cadence®

Elaboration Command-Line Options

Product Version 14.2
Decemeber 2014

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

1

Elaboration Command-Line Options

Before you can simulate your model, the design hierarchy defining the model must be elaborated. The tool you use for elaborating the design is called *ncelab*.

ncelab is a language-independent elaborator. It constructs a design hierarchy based on the instantiation and configuration information in the design, establishes signal connectivity, and computes initial values for all objects in the design. The elaborated design hierarchy is stored in a simulation snapshot, which is the representation of your design that the simulator uses to run the simulation. The snapshot is stored in the library database file along with the other intermediate objects generated by the compiler and elaborator.

Invoke *ncelab* with command-line options and arguments. You can specify the options and arguments in any order, but parameters to options must immediately follow the options they modify.

1.1 ncelab Command Syntax

The syntax of the `ncelab` command is as follows:

- Top-level design unit(s) specified on command line.

  ```
  % ncelab [options] [Lib.]Cell[:View] ...
  ```

  The top-level unit(s) specified on the command line can be:

  - One or more Verilog top-level units.
  - One or more VHDL top-level units.
  - One or more Verilog top-level units and one or more VHDL top-level units.

  For example:

  ```
  % ncelab [options] worklib.top:module
  % ncelab [options] worklib.s85:module worklib.pc:module
  ```

- Verilog configuration specified on command line.

  ```
  ncelab [options] [-libmap library_map_file] Verilog_config_name[:config]
  ```

  For example:

  ```
  % ncelab -messages -libmap lib.map cfg1
  ```

  Including `:config` after the name of the configuration is required only if the configuration name is the same as the cell name of a top-level design unit.

  If you specify a Verilog configuration on the command line, the top-level design unit is taken from the `design` statement in the configuration.

  You can specify a Verilog configuration and a top-level design unit on the command line.

- Lib.Cell:View of 5.x configuration specified on command line.

  ```
  ncelab [options] Lib.Cell:View
  ```

The *ncelab* command-line options can be entered in uppercase or lowercase, and can be abbreviated to the shortest unique string, indicated in this section with capital letters. The options listed in this section are divided into the following groups:

- General options, which apply to both languages
- VHDL-only options, which apply only to the VHDL portions of a design
- Verilog-only options, which apply only to the Verilog portions of a design

- AMS options
- NC-SC options
- Low-Power Simulation Options

1.1.1 General Options

[-64bit]
[-ABvnoassertamalg]
[-ACCESS [+] [-] *access_specification*]
[-AFile *access_file*]
[-APpend_log]
[-BBCEll *cell_name*]
[-BBCOnnect]
[-BBInst *instance_name*]
[-BBList *filename*]
[-BBOX_Create *directory*]
[-BBOX_Link *directory*]
[-BBOX_Overwrite]
[-BInding [*lib.*]*cell*[:*view*]]
[-CDS_Alternate_tmpdir *implicitTmpDir*]
[-CDS_IMPLICIT_TMPDir *implicitTmpDir*]
[-CDS_IMPLICIT_TMPOnly]
[-CDSLib *cdslib_pathname*]
[-CMdfile *compilation_command_file*]
[-CONFFIle *configuration_filename*]
[-COVDut *dut_module*]
[-COVErage *coverage_type*[:*coverage_type*]]
[-COVFile *coverage_configuration_file*]
[-ERrormax *integer*]
[-EXPand]
[-EXTBind *bind_file*]
[-EXTENDSnap *snapshot_name*]
[-FIle *arguments_filename*]
[+FSmdebug]
[-GENAfile *access_filename*]
[-GENHref *filename*]
[-GNoforce]
[-GPg *argument*]
[-GVerbose]
[-HDlvar *hdlvar_pathname*]

[-HElp]
[-HRef *filename*]
[-INCRBind *module_name*]
[-INCRPath [*top_level_unit*@]*path*]
[-INCRTop *module_name*]
[-INITBIopz]
[-INITBPx]
[-INITMEM0]
[-INITMEM1]
[-INITREG0]
[-INITREG1]
[-INTermod_path]
[-IProf]
[-LIBVerbose]
[-LICqueue]
[-LOCalbind]
[-LOGfile *filename*]
[-MAxdelays]
[-MEMdetail]
[-MESsages]
[-MINdelays]
[-MIXesc]
[-MKprimsnap]
[-NAmemap_mixgen]
[-NCError *warning_code*[:*warning_code* ...]]
[-NCFatal {*warning_code* | *error_code*}[:{*warning_code* | *error_code*} ...]]
[-NEGDelay]
[-NEG_Verbose]
[-NEVerwarn]
[-NOASsert]
[-NOBinding *design_unit_name*]
[-NOCopyright]
[-NODEAdcode]
[-NOLog]
[-NOMxindr]
[-NO_Sdfa_header]
[-NOSOurce]
[-NOSTdout]
[-NO_TCHK_Msg]
[-NOTImingchecks]
[-NOWarn *warning_code*[:*warning_code* ...]]

[-NTC_Warn]
[-OLddeposit]
[-OMicheckinglevel *checking_level*]
[-PARtialdesign]
[-PErfstat]
[-PRIMBind]
[-PRIMHrefupdate]
[-PRIMLibdir *directory*]
[-PRIMName *name*[@*directory*]]
[-PRIMParamsok]
[-PRIMSnap *snapshot_name*]
[-PRIMTop *module_name*]
[-PRIMVhdlcompat]
[-PRINt_hdl_precision]
[-Quiet]
[-SDF_Cmd_file *sdf_command_file*]
[-SDF_NO_Warnings]
[-SDF_Precision *precision*]
[-SDF_Verbose]
[-SNapshot *snapshot_name*]
[-STatus]
[-TYpdelays]
[-UPDate]
[-UPTodate_messages]
[-VErsion]
[-WANdwor_compat]
[-WARnmax *integer*]
[-WOrk *work_library*]
[-Zlib *compression_level*]

1.1.2 VHDL Only Options

[-DYnvhpi]
[-GENEric *generic_name => value*]
[-LIB_Binding]
[-NODEFbopen]
[-NOIpd]
[-NOVitalaccl]
[-NOXilinxaccl]
[-NO_TCHK_Xgen]
[-NO_VPD_Msg]

[-NO_VPD_Xgen]
[-PREserve]
[-Relax]
[-V93]
[-VHDLSParsearray *value*]
[-VHDLSYnc]
[-VHDL_Time_precision *time_precision*]
[-VIPDMAx]
[-VIPDMIn]

