## NAME

veredit 3.0.19 (Dec 4 2025)

Script based editor for structural Verilog netlists.

## SYNOPSYS

```
veredit <verilog file(s)> [<editfile>] [options]
```

### Options

```
-writeEdits            : generate default editfile.
  [-leafCells]         : include leaf cells.
  [-instLength #]      : only include cells with instantiation names
                         longer than #.
-writeDot              : generate Dotty hierarchy file.
  [-leafCells]         : include leaf cells.
  [-instLength #]      : only include cells with instantiation names
                         longer than #.
-install [MODULE-LIST] : write MPC install file.
-ifspec [MODULE-LIST]  : write GenTB ifspec file.
  [-clk CLOCK-LIST]    : space separated list of clock names.
-inst [MODULE-LIST]    : write hierarchical instance names of modules.
-edt [MODULE-LIST]     : write ED&T library.
-hier                  : write module hierarchy.
-ver                   : write Verilog file.
-top MODULE-NAME       : set top module name.
-expand                : expand port and net declarations, delete single
                         net declarations.
-o OUTFILE             : set outfile name.
```

### Edit Statements

```
1   Buffer reentrant outputs
2   Insert a buffer (or inverter) behind a cell output
3   Insert a buffer (or inverter) before a cell input
4   Adding input or output observe modules
5   Adding tie cells
6   List instances connected to a certain net
7   Adding a module
8   Deleting a module
9   Add ports or net-declarations to a module
10  Deleting ports or net-declarations from a module
11  Modify instance names
12  Add/Modify instances or module connections
13  Modify instances or module connections
14  Delete instances or module connections
15  Add/Modify assigns
16  Delete assigns
17  Create a shell around a module.
18  Replace assigns with buffers.
19  Pull pins to top level.
20  Add/Modify parameter declarations
21  Delete parameter declarations
```

## DESCRIPTION

Veredit can automate editing of a structural verilog netlist. It can automate any edit that a designer normally would do with a text editor. It has a few smarter functions like insert a buffer, buffer reentrant outputs, insert observe modules and show the hierarchy.

Veredit has proven to be mature, but the output must always be verified. The motto is: you get what you

specify.

## HISTORY

The first version of Veredit was made in 1999 during the CPA [1] project by Ondrej Popp. The first function was to modify instance names. All other edit functions have been added by Erwin Waterlander.

Veredit has been used in several IC design projects in Philips Research and Philips Semiconductors (now NXP). Veredit was part of "Paradice" flow.

Veredit was not an official company tool. It was made by IC designers for IC designers. See also [1] chapter 9.

Veredit was build with Ondrej Popp's C3PO: <https://sourceforge.net/projects/c3po>

## REFERENCES

[1] CPA designflow evaluation (Go with the flow!)
Erwin Waterlander, Ad Vaassen and Marcel Oosterhuis
Philips Research Eindhoven, NL-Rep 7121, Oct 2000

## OPTIONS

**−writeDot**

Display Netlist hierarchy

With this option veredit generates a Dotty input file that shows the hierarchy of the verilog netlist. All declared modules are included. If option −leafCells is also used, all instantiated modules are included.

Note that library cells can be removed with the −novermod option of 'dino'.

To convert the output to PDF type: dot −Tpdf −Gsize="7.5,10" foo.dot > foo.pdf

dot can be downloaded from: <http://www.research.att.com/sw/tools/graphviz/>

**−install [MODULE−LIST]**

Write MPC install file

With the −install option Veredit will generate MPC install views of verilog modules. Install files contain abstract views of leaf cells used by MPC.

MPC (Multi Purpose Network Compiler) is a program to automatically generate structure (e.g. the top level structure of a chip). Its output is VHDL or Verilog.

<https://sourceforge.net/projects/mpsc>

MPC can also generate YAPI application structure.

MODULE-LIST is an optional space seperated list of module names which will be written out. When no MODULE-LIST is specified all modules will be written.

**−ifspec [MODULE−LIST] [−clk CLOCK−LIST]**

Write GenTB ifspec file

With the −ifspec option Veredit will generate GenTB ifspec views of verilog modules.

GenTB is an automatic testbench generator. It can read a VHDL package or ifspec file and generate a testbench for the selected entity. The testbench needs a stimuli file and expected output data. The output is checked on the fly.

MODULE-LIST is an optional space seperated list of module names which will be written out. When no MODULE-LIST is specified all modules will be written.

with −clk CLOCK-LIST a list of clocks can be given. The same as with the −clk option of GenTB. The given clocks will not be in the ifspec file.

