGEN_XNOR2` — XNOR2 Macro-Generator

Synopsis

```
#include <genlib.h>
void GENLIB_MACRO(DPGEN_XNOR2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits two inputs exclusive NOR with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i1` : input.
2. `i0` : input.
3. `nq` : output.
4. `vdd` : power.
5. `vss` : ground.

Behavior

nq <= not(i0 xor i1)

Example

```
GENLIB_MACRO(DPGEN_XNOR2, "model_xnor2_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        , 1
                        );

GENLIB_LOINS( "model_xnor2_32"
            , "instance1_xnor2_32"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_XOR2

Name

`DPGEN_XOR2` — XOR2 Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_XOR2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *`N`* bits two inputs exclusive OR with an output power of *`drive`* named *`modelname`*.

Valid *`drive`* are : 1 or 4.

Terminal Names

1. `i1` : input.
2. `i0` : input.
3. `q` : output.
4. `vdd` : power.
5. `vss` : ground.

Behavior

q <= i0 xor i1

Example

```
GENLIB_MACRO(DPGEN_XOR2, "model_xor2_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 2
                       );

GENLIB_LOINS( "model_xor2_32"
            , "instance1_xor2_32"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NMUX2

Name

`DPGEN_NMUX2` — Multiplexer Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NMUX2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a `N` bits two inputs multiplexer with a complemented output and a power of `drive` named `modelname`.

Valid `drive` are : 2 or 4.

Terminal Names

1. `cmd` : select (1 bit).
2. `i1` : input (*N* bits).
3. `i0` : input (*N* bits).
4. `nq` : output (*N* bits).
5. `vdd` : power.
6. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_NMUX2, "model_nmux2_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        , 2
                        );

GENLIB_LOINS( "model_nmux2_32"
            , "instance1_nmux2_32"
            , "cmd"
            , "i1[31:0]"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_MUX2

Name

`DPGEN_MUX2` — Multiplexer Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_MUX2, char *modelname, long flags, long N,
long drive);
```

Description

Generate a *N* bits two inputs multiplexer with named *modelname*.

Valid *drive* are : 1, 2 or 4.

Terminal Names

1. `cmd` : select (1 bit).
2. `i1` : input (*N* bits).
3. `i0` : input (*N* bits).
4. `q` : output (*N* bits).
5. `vdd` : power.
6. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_MUX2, "model_mux2_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       , 1
                       );

GENLIB_LOINS( "model_mux2_32"
            , "instance1_mux2_32"
            , "cmd"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NBUSE

Name

`DPGEN_NBUSE` — Tristate Macro-Generator

Synopsis

```
#include <genlib.h>
void GENLIB_MACRO(DPGEN_NBUSE, char *modelname, long flags, long N);
```

Description

Generate a *`N`* bits two inputs tristate with a complemented output named *`modelname`*.

Terminal Names

1. `cmd` : select (1 bit).
2. `i0` : input (*`N`* bits).
3. `nq` : output (*`N`* bits).
4. `vdd` : power.
5. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_NBUSE, "model_nbuse_32"
                        , F_BEHAV|F_PLACE
                        , 32
                        );

GENLIB_LOINS( "model_nbuse_32"
            , "instance1_nbuse_32"
            , "cmd"
            , "i0[31:0]"
            , "nq[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_BUSE

Name

`DPGEN_BUSE` — tristate Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_BUSE, char *modelname, long flags, long N);
```

Description

Generate a *`N`* bits two inputs tristate with named *`modelname`*.

Terminal Names

1. `cmd` : select (1 bit).
2. `i0` : input (*`N`* bits).
3. `q` : output (*`N`* bits).
4. `vdd` : power.
5. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_BUSE, "model_buse_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       );

GENLIB_LOINS( "model_buse_32"
            , "instance1_buse_32"
            , "cmd"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NAND2MASK

Name

`DPGEN_NAND2MASK` — Programmable Mask Macro-Generator

Synopsis

```
#include <genlib.h>
void GENLIB_MACRO(DPGEN_NAND2MASK, char *modelname, long flags, long
N, char *constVal);
```

Description

Generate a *`N`* bits conditionnal NAND mask named *`modelname`*.

How it works :

- if the `cmd` signal is set to `'0'`, the mask is NOT applied, so the whole operator behave like an inverter.
- if the `cmd` signal is set to `'1'`, 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 C constant for the *`constVal`* argument. Be aware that it is a character string.

Terminal Names

1. `cmd` : mask control (1 bit).
2. `i0` : input (*N* bits).
3. `q` : output (*N* bits).
4. `vdd` : power.
5. `vss` : ground.

Behavior