1.1.3 Verilog Only Options

[-ACCESSReg [+] [-] *access_specification*]
[-ACCU_PATH_Delay]
[-ACCU_PATH_Verbose]
[-ADd_seq_delay *delay_value*]
[-ALways_trigger]
[-ANno_simtime]
[-ARr_access]
[-CAint]
[-DEFAult_delay_mode *delay_mode*]
[-DEFParam *parameter_pathname* = *value*]
[-DELAY_MODE [*full_path*[...]=]{path | distributed | unit | zero | none}]
[-DELAY_MODE_Punit]
[-DELTa_sequdp_delay]
[-DISABLE_Enht]
[-DPI_Void_task]
[-DPIHeader *filename*]
[-DPIImpheader *filename*]
[-DUmptiming *filename*]
[-ENable_eto_pulse]
[-EPULSE_NEg]
[-EPULSE_NOneg]
[-EPULSE_ONDetect]
[-EPULSE_ONEvent]
[-EXTEND_TCHECK_Data_limit *percent_relaxation*]
[-EXTEND_TCHECK_Reference_limit *percent_relaxation*]
[-GAteloopwarn]
[-IEEe1364]
[-LIBMap *library_map_file* [*library_map_file* ...]]
[-LIBName *library_name*]

[-LOADPli1 *shared_lib_name*:*boot_func_name*[:export][,*boot_func_name ...*]]
[-LOADVpi *shared_lib_name*:*boot_func_name*[:export][,*boot_func_name ...*]]
[-NCInitialize]
[-NOAUtosdf]
[-NOEsp]
[-NONEg_tchk]
[-NONOtifier]
[-NORtis]
[-NOSPecify]
[-NTC_Level *ntc_level*]
[-NTC_NEglim]
[-NTCNOtchks]
[-NTC_Poslim]
[-NTC_Tolerance *tolerance_level*]
[-NTC_Verbose]
[-OVERRIDE_Precision]
[-OVERRIDE_Timescale]
[-PATHPulse]
[-PATHTran]
[-PLI_Export]
[-PLINOOptwarn]
[-PLINOWarn]
[-PLIVerbose]
[-PULSE_E *error_percent*]
[-PULSE_INT_E *error_percent*]
[-PULSE_INT_R *reject_percent*]
[-PULSE_R *reject_percent*]
[-SDF_File *sdf_filename*]
[-SDF_NOCheck_celltype]
[-SDF_NOPAthedge]
[-SDF_NOPUlse]
[-SDF_Orig_dir]
[-SDF_SImtime]
[-SDF_SPecpp]
[-SDF_SPLIT_Two_timing_check]
[-SDF_SPLITVLOG_Setuphold]
[-SDF_SPLITVLOG_Recrem]
[-SDF_Worstcase_rounding]
[-SDFDir *directory*]
[-SDFSTats *filename*]
[-SEM2009]

[-SET_eto_pulse]
[-SEQ_udp_delay *delay_specification*]
[-SEQUDP_nba_delay]
[-SHow_forces]
[-SPArsearray *number_of_array_elements*]
[-SVPerf {+ | -} *checking_specification*]
[-TFile *timing_file*]
[-TImescale '*time_unit / time_precision*']
[-TRanmin]
[-VPicompat {1364v1995 | 1364v2001 | 1364v2005 | 1800v2005 | 1800v2008}]
[-XFile *filename*]
[-XLifnone]
[-XProp {F | C}]
[-XVerbose]

1.1.4 AMS Options

[-AMSFastspice]
[-AMSPartinfo *part_file*]
[-DISCipline *discipline_name*]
[-DResolution]
[-IEReport]
[-MODELIncdir *pathname* [:*pathname*]]
[-MODELPath *argument*]
[-NOParamerr]
[-PROpspath *property_file*]
[-SEtdiscipline *argument*]
[-SPECTRE_Argfile_spp *arg_file*]
[-SPECTRE_E]
[-SPECTRE_Spp]
[-USE5X4VHdl]
[-USE5X4VLog]

1.1.5 NC-SC Options

[-LOADSc *library_name*]
[-SCCreateviewables]
[-SCOnly]
[-SCParameter *param_name* = *value*]
[-SCTop *name*]
[-SCUpdate]

1.1.6 Low-Power Simulation Options

[-LPS_Assign_ft_buf]
[-LPS_Blackboxmm]
[-LPS_CEllrtn_off]
[-LPS_COnst_aon]
[-LPS_CPf *cpf_filename*]
[-LPS_DISABLE_Condsig_replay]
[-LPS_DISABLE_Force_mem]
[-LPS_Dtrn_min]
[-LPS_Force_reapply]
[-LPS_IMPLICITPSO_char '*value*']
[-LPS_IMPLICITPSO_nonchar *value*]
[-LPS_INT_index_nocorrupt]
[-LPS_INT_nocorrupt]
[-LPS_ISO_Off]
[-LPS_ISO_Verbose]
[-LPS_ISOFilter_verbose]
[-LPS_ISORuleopt_warn]
[-LPS_LOG_verbose *filename*]
[-LPS_LOGfile *filename*]
[-LPS_MOdules_wildcard]
[-LPS_MTrn_min]
[-LPS_MVs]
[-LPS_NO_xzshutoff]
[-LPS_NOtlp]
[-LPS_PA_model_on]
[-LPS_PMCheck_only]
[-LPS_PMOde]
[-LPS_PSn_verbose {1 | 2}]
[-LPS_RTN_Lock]
[-LPS_RTN_Off]
[-LPS_SImctrl_on]
[-LPS_SRFilter_verbose]
[-LPS_SRRuleopt_warn]
[-LPS_STDby_nowarn]
[-LPS_STIme *time*]
[-LPS_STL_off]
[-LPS_Upcase]
[-LPS_VERBose {1 | 2 | 3 | 4}]
[-LPS_VERIfy]

[-LPS_VPlan *vplan_filename*]

1.2 ncelab Command Options

This section describes the options that you can use with the `ncelab` command. Options can be entered in upper or lowercase. Capital letters indicate the shortest possible abbreviation for an option.

1.2.1 -64bit

Invoke the 64-bit version of the *ncelab* executable.

Besides including the `-64bit` command-line option when you run the executables, you can also run the 64-bit version by:

- Setting the `INCA_64BIT` environment variable.
- Setting up your `PATH` and library path environment variables so that you are pointing to the 64-bit version.

  See "64-Bit Version of the Simulator" for more information.

The `-64bit` command-line option is ignored if you have already specified that you want to run in 64-bit mode by setting environment variables.

You cannot use the `-64bit` option in flows with tools that do not support 64-bit.