**−edt [MODULE−LIST]**

Write ED&T library

With opition −edt one can create a simple ED&T library with the interfaces of the verilog modules.

There are also fanout, fanin and area figures written in the library, but that are dummy values just to prevent complaining of the ED&T tools.

MODULE-LIST is an optional space seperated list of module names which will be written in the ED&T library. When no MODULE-LIST is specified all modules will be written.

Examples:

```
veredit foo.v -edt top -o foo.edt_lib

veredit foo.v -edt
```

**−ver**
Write Verilog file

When there are edit statements a Verilog output file will always be written. When there are no edit statements this option −ver can be used to force veredit to write out the parsed Verilog in a file.

**−top MODULE-NAME**
Write Verilog file

When −top is used Veredit will write out a verilog file with only the hierarchy from module MODULE-NAME and down.

Option −top is optional. If −top is not used Veredit will write out all modules as usual.

**−expand**
Expand port and net declarations, delete single net declarations.

With this option Veredit expands all port and net declarations and removes single wire declarations.

This is handy when you want to run some AWK or Perl script on your netlist. It makes parsing much simpler.

E.g.

```
output [2:0] a, b;
input c, d;
wire [2:0] a, b;
wire c, d;
```

becomes:

```
output [2:0] a;
output [2:0] b;
input c;
input d;
wire [2:0] a;
wire [2:0] b;
```

**−inst [MODULE−LIST]**
Write hierarchical instance names of modules.

Write hierarchal instance names to file. Names are also written to stdout. When MODULE-LIST is omitted hierachical names of all modules are printed.

Only modules that are in the netlist are listed.

**−hier**
Write module hierarchy

Write module hierarchy to file. Modules are only listed once per hierarchy level.

**−o OUTFILE**
Set the name of the output Verilog file.

## EDIT-STATEMENTS SYNTAX

Veredit expects first verilog and then edit-statements. Edit statements work on a module. An edit section for one module looks like this:

```
MODULE <modulename>
{
   [edit-statements]*
}
```

For each module you want to edit you can place such a section in an edit command file.

E.g.:

```
MODULE module_1
{
   edit-statement_1
   edit-statement_2
}

MODULE module_2
{
   edit-statement_3
   edit-statement_4
   edit-statement_5
}
```

Edit-statements are applied in the order they are written down. A next edit statement can immediate edit the result of a previous one. Only the editing of instance names (−instance ...) is always done last during netlist write out.

Edit statements can have multiple options. The order of the options is fixed as is shown in this README file. Changing the order will result in a syntax error. Some options in this file are shown between square brackets. This means that these options are optional.

For instance the edit statement '−buffer' has three options. The options −input and −output are optional. The <instantiation−list outputs> option is required. One cannot change the order of the three options.

```
MODULE <modulename>
{
   -buffer <buffer name>
   [ -input <buffer input pin name> ]
   [ -output <buffer output pin name> ]
   <instantiation-list outputs>
}
```

The <modulename> may contain one or more asterisk wildcard characters '*'. The matching algorithm is similar to file name matching. For instance to apply a certain edit-statement on all modules, one can do the following:

```
MODULE *
{
   edit-statement_1
}
```

### 1 Buffer reentrant outputs

veredit inserts buffers before outputs of the module if the output net is connected to an input pin of an instance of that module.

To determine the pins of the buffer and the direction of submodule pins it is required that the module declarations are in the input file.

syntax:

```
MODULE <modulename>
{
   -bufferoutputs <buffer name> [ -exclude <signal list> ]
}
```

The −exclude option is optional.

Example:

```
MODULE MACRO_DBESS
{
  -bufferoutputs BF4TX32_P -exclude TIMB0CLK,TIMMCORE_CLOCKDR
}
```

**2 Insert a buffer (or inverter) behind a cell output**

Veredit can insert a buffer in the netlist BEHIND A CELL OUTPUT, creating a new unique net name and a new unique instance name.

It has not much intelligence yet: − The feature does only single bit until now. − you have to provide the name of the instance output pin(s) whereto
the buffer has to be connected. − The input pin name of the buffer can be specified with the optional
parameter −input. The default input pin name is A. − The output pin name of the buffer can be specified with the optional
parameter −output. The default input pin name is Z.

You need to specify only the outputs to which the buffer will be connected.