```
nq <= WITH cmd SELECT not(i0)                 WHEN '0',
                      not(i0 and X"0000FFFF") WHEN '1';
```

Example

```
GENLIB_MACRO(DPGEN_NAND2MASK, "model_nand2mask_0000FFFF_32"
                            , F_BEHAV|F_PLACE
                            , 32
                            , "0x0000FFFF" /* A string! */
                            );

GENLIB_LOINS( "model_nand2mask_0000FFFF_32"
            , "instance1_nand2mask_32"
            , "cmd"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NOR2MASK

Name

`DPGEN_NOR2MASK` — Programmable Mask Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NOR2MASK, char *modelname, long flags, long
N, char *constVal);
```

Description

Generate a *N* bits conditionnal NOR mask named `modelname`.

How it works :

- if the `cmd` signal is set to `'0'`, the mask is NOT applied, so the whole operator behave like an inverter.
- if the `cmd` signal is set to `'1'`, 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 C constant for the `constVal` argument. Be aware that it is a character string.

Terminal Names

1. `cmd` : mask control (1 bit).
2. `i0` : input (*N* bits).
3. `q` : output (*N* bits).
4. `vdd` : power.
5. `vss` : ground.

Behavior

```
nq <= WITH cmd SELECT not(i0)                WHEN '0',
                      not(i0 or X"0000FFFF") WHEN '1';
```

Example

```
GENLIB_MACRO(DPGEN_NOR2MASK, "model_nor2mask_0000FFFF_32"
                           , F_BEHAV|F_PLACE
                           , 32
                           , "0x0000FFFF" /* A string! */
                           );

GENLIB_LOINS( "model_nor2mask_0000FFFF_32"
            , "instance1_nor2mask_32"
            , "cmd"
            , "i0[31:0]"
```

```
              ,  "q[31:0]"
              , "vdd", "vss", NULL
              );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_XNOR2MASK

Name

`DPGEN_XNOR2MASK` — Programmable Mask Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_XNOR2MASK, char *modelname, long flags, long
N, char *constVal);
```

Description

Generate a *`N`* bits conditionnal XNOR mask named *`modelname`*.

How it works :

- if the `cmd` signal is set to `'0'`, the mask is NOT applied, so the whole operator behave like an inverter.
- if the `cmd` signal is set to `'1'`, 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 C constant for the *`constVal`* argument. Be aware that it is a character string.

Terminal Names

1. `cmd` : mask control (1 bit).
2. `i0` : input (*`N`* bits).
3. `q` : output (*`N`* bits).
4. `vdd` : power.

5. vss : ground.

Behavior

```
nq <= WITH cmd SELECT not(i0)                 WHEN ’0’,
                      not(i0 xor X"0000FFFF") WHEN ’1’;
```

Example

```
GENLIB_MACRO(DPGEN_XNOR2MASK, "model_xnor2mask_0000FFFF_32"
                            , F_BEHAV|F_PLACE
                            , 32
                            , "0x0000FFFF" /* A string! */
                            );

GENLIB_LOINS( "model_xnor2mask_0000FFFF_32"
            , "instance1_xnor2mask_32"
            , "cmd"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_ADSB2F

Name

`DPGEN_ADSB2F` — Adder/Substractor Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_ADSB2F, char *modelname, long flags, long N);
```

Description

Generate a `N` bits adder/substractor named `modelname`.

How it works :

- if the `add_sub` signal is set to `'0'` an addition is performed, otherwise it's a substraction.
- Operation can be either signed or unsigned. In unsigned mode `c31` is the overflow, in unsigned mode you have to compute overflow by *XORing* `c31` and `c30`.

Terminal Names

1. `add_sub` : select addition or substraction (input, 1 bit).
2. `c31` : carry out. In unsigned mode, this is the overflow (output, 1 bits).
3. `c30` : used to compute overflow in signed mode : `overflow := c31 xor c30` (output, 1 bits).
4. `i1` : first operand (input, `N` bits).
5. `i0` : second operand (input, `N` bits).
6. `q` : output (`N` bits).
7. `vdd` : power.
8. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_ADSB2F, "model_adsb2f_32"
                            , F_BEHAV|F_PLACE
                            , 32
                            );

GENLIB_LOINS( "model_adsb2f_32"
            , "instance1_adsb2f_32"
            , "add_sub"
            , "c32"
            , "c31"
            , "i1[31:0]"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_SHIFT

Name

`DPGEN_SHIFT` — Shifter Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_SHIFT, char *modelname, long flags, long N);
```

Description

Generate a *`N`* bits shifter with name *`modelname`*.