This option is ignored if you include it with the `NCVLOGOPTS` or `NCVHDLOPTS` variable in an `hdl.var` file.

1.2.2 -ABvnoassertamalg

Disables the optimization for multiple identically-clocked assertions. See Maximizing Assertion Performance in the *Assertion Writing Guide* for more information.

1.2.3 -ACcess [+] [-] access_specification

Set the visibility access for all objects in the design. The *`access_specification`* argument can be:

- `r` (read access)
- `w` (write access)
- `c` (connectivity access)

- Any combination of these three access types

Use the plus sign (+) to turn on the specified access. Use the minus (-) sign to turn off the specified access. If no plus or minus sign is used, + is the default. The + and - options apply to all subsequent r, w, or c specifications until the next + or -.

By default, objects do not have read, write, or connectivity access. In other words, the default is -access -r-w-c.

Objects that are given write access are also given read access. Objects that are given connectivity access are also given write access, and, therefore, read access.

Examples:

1. Read access only

```
-access +r (same as -access r)
```

2. Write access (objects also get read access)

```
-access +w (same as -access w)
```

3. Read/Write access

```
-access +r+w (same as -access +rw or -access rw)
```

4. Read/Write/Connectivity access

```
-access +r+w+c (same as -access +rwc or -access rwc)
```

5. Connectivity access

```
-access +c (same as -access c)
```

Note: Objects that are given connectivity access are also given read and write access. The following option results in connectivity, read, and write access to all objects:

```
-access +c-rw
```

You can also use multiple -access options. For example:

```
-access +r -access -w
```

See "Enabling Read, Write, or Connectivity Access to Simulation Objects" for more information.

1.2.4 -ACCESSReg [+] [-] access_specification

(Verilog only)

Set the visibility access for registers only.

The `-access` option sets the visibility access for all objects in the design. Use the `-accessreg` option (`ncelab -accessreg` or `irun -accessreg`) to set the access for Verilog regs only.

See the `-access` option for a description of the *`access_specification`* argument.

The `-accessreg` option has the same behavior as the `-access` option, except that it applies access only to the following Verilog object types:

- `reg`
- `integer`
- `real`
- `time`
- `event`

If both `-access` and `-accessreg` are included on the command line, the access specified by the `-accessreg` option will be used for the above objects.

See "Enabling Read, Write, or Connectivity Access to Simulation Objects" for more information.

1.2.5 -ACCU_PATH_Delay

Enables the Enhanced Timing Output (ETO) delay algorithm by default for all modules with specify blocks that qualify. If this option is not used, the ETO delay algorithm will only be used for those modules with the `pathdelay_enhanced` qualifier in the specify block.

The following example shows how to use the ETO delay algorithm without the `-accu_path_delay` option:

```
specify
pathdelay_enhanced;          // ETO qualifier needed in specify block (without
                             // -accu_path_delay option)
(A => OUT) = 5;
(B => OUT) = 10;
endspecify
```

This option will print a list of all modules with specify blocks that did *not* qualify for the ETO delay algorithm during elaboration. Include the `-accu_path_verbose` option to get more detailed information about what disqualified a module from using the ETO delay algorithm.

See Enhancing Path Delay Accuracy for details on the ETO delay algorithm.

1.2.6 -ACCU_PATH_Verbose

Prints a reason why a cell was disqualified from using the Enhanced Timing Output (ETO) delay algorithm during elaboration. This option must be used in conjunction with the `-accu_path_delay` option.

1.2.7 -ACg

Enable a more accurate interconnect annotation analysis.

Note: The `-acg` option was deprecated in INCISIV Release 13.2. Using this option on the command line will cause the elaborator to generate the following warning:

```
*W,ACGWARN: ACG is the default interconnect behavior.
```

The default behavior supports the ability to stack interconnect delays. In previous releases, the default behavior did not allow stacked interconnect delays, and produced a less accurate representation.

Example:

The following SDF file annotates two interconnect delays where the following two delays should be stacked:

(INTERCONNECT dut.c.o dut.o(2))

(INTERCONNECT dut.c.b.a.o dut.c.b.o (3))

```
(DELAYFILE
  (DESIGN "TOP")
  (DIVIDER .)
  (VOLTAGE  )
  (PROCESS  )
  (TEMPERATURE  )
  (TIMESCALE)
  (CELL
    (CELLTYPE  "top")
```

```
    (INSTANCE  top)
    (DELAY
      (ABSOLUTE
        (INTERCONNECT dut.c.b.a.o dut.c.b.o (3))
        (INTERCONNECT dut.c.o dut.o (2))
      )
    )
  )
)


`timescale 1ns/10ps
module A(o, i);
  output o;
  input i;
  buf (o, i);
endmodule

module B(o, i);
  output o;
  input i;
  A a(o, i);
endmodule

module C(o, i);
  output o;
  input i;
  B b(o, i);
endmodule

module D(o, i);
  output o;
  input i;
  C c(o,i);
endmodule

module top;
  reg r;
  wire w, w1;
  D dut(w1, w);
  assign w = r;
```

```
  initial
    begin
      $monitor("%d:\tr=%b -> dut.c.b.a.o=%b -> top.dut.c.b.o=%b ->
          top.dut.c.o=%b top.dut.o=%b w=%b", $time, r, top.dut.c.b.a.o,
          top.dut.c.b.o, top.dut.c.o, top.dut.o, w);
      $sdf_annotate("test.sdf", , ,"sdf.log");
      #100 r = 0;
      #100 r = 1;
end
endmodule
```

When the values are not stacked, the simulation shows the point between `dut.c.b.a.o` to `dut.c.b.o` as having a delay of 5. When the values are stacked in the second run, the more accurate result shows the delay from `dut.c.b.a.o` to `dut.c.b.o` as having the correct delay of 3.

Simulation without stacked interconnect delays:

```
0:      r=x -> dut.c.b.a.o=x -> top.dut.c.b.o=x -> top.dut.c.o=x top.dut.o=x w=x
100:    r=0 -> dut.c.b.a.o=0 -> top.dut.c.b.o=x -> top.dut.c.o=x top.dut.o=x w=0
105:    r=0 -> dut.c.b.a.o=0 -> top.dut.c.b.o=0 -> top.dut.c.o=0 top.dut.o=0 w=0
```

Simulation with stacked interconnect delays:

```
0:      r=x -> dut.c.b.a.o=x -> top.dut.c.b.o=x -> top.dut.c.o=x top.dut.o=x w=x
100:    r=0 -> dut.c.b.a.o=0 -> top.dut.c.b.o=x -> top.dut.c.o=x top.dut.o=x w=0
103:    r=0 -> dut.c.b.a.o=0 -> top.dut.c.b.o=0 -> top.dut.c.o=0 top.dut.o=x w=0
105:    r=0 -> dut.c.b.a.o=0 -> top.dut.c.b.o=0 -> top.dut.c.o=0 top.dut.o=0 w=0
```

1.2.8 -ADd_seq_delay delay_value

Updates undelayed sequential UDPs with a specific delay value. This option applies delays to those sequential UDPs that do not have a path delay already provided in the instantiation.