Syntax:

```
MODULE <modulename>
{
   -buffer <buffer name>
   [ -input <buffer input pin name> ]
   [ -output <buffer output pin name> ]
   <instantiation-list outputs>
}
```

Example (will add 3 buffers):

```
MODULE test
{
   -buffer BF4TX32_P
     AO39_P  inst_a(.Z());
     FD1SQ_P inst_b(.Q());
     ND2_P   inst_C(.Z());
}
```

**3 Insert a buffer (or inverter) before a cell input**

This command works the same as the one above (−buffer), except that this command inserts a buffer BEFORE A CELL INPUT.

Syntax:

```
MODULE <modulename>
{
   -bufferinput <buffer name>
   [ -input <buffer input pin name> ]
   [ -output <buffer output pin name> ]
   <instantiation-list outputs>
}
```

Example (will add 3 buffers):

```
MODULE test
{
   -bufferinput BF4TX32_P
     AO39_P  inst_a(.A());
     FD1SQ_P inst_b(.D());
     ND2_P   inst_C(.A());
}
```

Example, buffer has non-default pin names

```
MODULE test
{
   -bufferinput BUFFD
   -input I
   -output X
     AO39_P  inst_a(.A());
     FD1SQ_P inst_b(.D());
     ND2_P   inst_C(.A());
}
```

**4 Adding input or output observe modules**

This function is used to insert observe modules for boundary scan test, but can also be used for insertion of other modules (for instance flipflops).

One has to specify the input pin and output pin of the module to insert. With the '−other' directive one can specify common nets that are connected to all of the inserted modules (for instance the clock). In addition the nets for which this has to be applied have to be given.

With −insertobsinput one can insert a module in an input net. A new unique net name is created for connection to the output of the module. In the module connections of the other instances the original input net name is also changed to the new created net name.

to add input observe modules:

```
MODULE <top module name>
{
  -insertobsinput <observe module name>
    -input <input pin name of obs module>
    -output <output pin name of obs module>
  [ -other <other pins that must be connected> ]
    -nets <net names were obs module must be inserted>
}
```

The construct
−other <other pins that must be connected> is optional.

With −insertobsoutput one can insert a module in an output net. A new unique net name is created for connection to the input of the module. In the module connections of the other instances the original output net name is also changed to the new created net name.

to add output observe module:

```
MODULE <top module name>
{
  -insertobsoutput <observe module name>
    -input <input pin name of obs module>
    -output <output pin name of obs module>
  [ -other <other pins that must be connected> ]
    -nets <net names were obs module must be inserted>
}
```

The construct

–other <other pins that must be connected> is optional.

Example:

```
MODULE psys_1801
{
   -insertobsinput dft_obs_ctrl_comb
        -input D
        -output Q
        -other .CP(dtl_mmio_clk), .HOLD(hold_inputs)
        -nets dtl_mmio_rst_an pi_sel[2:0] pi_tout

    -insertobsoutput dft_obs_ctrl_ff
        -input D
        -output Q
        -other .CP(dtl_mmio_clk), .HOLD(hold_inputs), .X(X)
        -nets pi_ack[2:0] dtl_mmio_abort_all
}
```

Note that you are allowed to omit the vector range if the net is a vector (bus) and you want to apply to the complete vector.

## 5 Adding tie cells

Veredit can add tie cells to your design. You can add any kind of tie cell. One you made yourself or one from the corelib library.

```
MODULE <module name>
{
    -tiecell <module name of the tie cell>
    -output <output pin name of the tie cell>
  [ -other <other pins that must be connected> ]
    -net <net name that needs to be connected to tie cell>
  [ -exclude <pins to exclude> ]
  [ -inst <instance names> ]
  [ -label <instance name label>]
}
```

**–tiecell**

Name of the tiecell. This can be a library cell or a cell you made yourself.

**–output**

Name of the output pin of the tie cell.

**–other**

Optional parameter to specify other pins. This is for selfmade tiecells. For instance to connect a clock and a reset pin. Use the Verilog portmap syntax to specify connections.

**–net**

Specify the nets for which a tiecell has to be inserted. This can be a list separated by spaces. Only non-bus nets are supported.

**–exclude**

Optional parameter. With this option one can specify a list of pins which will be excluded from tieing to a tiecell. For instance testpins like si and ssi. The list is separated by spaces.

**–inst**

Optional parameter. With this option one can specify instance names. Only the instances in this space separated list will get a tiecell connection, all others are skipped. Without this option all instances will get a tiecell connection if needed. Wildcard '*' is supported.