How it works :

- if the `op[0]` signal is set to `'1'` performs a right shift, performs a left shift otherwise.
- if the `op[1]` signal is set to `'1'` 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(`$\log_2$`(N)) - 1`.

Terminal Names

1. `op` : select the kind of shift (input, 2 bit).
2. `shamt` : the shift amount (input, *`Y`* bits).
3. `i` : value to shift (input, *`N`* bits).
4. `o` : output (*`N`* bits).
5. `vdd` : power.
6. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_SHIFT, "model_shift_32",
                          F_BEHAV|F_PLACE,
                          32);

GENLIB_LOINS("model_shift_32",
             "instance1_shift_32",
             "op[1:0]",
             "shamt[4:0]",
             "x[31:0]",
             "y[31:0]",
             "vdd", "vss", NULL);
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_NUL

Name

`DPGEN_NUL` — Zero Detector Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_NUL, char *modelname, long flags, long N);
```

Description

Generate a `N` bits zero detector named `modelname`.

Terminal Names

1. `i0` : value to check (input, `N` bits).
2. `q` : null flag (output, 1 bit).
3. `vdd` : power.
4. `vss` : ground.

Behavior

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

Example

```
GENLIB_MACRO(DPGEN_NUL, "model_nul_32"
                      , F_BEHAV|F_PLACE
                      , 32
                      );

GENLIB_LOINS( "model_nul_32"
            , "instance1_nul_32"
            , "i0[31:0]"
            , "q"  /* Flag null. */
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_CONST

Name

`DPGEN_CONST` — Constant Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_CONST, char *modelname, long flags, long N,
char *constVal0);
```

Description

Generate a *N* bits constant named *modelname*.

Terminal Names

1. `q` : the constant (output, *N* bit).
2. `vdd` : power.
3. `vss` : ground.

Behavior

```
q <= constVal;
```

Example

```
GENLIB_MACRO(DPGEN_CONST, "model_const_0xFFFF0000"
                       , F_BEHAV|F_PLACE
                       , 4
                       , "0xFFFF0000"  /* A string! */
                       );

GENLIB_LOINS( "model_const_0xFFFF0000"
            , "instance1_const_0xFFFF0000"
            , "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_ROM2

Name

`DPGEN_ROM2` — 2 words ROM Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_ROM2, char *modelname, long flags, long N,
char *constVal0, char *constVal1);
```

Description

Generate a *`N`* bits 2 words optimized ROM named *`modelname`*.

Terminal Names

1. `sel0` : address of the value (input, 1 bits).
2. `q` : the selected word (output, *`N`* bit).
3. `vdd` : power.
4. `vss` : ground.

Behavior

```
q <= WITH sel0 SELECT
     contsVal0  WHEN B"0",
     constVal1  WHEN B"1";
```

Example

```
GENLIB_MACRO(DPGEN_ROM2, "model_rom2_set1_4"
                       , F_BEHAV|F_PLACE
                       , 4
                       , "0b1010"  /* A string! */
                       , "0b1100"  /* A string! */
                       );

GENLIB_LOINS( "model_rom2_set1_4"
            , "instance1_rom2_4"
            , "sel0"
            , "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_ROM4

Name

`DPGEN_ROM4` — 4 words ROM Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_ROM4, char *modelname, long flags, long N,
char *constVal0, char *constVal1, char *constVal2, char *constVal3);
```

Description

Generate a *`N`* bits 2 words optimized ROM named *`modelname`*.

Terminal Names

1. `sel1` : upper bit of the address (input, 1 bits).
2. `sel0` : lower bit of the address (input, 1 bits).
3. `q` : the selected word (output, *`N`* bit).
4. `vdd` : power.
5. `vss` : ground.

Behavior

```
q <= WITH sel1 & sel0 SELECT contsVal0  WHEN B"00",
                             contsVal1  WHEN B"01",
                             contsVal2  WHEN B"10",
                             constVal3  WHEN B"11";
```

Example

```
GENLIB_MACRO(DPGEN_ROM4, "model_rom4_set1_16"
                       , F_BEHAV|F_PLACE
                       , 4
                       , "0xFF00"  /* A string! */
                       , "0xCCCC"  /* A string! */
                       , "0xF0F0"  /* A string! */
                       , "0xAAAA"  /* A string! */
                       );