The *`delay_value`* argument is an integer that is annotated using the primitive's precision. If the delay has the following extensions the absolute delay is used:

```
fs
ps
ns
```

Primitives that already have a delay are not modified. No other timing aspects are modified by the use of this option.

For example, if the *timescale* in the following snippet is used, then the command: `irun -add_seq_delay 2 ...` results in `u1` having a delay of `20ps`.

```
`timescale 1ns/10ps
...
myseq u1(...
```

Likewise, the command: `irun -add_seq_delay 1ns ...` results in a `1ns` delay for `u1`.

Specifying a Specific Instance

You can use `-add_seq_delay` to specify a specific instance. For example, the following line applies a `40ps` delay to the `top.dut_top.u3` instance:

```
-add_seq_delay top.dut_top.u3=40ps
```

If the specific UDP already has a delay in the design, the specified delay value is ignored.

Note: When specifying a specific instance, the = (equal sign) is required and there can be no spaces between the instance and the value. For example, the following lines are not valid:

```
-add_seq_delay top.dut_top.u3 40ps
-add_seq_delay top.dut_top.u3 = 40ps
```

See "Timing Delays and Race Conditions in Gate-Level Netlists" on page 173 for more information on using the option to avoid race conditions. Also see the -seq_udp_delay, -delta_sequdp_delay, and -sequdp_nba_delay for information on specifying delays.

1.2.9 -AFile access_file

Use the specified access file. An access file is a text file that lets you set the visibility access for particular instances or portions of a design. See "Using an Access File" for details on writing and using an access file.

Use the -access option to specify global visibility access for all objects in the design.

The `-afile` option can also be used to include a *PLI map file*. A PLI map file associates user-defined system tasks and system functions with functions in a PLI application. The file contains a line for each user-defined system task or system function your application needs. In each line, you specify:

- The name of the system task or system function.
- Additional specifications for the system task or system function.

  For a user-defined system function, you must specify the size of the return value.

Other, optional, specifications include the name of the call function, the name of the check function, the name of the misc function, and the data value passed as the first argument to the call, check, and misc routines.

The PLI map file can be created as a separate file, which you can include at elaboration time using the `-afile` option, or at simulation time with the `-plimapfile` option. If passed at elaboration time, the system tasks and functions defined in the file are known to both *ncelab* and *ncsim*. If passed at simulation time, the system tasks and functions defined in the file are known only to *ncsim*.

```
ncelab -afile plimapfile.file ....
irun -afile plimapfile.file ....

ncsim -plimapfile plimapfile.file ....
irun -plimapfile plimapfile.file ....
```

You can also include the PLI map information in an access file. An access file must be included at elaboration time, so if you include the PLI map information in an access file, use the `-afile` option, as shown above.

See the section “Using a PLI/VPI Map File” in the chapter “Using VPI” in the *VPI User Guide and Reference* for details on the PLI map file.

1.2.10 -ALways_trigger

(Verilog only)

Run `always` blocks that have an event control at time zero when an object on the sensitivity list is modified either during time zero or before simulation begins.

An `always` block is not sensitive to value changes on its sensitivity list until it executes for the first time at time zero and the `always @(...)` statement is reached. For example:

```
module test();
   logic x;
   logic y;
   ...
   ...
   always @(x)
      y = x;

endmodule
```

In this example, `x` can be initialized to something other than `1'bx` by:

- Using the -ncinitialize option.
- A Tcl `deposit` or `force` command before the first `run` command.
- An initializer in the declaration of `x` (for example, `logic x = 1'b1;`).

Because these initializations occur before the `always @(...)` statement is executed, the initialization will not trigger the execution of the `always` block. Variable `y` would not be equal to `x` until `x` changes again to some other value.

The same behavior may occur if `x` is initialized in an `initial` block, as follows:

```
module test();
   logic x;
   logic y;

   initial
      x = 1;

   always @(x)
      y = x;

endmodule
```

The Verilog LRM (Section 9.9) states that there “shall be no implied order of execution between initial and always constructs.” Therefore, if the `initial` block that initializes the objects on the sensitivity list runs before the `always` block, the `always` block will not see the change of value, and the block will continue to wait for the next value change. If the `always` block runs first, then it will be waiting for a value change when the `initial` block runs, and this will cause the `always` block to wake up.

An `always` block is often meant to be sensitive to value changes from the very beginning, so any change to any objects on its sensitivity list should cause it to execute. To delay the initializations until after all `always` blocks get a chance to run up to their `@(...)` statement, you can alter the Verilog code to:

- Specify a delay in the initialization statement.

```
initial
#0 x = 1;
```

- Use the SystemVerilog `always_comb` construct.

```
always_comb
   x = y;
```

If you cannot change your source code, use the `-always_trigger` option (`ncelab -always_trigger` or `irun -always_trigger`). This option causes the event ordering to change so that `always` blocks run at time zero in response to the initialization of an object on the sensitivity list no matter what caused that initialization.

1.2.11 -AMSFastspice

(AMS)

Use the UltraSim solver.

See the description of the `-amsfastspice` option in the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for additional information.

1.2.12 -AMSPartinfo part_file

(AMS)

Use the specified file, which contains mixed-signal partition and connect module insertion information.

See the description of the `-amspartinfo` option in the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for additional information.

1.2.13 -ANno_simtime

(Verilog only)

Enable the use of PLI/VPI routines that modify delays at simulation time. These routines are `acc_replace_delays`, `acc_append_delays`, and `vpi_put_delays`.

If this option is not specified at elaboration time, and a PLI/VPI routine that modifies delays is executed at simulation time, a message is issued and the delay modification does not take place.

This option disables optimizations in the simulator that take delays into account, and will, therefore, have some performance impact. In addition, the option sets the default access to simulation objects to read/write when the design is elaborated, which can have a severe performance impact. Use this option only if you intend to modify delays at simulation time.

Note: Negative limit values in `$setuphold` or `$recrem` timing checks cannot be modified using PLI/VPI routines.