**–label**
A label will be added in the instance name of the tiecell. The instance name will be "<tiecell name><label>_inst"

An example to insert tie cells in all modules:

```
MODULE *
{
   -tiecell tie_down
   -output Z
   -other .rst_n(rst_n)
   -net 1'b0
   -exclude si ssi ssi_q
   -label _low

   -tiecell tie_up
   -output Z
   -other .rst_n(rst_n)
   -net 1'b1 vdd
   -label _high
}
```

Tie off a specific instance in module "top".

```
MODULE top
{
    -tiecell tohx1
    -output TH
    -net 1'b1
    -inst i_34 U*
}
```

## 6 List instances connected to a certain net

Veredit will create a file named "MODULE–NAME–<net name>.instlist" with a list of all the instances that have a port connected to <net name>.

This function is unfinished work and works only correct on single nets that are not part of a vector.

```
MODULE <modulename>
{
  -listinst
  -net <net name>

}
```

Example:

```
MODULE rdt_router_data
{
  -listinst
  -net clk

}
```

## 7 Adding a module

Add module to the verilog files you read in, or add it at the top of the edit-file.

## 8 Deleting a module

```
MODULE <modulename>
{
   -delete [ -empty ]
}
```

If the optional construct –empty is used the module is only deleted if it has no instances an no continuous assigns.

Example:

```
MODULE *
{
  -delete -empty
}
```

### 9 Add ports or net-declarations to a module

Specify new ports or nets with the verilog declaration list syntax.

For ports this command will add a port and a port-declaration.

When you try to add a port or net that is already in the module it will be ignored. You have to delete it first with –delports.

Example:

```
MODULE modulename
{
   -addports
     input Aap,Noot ;
     output [2:0] Banaan;
     inout Boom ;
     wire [2:0] Banaan;
}
```

### 10 Deleting ports or net-declarations from a module

Specify the ports with the verilog port-list syntax. This command will delete the port, port-declaration and net-declaration. (Wires are also deleted.)

This command can also be used to delete a net declaration (wire) when there is no port with that same name. You will get a warning that there was no port found with that name.

Examples:

```
MODULE modulename
{
  -delports
    TCP_CAS, TCP_WE, boot  // remove only these ports and net declarations.
}

MODULE modulename
{
  -delports        // remove all ports and their net declarations.
    -ALL
}
```

When using –rmports the net declarations (wires) are not deleted. Net declarations are added when they don't exist yet.

```
MODULE modulename
{
  -rmports
    TCP_CAS, TCP_WE, boot  // remove only these ports and add wires.
}

MODULE modulename
{
  -rmports
    -ALL        // this will remove all ports and place wires instead.
}
```

When using –delnets only net declarations are deleted.

```
MODULE modulename
{
  -delnets
    TCP_CAS, TCP_WE, boot  // remove only these nets.
}

MODULE modulename
{
  -delnets
    -ALL        // this will remove all nets.
}
```

**11 Modify instance names**

With this command instantiaton names can be modified.

```
MODULE <modulename>
{
   -instance oldname = newname
}
```

These edits are always applied last, during netlist write out.

A default editfile with all instantiation names can be generated with the –writeEdits option.

Additional options –leafCells and –instLength can be used to generate a default edit file. For instance to include all leaf cells and only instantiation names longer than 10 characters:

```
veredit input.v -writeEdits -leafCells -instLength 10
```

**12 Add/Modify instances or module connections**

Specify with the verilog instance syntax. This statement can

```
- add a new instance
- modify module name of an instance
- modify net name if port name exists
- create a new port connection with new net name
```

```
MODULE modulename
{
   -addinst
   [ -keepmodname ]
     newmodname instancename ( .portname(newnetname)) ;
}
```

Veredit does only use the instance name to find an instance. When option –keepmodname is used the module name of the instance will not be changed. Then the module name of the instance is a don't care.

Examples:

```
MODULE cpa
{
  -addinst
    newcell i0 (.CK(CK), .Z(Z));  // add a new cell connected to 'CK' and 'Z'
    newname i1 ();                // change module name only
    cell    i2 (.myport(newnet)); // connect 'newnet' to port 'myport'.
    cell2   i3 (.newport(reset)); // add 'newport' and connect to 'reset'
}
```

Be aware that the module name of the instance will always be changed, also when you specify new connections, unless you use option –keepmodname.

```
MODULE top
{
    -addinst -keepmodname
    module_Y inst_2 (.a(a));    // module_Y is a don't care here.
}
```

### 13 Modify instances or module connections

Specify with the verilog instance syntax. This statement can

- modify module name of an instance
- modify net name if port name exists
- create a new port connection with new net name

```
MODULE modulename
{
   -modinst
   [ -keepmodname ]
     newmodname instancename ( .portname(newnetname)) ;
}
```

This command works the same as –addinst with the exception that it will never add a new instance if it doesn't exist and it will issue a warning if the instance doesn't exist.