GENLIB_LOINS( "model_rom4_set1_16"
            , "instance1_rom4_set1_16"
            , "sel1"
            , "sel0"
            , "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_RF1

Name

`DPGEN_RF1` , `DPGEN_RF1R0` — Register File Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_RF1, char *modelname, long flags, long N);
void GENLIB_MACRO(DPGEN_RF1R0, char *modelname, long flags, long N);
```

Description

Generate a register file of ***regNumber*** words of ***N*** bits whitout decoder named ***modelname***. The `DPGEN_RF1R0` variant differs from the `DPGEN_RF1` 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.

How it works :

- `datain0` and `datain1` : the two write busses. Only one is used to actually write the register word, it is selected by the `sel` signal.
- `sel` : when set to `'0'` the `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.

Terminal Names

1. `ckok` : clock signal (input, 1 bit).
2. `sel` : select the write bus (input, 1 bit).
3. `selr` : the decoded read address (input, ***regNumber*** bits).
4. `selw` : the decoded write address (input, ***regNumber*** bits).
5. `datain0` : first write bus (input, ***N*** bits).
6. `datain1` : second write bus (input, ***N*** bits).
7. `dataout` : read bus (output, ***N*** bits).
8. `vdd` : power.
9. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_RF1, "model_rf1x8_32"
                      , F_BEHAV|F_PLACE
                      , 32  /* Words size.      */
                      , 8   /* Number of words. */
                      );

GENLIB_LOINS( "model_rf1x8_32"
            , "instance1_rf1_32"
            , "ckok"
            , "sel"
            , "selr[7:0]"
            , "selw[7:0]"
            , "datain0[31:0]"
            , "datain1[31:0]"
            , "dataout[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_FIFO

Name

`DPGEN_FIFO` — FIFO Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_FIFO, char *modelname, long flags, long N);
```

Description

Generate a FIFO of *`regNumber`* words of *`N`* bits named *`modelname`*.

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 `'0'` 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).

Terminal Names

1. `ck` : clock signal (input, 1 bit).
2. `reset` : reset signal (input, 1 bit).
3. `r` : read requested (input, 1 bits).
4. `w` : write requested (input, 1 bits).
5. `rok` : read acknowledge (output, 1 bits).
6. `wok` : write acknowledge (output, 1 bits).
7. `sel` : select the write bus (input, 1 bit).
8. `datain0` : first write bus (input, *`N`* bits).
9. `datain1` : second write bus (input, *`N`* bits).
10. `dataout` : read bus (output, *`N`* bits).
11. `vdd` : power.

12. vss : ground.

Example

```
GENLIB_MACRO(DPGEN_RF1, "model_fifox8_32"
                      , F_BEHAV|F_PLACE
                      , 32  /* Words size.      */
                      , 8   /* Number of words. */
                      );

GENLIB_LOINS( "model_fifox8_32"
            , "instance1_fifo1_32"
            , "ck"
            , "r"
            , "w"
            , "rok"
            , "wok"
            , "sel"
            , "datain0[31:0]"
            , "datain1[31:0]"
            , "dataout[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_RF1D

Name

`DPGEN_RF1D` , `DPGEN_RF1DR0` — Register File with Decoder Macro-Generator

Synopsis

```
#include <genlib.h>
void GENLIB_MACRO(DPGEN_RF1D, char *modelname, long flags, long N);
void GENLIB_MACRO(DPGEN_RF1DR0, char *modelname, long flags, long N);
```

Description

Generate a register file of `regNumber` words of `N` bits with decoder named `modelname`. The `DPGEN_RF1DR0` variant differs from the `DPGEN_RF1D` 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.

How it works :

- `datain0` and `datain1` : the two write busses. Only one is used to actually write the register word, it is selected by the `sel` signal.
- `sel` : when set to `'0'` the `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` = $\log_2$(`regNumber`).
- `wen` and `ren` : write enable and read enable, allows reading and writing when sets to `'1'`.

Terminal Names

1. `ck` : clock signal (input, 1 bit).
2. `sel` : select the write bus (input, 1 bit).
3. `wen` : write enable (input, 1 bit).
4. `ren` : read enable (input, 1 bit).
5. `adr` : the read address (input, `Y` bits).
6. `adw` : the write address (input, `Y` bits).
7. `datain0` : first write bus (input, `N` bits).
8. `datain1` : second write bus (input, `N` bits).
9. `dataout` : read bus (output, `N` bits).
10. `vdd` : power.
11. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_RF1D, "model_rf1dx8_32"
                       , F_BEHAV|F_PLACE
                       , 32  /* Words size.      */
                       , 8   /* Number of words. */
                       );

GENLIB_LOINS( "model_rf1dx8_32"
            , "instance1_rf1d_32"
            , "ck"
            , "sel"
            , "wen"
            , "ren"
            , "adr[2:0]"
            , "adw[2:0]"
            , "datain0[31:0]"
            , "datain1[31:0]"
            , "dataout[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_DFF

Name

`DPGEN_DFF` — Dynamic Flip-Flop Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_DFF, char *modelname, long flags, long N);
```

Description

Generate 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.