1.2.14 -APpend_log

Append log information from multiple runs of *ncelab* to one log file. Use this option if you are going to run *ncelab* multiple times and you want all log information appended to one log file. If you do not use this option, the log file is overwritten each time you run *ncelab*.

If you use both `-append_log` and `-nolog` on the command line, `-nolog` overrides `-append_log`.

Because the log file is opened before variables in the `hdl.var` file are read, the `-append_log` option is ignored with a warning if you define it with the `NCELABOPTS` variable in an `hdl.var` file.

1.2.15 -ARr_access

(Verilog only)

Store all Verilog arrays in byte-aligned format, as mandated for memories by the LRM. This format lets you use the PLI routine `tf_nodeinfo()` to access array values for single-dimensional arrays of registers (memories).

By default, the simulator optimizes the array layout for fast access. However, this format does not allow access to array data using the `tf_nodeinfo` interface.

Because the array layout must be homogeneous throughout a snapshot, the `-arr_access` option is turned on by default if any design unit has been compiled with the `ncvlog -nomempack` option.

Using the `-arr_access` option may decrease the amount of memory consumed for elaboration and simulation, but may have a negative effect on simulation performance.

1.2.16 -BBCEll cell_name

Ignore the specified cell when elaborating the design.

For some simulations, you may want to ignore some parts of the design hierarchy and treat these parts of the hierarchy as black boxes to increase the performance of the simulator. One way to do this is to replace actual models in the design with empty (black box) modules. You can create empty modules for the actual modules you want to replace, compile these units into a library, and then write a configuration to specify the source description to be used to represent each instance in the design.

The `-bbcell` option lets you blackbox all instances of a particular cell from the command line without changing the HDL or writing a configuration. The argument to this option is the *`Lib.Cell`* specification of the cell you want to ignore. All instances of the specified cell will be treated as a black box.

Only one *`cell_name`* can be specified. You can use multiple `-bbcell` options on the command line.

Note: Because this option changes the design hierarchy, the design must be re-elaborated. Elaboration (*ncelab*) performance may decrease, depending on the number of instances being blackboxed. However, simulation (*ncsim*) performance will improve as the blackboxed hierarchy will not be simulated.

Note: Coverage is not available in designs using blackboxing.

Example:

```
module top;
  reg a,b;

  hier1 u1();

  initial
    $display("\nTop Vlog");
endmodule

module hier1;
  reg a,b;

  hier2 u21();
  hier2 u22();
  hier2 u23();

  initial
    $display("\nHier1 Vlog");
endmodule

module hier2;
  reg a,b;

  hier3 u3();

  initial
    $display("\nHier2 Vlog");
endmodule
```

```
module hier3;
  reg a,b;

  initial
    $display("\nHier3 Vlog");
endmodule

# Compile
% ncvlog -mess vlog.v
file: vlog.v
        module worklib.top
                errors: 0, warnings: 0
        module worklib.hier1
                errors: 0, warnings: 0
        module worklib.hier2
                errors: 0, warnings: 0
        module worklib.hier3
                errors: 0, warnings: 0
# Elaborate the design. Ignore all instances of cell worklib.hier2.
% ncelab -messages -access +rwc -bbcell worklib.hier2 top
        Elaborating the design hierarchy:
...
        Generating native compiled code:
                worklib.hier1:module <0x6deb054e>
                        streams:   1, words:    56
                worklib.top:module <0x589e81d9>
                        streams:   1, words:    56
...
        Writing initial simulation snapshot: worklib.top:module
# Simulate
% ncsim -input sim.tcl worklib.top
ncsim> run 1ns

Hier1 Vlog

Top Vlog
Ran until 1 NS + 0
ncsim> scope -desc /top
a.........variable reg = 1'hx
b.........variable reg = 1'hx
```

```
u1.........instance of module hier1
ncsim> scope -desc /top/u1
a..........variable reg = 1'hx
b..........variable reg = 1'hx
u21........instance       u21, u22, and u23 are reported simply as
                          "instance", not as "instance of module hier2"
u22........instance
u23........instance
ncsim> force top.u1.b 1'b1
ncsim> value top.u1.b
1'h1
;# Operations on blackboxed instances or objects in these instances are errors
ncsim> force top.u1.u21.a 1'b1
ncsim: *E,PUNBND: Path name contains an unbound instance: u21.
ncsim> value top.u1.u21.b
ncsim: *E,PUNBND: Path name contains an unbound instance: u21.
ncsim> force top.u1.u22.u3.b 1'b1
ncsim: *E,PUNBND: Path name contains an unbound instance: u22.
ncsim> value top.u1.u23.a
ncsim: *E,PUNBND: Path name contains an unbound instance: u23.
ncsim> exit
```

If you are simulating in single-step mode with *irun*, use the following command:

```
% irun -access +rwc -bbcell worklib.hier2 [other_options] -input sim.tcl vlog.v
```

If you need a finer-grained level of control over instances to be ignored, use the -bbinst option, which lets you specify a particular instance to be ignored, or the -bblist option, which lets you specify a file that contains a list of instances to be blackboxed. In order to *preserve* the port mapping information for a cell being treated as a black box, you can use the -bbcell option with the -bbconnect option.

1.2.17 -BBCOnnect

Do not ignore the cell or instance being treated as a black box, and preserve all port mapping information.

The -bbconnect option requires one of the following compatible blackbox options to run: -bbcell, -bbinst, or -bblist. When using the -bbconnect option together with one of the compatible blackbox options, the elaborator recognizes the cell or instance that is being treated as a black box and preserves its boundary ports information. The output ports of the

module specified as a black box will be treated as undriven nets while the input and inout ports will be driven with the following values:

Data Type	Initial Value
Bit, integer, logic, real	Left
Std_logic	Z
SV net	Z
SV reg	X

You can also use more than one `-bbconnect` option on the command line. When using multiple `-bbcell` or `-bbinst` options, you should have one `-bbconnect` option for each corresponding `-bbcell` or `-bbinst` option.