Veredit does only use the instance name to find an instance. When option –keepmodname is used the module name of the instance will not be changed. Then the module name of the instance is a don't care.

Examples:

```
MODULE cpa
{
  -modinst
    newname  i1 ();                 // change module name only
    mycell   i2 (.myport(newnet));  // connect 'newnet' to port 'myport'.
    mycell2  i3 (.newport(reset));  // add 'newport' and connect to 'reset'
}
```

Be aware that the module name of the instance will always be changed, also when you specify new connections, unless you use option –keepmodname.

```
MODULE top
{
    -modinst -keepmodname
    module_Y inst_2 (.a(a));    // module_Y is a don't care here.
}
```

### 14 Delete instances or module connections

Specify with the verilog instance syntax. This statement can

- delete an instance
- delete a connection from an instance
- delete all instances

When there are no module connections given veredit deletes the complete instance.

To select an instance only the instance name has to match. The cell name is a don't care.

To delete a connection you don't need to provide the net-name to which the port is connected. It is a don't care.

```
MODULE modulename
{
  -delinst
    cellname instname (.portname(netname));
}

MODULE modulename
{
  -delinst
    -ALL
}
```

Examples:

```
MODULE cpa_top
{
   -delinst
    IOP IOP_inst ();           // remove complete instance
    MT  MT_inst (.TIMC0SDA(),  // remove 3 connections from instance
          .TIMXZ2L0B0(), .A(Z));
}

MODULE global_controller
{
   -delinst
     -ALL              // delete all instances
}
```

### 15 Add/Modify assigns

use assign syntax. This statement can

- add a new assign statement
- modify an assign statement

Example:

```
MODULE modulename
{
   -addassign
     assign clkout[3:0] = TD[7:4] ;
     assign clkout[10:9] = wat[1:0];
     assign a = b ;
}
```

### 16 Delete assigns

This statement can

```
- delete an assign statement
- delete all assign statements
```

Examples:

```
MODULE modulename
{
   -delassign
     assign clkout = TQ[5] ;
     assign c = TQ[5] ;
}

MODULE modulename2
{
   -delassign
     -ALL
}
```

**17 Create a shell around a module.**

This statement will create a new module which is a shell (wrapper) around the module. IO ports will be mapped one to one. You have to specify the new module name and take care that it is unique. The instance name of the original module in the shell will be the module name with "_inst" concatenated.

```
MODULE MODULE-NAME
{
  -shell <shell module name>
}
```

After this statement you can direct run edit statements on the new module.

Example:

```
MODULE top
{
  -shell top_shell
}
```

**18 Replace assigns with buffers.**

This statement will replace simple single bit continuous assigns with a buffer. The input pin name of the buffer can be specified with the optional parameter –input. The default input pin name is `A`. The output pin name of the buffer can be specified with the optional parameter –output. The default input pin name is `Z`.

```
MODULE MODULE-NAME
{
   -bufferassigns <buffer name>
   [ -input <buffer input pin name> ]
   [ -output <buffer output pin name> ]
}
```

Example:

```
MODULE *
{
   -bufferassigns bfx1
}

MODULE *
{
   -bufferassigns BUFFD1 -input I
}
```

**19 Pull pins to top level.**

Pull pins to top level. Pins are merged up in the hierarchy. This counts for pins that already exist and for pins that meet each other, because there are multiple instances of the module in the hierarchy.

```
MODULE MODULE-NAME
{
    -pulluppin <pin list>
}
```

<pin list> is a list of pin names separated by spaces. Bus names are also supported.

Example:

```
MODULE a
{
    // this will pull up pins Q and DBUS to toplevel.
    -pulluppin Q DBUS
}
```

**20 Add/Modify parameter declarations**

use parameter declaration syntax. This statement can

- add a new parameter declaration statement
- modify a parameter declaration statement

Example:

```
MODULE modulename
{
   -addparameterdecl
    parameter x = 300 ;
    parameter y = 200 ;

}
```

**21 Delete parameter declarations**

This statement can

- delete a parameter declaration statement
- delete all parameter declaration statements

Examples:

```
MODULE modulename
{
   -delparameterdecl
    parameter y = 200 ;
}

MODULE modulename2
{
   -delparameterdecl
     -ALL
}
```

## AUTHORS

Veredit is made by C3PO, Ondrej Popp and Erwin Waterlander

Erwin Waterlander <waterlan@xs4all.nl>

Ondrej Popp <ondrejpopp@users.sourceforge.net>

C3PO: <http://sourceforge.net/projects/c3po>