How it works :

- `wen` when set to `'1'` enables the writing of the flip-flop.

Terminal Names

1. `wen` : write enable (1 bit).
2. `ck` : clock signal (1 bit).
3. `i0` : data input (*N* bits).
4. `q` : output (*N* bits).
5. `vdd` : power.
6. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_DFF, "model_dff_32"
                      , F_BEHAV|F_PLACE
                      , 32
                      );

GENLIB_LOINS( "model_dff_32"
            , "instance1_dff_32"
            , "wen"
            , "ck"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_DFFT

Name

`DPGEN_DFFT` — Dynamic Flip-Flop with Scan-Path Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_DFFT, char *modelname, long flags, long N);
```

Description

Generate 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.

How it works :

- `scan` when set to `'1'` 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[31]`).

- `wen` when set to `'1'` enables the writing of the flip-flop.

Terminal Names

1. `scan` : scan-path mode (input, 1 bit).
2. `scin` : scan path in (input, 1 bit).
3. `wen` : write enable (input, 1 bit).
4. `ck` : clock signal (input, 1 bit).
5. `i0` : data input (*N* bits).
6. `q` : output (*N* bits).
7. `vdd` : power.
8. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_DFFT, "model_dfft_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       );

GENLIB_LOINS( "model_dfft_32"
            , "instance1_dfft_32"
            , "scan"
            , "scin"
            , "wen"
            , "ck"
            , "i0[31:0]"
            ,  "q[31:0]"  /* a[31] is "scout". */
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_SFF

Name

`DPGEN_SFF` — Static Flip-Flop Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_SFF, char *modelname, long flags, long N);
```

Description

Generate 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 togethers.

How it works :

- `wen` when set to `'1'` enables the writing of the flip-flop.

Terminal Names

1. `wen` : write enable (1 bit).
2. `ck` : clock signal (1 bit).
3. `i0` : data input (`N` bits).
4. `q` : output (`N` bits).
5. `vdd` : power.
6. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_SFF, "model_sff_32"
                      , F_BEHAV|F_PLACE
                      , 32
                      );

GENLIB_LOINS( "model_sff_32"
            , "instance1_sff_32"
            , "wen"
            , "ck"
            , "i0[31:0]"
            ,  "q[31:0]"
            , "vdd", "vss", NULL
            );
```

See Also

GENLIB_MACRO(3), genlib(1)

DPGEN_SFFT

Name

`DPGEN_SFFT` — Static Flip-Flop with Scan-Path Macro-Generator

Synopsis

```
#include  <genlib.h>
void GENLIB_MACRO(DPGEN_SFFT, char *modelname, long flags, long N);
```

Description

Generate a *N* bits static flip-flop with scan-path named *modelname*. The two latches of this flip-flop are i.e. each one is made of two interters looped togethers.

How it works :

- `scan` when set to `'1'` 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[31]`).

- `wen` when set to `'1'` enables the writing of the flip-flop.

Terminal Names

1. `scan` : scan-path mode (input, 1 bit).
2. `scin` : scan path in (input, 1 bit).
3. `wen` : write enable (input, 1 bit).
4. `ck` : clock signal (input, 1 bit).
5. `i0` : data input (*N* bits).
6. `q` : output (*N* bits).
7. `vdd` : power.
8. `vss` : ground.

Example

```
GENLIB_MACRO(DPGEN_SFFT, "model_sfft_32"
                       , F_BEHAV|F_PLACE
                       , 32
                       );

GENLIB_LOINS( "model_sfft_32"
            , "instance1_sfft_32"
```

```
                    , "scan"
                    , "scin"
                    , "wen"
                    , "ck"
                    , "i0[31:0]"
                    ,  "q[31:0]"  /* a[31] is "scout". */
                    , "vdd", "vss", NULL
                    );
```

See Also

GENLIB_MACRO(3), genlib(1)