Example:

```
module top;
  wire [0:3] r;
  wire [0:3] r1;

  mid1 m1 (r);
  mid2 m2 (r1);
endmodule

module mid1 (output [0:3] r);
  assign r = 4'b0010;
  initial
    $display ("inside instance m1\n");
endmodule

module mid2 (output [0:2] r1);
  bot b1 (r1);
  initial
    $display ("in instance m2 value of r1 changed \n");
endmodule

module bot (output [0:2] rb);
  assign #2 rb = 3'b001;
  always@(rb)
    $display ("in instance b1 value of rb changed \n");
```

```
endmodule

# Compile
% ncvlog -mess -sv top.sv
file top.sv
        module worklib.top
                errors: 0, warnings: 0
        module worklib.mid1
                errors: 0, warnings: 0
        module worklib.mid2
                errors: 0, warnings: 0
        module worklib.bot
                errors: 0, warnings: 0
# Elaborate the design. Blackbox worklib.mid2:sv,
# run connectivity checks, and preserve port information
% ncelab -messages -access +rwc -bbcell worklib.mid2 top -bbconnect
        Elaborating the design hierarchy:
...
        Generating native compiled code:
                worklib.bot:module <0x7e70832d>
                        streams:   2, words    617
                worklib.mid1:module <0x1dcaa83e>
                        streams:   2, words    687
                worklib.mid2:module <0x59db9be1>
                        streams:   2, words    909
...
        Writing initial simulation snapshot: worklib.top:module
# Simulate
% ncsim -input sim.tcl worklib.top
ncsim> scope -desc
r..........net (wire/tri) logic [0:3] = 4'hx
r1.........net logic [0:3]
   r1[0] (wire/tri) = StX
   r1[1] (wire/tri) = StX
   r1[2] (wire/tri) = StX
   r1[3] (wire/tri) = StX
m1.........instance of module mid1
m2.........instance of module mid2          m2 is not ignored during simulation
ncsim> run
inside instance m1
inside instance m2
```

```
in instance m2 value of r1 changed
in instance b1 value of rb changed

ncsim> scope -desc m1
r.........output net (wire/tri) logic [0:3] = 4'h2

ncsim> scope -desc m2
r1.........output net logic [0:2]
   r1[0] (wire/tri) = St0
   r1[1] (wire/tri) = St0
   r1[2] (wire/tri) = St1
b1.........instance of module bot
```

If you are simulating in single-step mode with *irun*, use the following command:

```
% irun -access +rwc -bbcell worklib.mid2 -bbconnect -input sim.tcl top.sv
```

1.2.18 -BBInst instance_name

Ignore the specified instance when elaborating the design.

For some simulations, you may want to ignore some parts of the design hierarchy and treat these parts of the hierarchy as black boxes to increase the performance of the simulator. One way to do this is to replace actual models in the design with empty (black box) modules. You can create empty modules for the actual modules you want to replace, compile these units into a library, and then write a configuration to specify the source description to be used to represent each instance in the design.

The `-bbcell` option lets you blackbox all instances of a particular cell from the command line without changing the HDL or writing a configuration. The argument to this option is the *`Lib.Cell`* specification of the cell to be ignored. All instances of the specified cell will be treated as a black box.

Use the `-bbinst` option if you need to blackbox an instance of a module, but not all instances. The argument to this option is the hierarchical name of the instance that you want to blackbox. For example:

```
% ncelab -messages -access +rwc -bbinst top.u1.u22 \
[other_options] top_level_unit
Or:
% irun -bbinst top.u1.u22 [other_options] source_files
```

Only one *`instance_name`* can be specified. You can use multiple `-bbinst` options on the command line. Alternatively, you may want to specify the instance names in a file and then

include the file using the -bblist option. In order to *preserve* the port mapping information for an instance being treated as a black box, you can use the -bbinst option with the -bbconnect option.

Note: Because this option changes the design hierarchy, the design must be re-elaborated. Elaboration (*ncelab*) performance may decrease, depending on the number of instances being blackboxed. However, simulation (*ncsim*) performance will improve as the blackboxed hierarchy will not be simulated.

Note: Coverage is not available in designs using blackboxing.

Example:

```
% irun -nocopyright -access +rwc -bbinst top.u1.u22 -input sim.tcl vlog.v
file: vlog.v
        module worklib.top:v
                errors: 0, warnings: 0
        module worklib.hier1:v
                errors: 0, warnings: 0
        module worklib.hier2:v
                errors: 0, warnings: 0
        module worklib.hier3:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
...
...
        Writing initial simulation snapshot: worklib.top:v
Loading snapshot worklib.top:v ..................... Done
ncsim> run 1 ns
Ran until 1 NS + 0
ncsim> scope -desc /top
a.........variable reg = 1'hx
b.........variable reg = 1'hx
u1........instance of module hier1
ncsim> scope -desc /top/u1
a.........variable reg = 1'hx
b.........variable reg = 1'hx
u21.......instance of module hier2
u22.......instance                    u22 is reported simply as "instance", not
                                      as "instance of module hier2"
u23.......instance of module hier2
ncsim> force top.u1.b 1'b1
```

```
ncsim> value top.u1.b
1'h1
ncsim> force top.u1.u21.a 1'b1
ncsim> value top.u1.u21.b
1'hx
;# Operations on blackboxed instances or objects in these instances are errors
ncsim> force top.u1.u22.u3.b 1'b1
ncsim: *E,PUNBND: Path name contains an unbound instance: u22.
ncsim> exit
```

1.2.19 -BBList filename

Ignore the instances listed in the specified file when elaborating the design.

You can ignore (blackbox) all instances of a particular cell in a design by using the -bbcell option, or ignore one particular instance with the -bbinst option. If you want to blackbox several instances, you can list the hierarchical instance names in a file, and then include the file with the -bblist option. The argument to this option is the path to a file that contains a list of hierarchical instance names. For example:

```
# File: bblist.txt
top.u1.u21
top.u1.u22

% ncelab -messages -bblist bblist.txt [other_options] top_level_unit
Or:
% irun -bblist bblist.txt [other_options] source_files
```

In order to preserve port mapping information for several instances in a file being treated as black boxes, you can use the -bblist option with the -bbconnect option.

Note: Because this option changes the design hierarchy, the design must be re-elaborated. Elaboration (*ncelab*) performance may decrease, depending on the number of instances being blackboxed. However, simulation (*ncsim*) performance will improve as the blackboxed hierarchy will not be simulated.

Note: Coverage is not available in designs using blackboxing.

Example:

```
% cat bblist.txt
# File: bblist.txt
top.u1.u21
top.u1.u22
```

```
% irun -nocopyright -access +rwc -bblist bblist.txt -input sim.tcl vlog.v
file: vlog.v
        module worklib.top:v
                errors: 0, warnings: 0
        module worklib.hier1:v
                errors: 0, warnings: 0
        module worklib.hier2:v
                errors: 0, warnings: 0
        module worklib.hier3:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
...
...

        Writing initial simulation snapshot: worklib.top:v
Loading snapshot worklib.top:v ..................... Done
ncsim> run 1 ns
Ran until 1 NS + 0
ncsim> scope -desc /top
a..........variable reg = 1'hx
b..........variable reg = 1'hx
u1.........instance of module hier1
ncsim> scope -desc /top/u1
a..........variable reg = 1'hx
b..........variable reg = 1'hx
u21........instance          u21 and u22 are reported simply as "instance", not
                             as "instance of module hier2"
u22........instance
u23........instance of module hier2
ncsim> force top.u1.b 1'b1
ncsim> value top.u1.b
1'h1
;# Operations on blackboxed instances or objects in these instances are errors
ncsim> force top.u1.u21.a 1'b1
ncsim: *E,PUNBND: Path name contains an unbound instance: u21.
ncsim> value top.u1.u21.b
ncsim: *E,PUNBND: Path name contains an unbound instance: u21.
ncsim> force top.u1.u22.u3.b 1'b1
ncsim: *E,PUNBND: Path name contains an unbound instance: u22.
ncsim> exit
```

1.2.20 -BBOX_Create directory

Generate the snapshot and its dependent files in the specified directory.

When elaborating a primary snapshot, use this option to create a self-sufficient IP model that will act as a black box. This black box contains the snapshot, and includes all dependent files and work libraries in the designated directory. If the directory does not already exist, then the elaborator will create it automatically.

For example, an IP author may elaborate two Verilog design files, `ip1.v` and `ip2.v`, using multiple `-bbox_create` options. When creating self-sufficient IP models from the two primary snapshots, this option also creates `./tmp/box1` and `./tmp/box2` if the two directories do not already exist.

The following shows how to create IP1 using the design `ip1.v`:

```
% ncvlog ip1.v
% ncelab -mkprimsnap -incrbind top -bbox_create ./tmp/box1 ip1
```

Or use *irun*, as shown:

```
% irun -mkprimsnap -incrbind top -bbox_create ./tmp/box1 ip1.v
```

Similarly, to create IP2 using the design `ip2.v`:

```
% ncvlog ip2.v
% ncelab -mkprimsnap -incrbind top2 -bbox_create ./tmp/box2 ip2
```

Or use *irun*:

```
% irun -mkprimsnap -incrbind top2 -bbox_create ./tmp/box2 ip2.v
```

Use the `-bbox_link` option to integrate the available self-sufficient IP models.

Note: The `-cds_implicit_tmpdir` and `-bbox_create` options are not compatible. Do not use both options together on the command line.

This option (`ncelab -bbox_create` or `irun -bbox_create`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.21 -BBOX_Link directory

Load the IP model located in the specified directory.

When elaborating the simulation snapshot, use this option to attach a self-sufficient IP model from a particular directory without having to re-elaborate the design. If there are multiple IP models (or black boxes), then you must specify multiple `-bbox_link` options.

For example, the `-bbox_create` option is used to elaborate two self-sufficient IP models. The blackboxed IP models IP1 and IP2 are available to the SOC integrator in the `./tmp/box1` and `./tmp/box2` directories. The SOC design is in the incremental partition. Using the `-bbox_link` option, the SOC integrator can attach IP1 and IP2 to the design, `soc.v`.

The following shows how to integrate the blackboxed IP models into the SOC design, `soc.v`:

```
% ncvlog soc.v
% ncelab -primbind -bbox_link ./tmp/box1 \
  -bbox_link ./tmp/box2 soc
```

Or use *irun*, as shown:

```
% irun -primname ip1 -bbox_link ./tmp/box1 -primname ip2 \
  -bbox_link ./tmp/box2 soc.v
```

Note: The `cds.lib` file cannot map the same logical name to multiple physical paths. At SOC level, the `cds.lib` file should include libraries specific to SOC design.

This option (`ncelab -bbox_link` or `irun -bbox_link`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.22 -BBOX_Overwrite

Replace the contents of the directory specified when updating a self-sufficient IP model.

When elaborating the primary snapshot, you can use the `-bbox_create` option to create a self-sufficient IP model that will act as a black box. This black box contains the snapshot, and includes all dependent files and work libraries in the designated directory.

Use the `-bbox_overwrite` option in those cases when you need to update the IP model and replace the contents of the specified directory. For example:

```
# Build and elaborate the original blackboxed model
% ncvlog ip1_a.v
% ncelab -mkprimsnap -incrbind top -bbox_create ./tmp/box1 ip1

# Build and elaborate the updated blackboxed model
% ncvlog ip1_b.v
% ncelab -mkprimsnap -incrbind top -bbox_create ./tmp/box1 ip1 -bbox_ovewrite
```

Or use *irun*, as shown:

```
% irun -mkprimsnap -incrbind top -bbox_create ./tmp/box1 ip1_a.v
% irun -mkprimsnap -incrbind top -bbox_create ./tmp/box1 ip1_b.v -bbox_overwrite
```

This option (`ncelab -bbox_overwrite` or `irun -bbox_overwrite`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.23 -BInding [lib.]cell[:view]

Force an explicit binding to the specified compiled design unit. You can use the `-binding` option to force an explicit binding to:

- A specified Verilog module or UDP.

- A specified VHDL architecture when instantiating a VHDL design unit into Verilog.
- A specified Verilog module or UDP, or VHDL architecture, when instantiating the Verilog or VHDL design unit into SystemC.

For example, suppose that you have different views (RTL, gate-level, and so on) for a Verilog module called `foo`, and that you have compiled these design units with different view names (`foo:rtl`, `foo:gate`, and so on) into a library called `worklib`. You can use the `-binding` option as follows to specify that you want to bind to the RTL view.

```
% ncelab -binding worklib.foo:rtl top_level_module
```

In this example, specifying the library is optional because all views have been compiled into the same library. However, the different views could be compiled into different libraries, or two views with the same name could be compiled into different libraries. It is recommended that the argument be explicitly specified by using the complete `lib.cell:view` syntax.

The `-binding` option is global to the design. Once the first instance has been resolved, all instances of the same module or UDP are resolved the same way. Use a configuration to force different bindings for modules or UDPs with the same name.

See "How Modules and UDPs Are Resolved During Elaboration" for more information.

In a mixed-language design in which a Verilog module instantiates a VHDL entity with multiple architectures, the elaborator will not bind the instance of the entity because there are multiple possible bindings. For example, suppose that entity `dff` has three architectures called `first`, `second`, and `third`. In the Verilog module, the entity is instantiated as follows:

```
dff u1 (q, en);
```

You can use the `-binding` option to specify which architecture to use.

```
% ncelab -binding worklib.dff:second top_level_design_unit
```

1.2.24 -CAint

(Verilog only)

Annotate an SDF `PORT` delay or `INTERCONNECT` delay even if the source port and the load port are not hierarchically connected by a wire because they have been disconnected by a unidirectional continuous assignment statement.

The Verilog LRM (Section 16.2.4, “SDF annotation of interconnect delays”) states that, when annotating an `INTERCONNECT` construct:

“If the source port is not found, or if the source port and the load port are not actually on the same net, then a warning message is issued, but the delay to the load port is annotated anyway. If this happens for a load port that is part of a multi-source net, then the delay is treated as if it were the delay from all source ports, which is the same as the annotation behavior for a `PORT` delay.”

By default, the SDF annotator adheres to the IEEE standard. If the source port and the load port are not actually on the same net, warning messages are issued. For a `PORT` interconnect, the delay is annotated at the destination. An `INTERCONNECT` delay is replaced with a `PORT` annotation at the destination.

In some cases, the source and destination for the requested interconnect are disconnected because a synthesis tool has inserted a unidirectional continuous assignment to alias two or more nets together. The `-caint` option can be used to override the default behavior of the annotator for these cases.

If you use the `-caint` option, the SDF annotator does not generate warning messages about the source and destination being separated by a unidirectional continuous assignment. For a `PORT` delay, the destination port is annotated without a warning. For a multi-source interconnect delay (MSID), unique delays are annotated between each source/load pair. If the destination is to change value, the delay associated with the monitored driver will be used to schedule the MSID’s output transition. The continuous assign must change value before the drivers of the continuous assign can be used to calculate the transition delay.

Note: The continuous assignment cannot have delays. If it does, a warning is generated, and the interconnect is modeled as a `PORT` delay.

The destination of the interconnect must be driven by the unidirectional continuous assignment. Continuous assigns enclosed by an SDF interconnect request and that are driving the source of the interconnect are ignored.

1.2.25 -CDS_Alternate_tmpdir implicitTmpDir

Specify an alternate library directory to search for design data.

For example:

```
% ncvlog tb.v dut.v
% ncelab -mkprimsnap -cds_implicit_tmpdir primdir dut
% ncelab -primsnap dut -cds_alternate_tmpdir primdir tb
```

The software writes the specified alternate library directory to the snapshot header for later use by the simulator. The `-cds_alternate_tmpdir` option is not required on the command line when simulating the testbench, as shown:

```
% ncsim tb
```

1.2.26 -CDS_IMPLICIT_TMPDir implicitTmpDir

Specify an implicit directory to search for design data and to hold new design data.

The software writes this option to the snapshot header for later use by the simulator and *ncupdate*.

1.2.27 -CDS_IMPLICIT_TMPOnly

Force the elaborator to look at only design data within the *`implicitTmpDir`* specified by the `-cds_implicit_tmpdir` option. When the `-cds_implicit_tmponly` option is not used, the elaborator also considers design data found in the libraries defined by `cds.lib` files.

The `-cds_implicit_tmponly` option can be used only when the `-cds_implicit_tmpdir` option is also used.

1.2.28 -CDSLib cdslib_pathname

Use the specified `cds.lib` file. See "The cds.lib File" for details on the `cds.lib` file.

All tools and utilities that read a `cds.lib` file use a default search mechanism to find the `cds.lib` file. See "The setup.loc File" for information on this search mechanism. Use the `-cdslib` option to override the default search order and force the elaborator to use the specified `cds.lib` file.

Example:

```
% ncelab -cdslib ~/design_lib/cds.lib top
```

The elaborator reads the `cds.lib` file before it processes any variables defined in the `hdl.var` file. You cannot, therefore, include the `-cdslib` option with the `NCELABOPTS` variable in an `hdl.var` file.

1.2.29 -CMdfile compilation_command_file

Use the specified compilation command file when updating the design with the `-update` option.

This option can be used if the location of a source file has been changed. The compilation command file contains a definition of the `SEARCH_PATH` variable, which lists the directories to be searched for locating the design files.

```
% ncelab -update -cmdfile cmdfile.cmd top_level_unit
```

See "Compiling Source Files by Specifying the Top-Level of the Design" for details on the compilation command file.

1.2.30 -CONFFIle configuration_filename

Generate a VHDL configuration file with the specified name for the design unit specified on the command line. When you include the `-conffile` option to generate a configuration file, the elaborator generates the configuration file and then exits. The design is not actually elaborated.

You can use a VHDL configuration declaration to configure a VHDL, Verilog, or mixed Verilog/VHDL design.

You must use the `-conffile` option to generate a configuration. This option has several suboptions that you can use to control the generator.

See "VHDL Configuration File Generator" for details on the configuration generator, and for a description of all options that are specific to the generator.

1.2.31 -COVDut dut_module

Specify a design under test for coverage.

Use the `-covdut` option to limit instrumentation of selected coverage to an instance of *`dut_module`* with its sub-hierarchy. You can use multiple `-covdut` options on the command line.

See the chapter "Generating Coverage Data" in the *ICC User Guide* for details on this option.

1.2.32 -COVErage coverage_type[:coverage_type]

Enable coverage data generation.

The following coverage types can be specified as the argument:

- `block`–Enable block coverage.
- `expr`–Enable expression coverage.
- `fsm`–Enable fsm coverage.
- `toggle`–Enable toggle coverage.
- `Functional`–Enable functional coverage
- `all`–Enable all supported code coverage types.

You can specify more than one coverage type by separating the coverage types with a colon. For example:

```
% ncelab -messages -coverage block worklib.top
% ncelab -messages -coverage block:fsm worklib.top
```

Note: Enabling coverage may turn off some optimizations, such as dead code optimizations. In addition, to ensure accurate scoring of coverage, read access may be automatically provided to some objects in the design.

See the chapter "Generating Coverage Data" in the *ICC User Guide* for details on this option.

1.2.33 -COVFile coverage_configuration_file

Use the specified configuration file for code coverage instrumentation.

The `-covfile` option is used to control instrumentation in more detail by limiting the scope of instrumentation. This configuration file includes commands that need to be executed during instrumentation. The commands that you include in the configuration file are based on the type of coverage you are implementing. You can either include all the commands in one configuration file or create separate configuration files for each type of coverage. If you create separate configuration files, you must specify multiple `-covfile` options.

For example, the following command specifies a configuration file named `cov.args`.

```
% ncelab -covfile cov.args worklib.top:v
```

See the chapter “Generating Coverage Data” in the *ICC User Guide* for details on the commands you can include in a configuration file.

1.2.34 -DEFAult_delay_mode delay_mode

(Verilog only)

Applies a specific delay mode to all Verilog modules with no delay mode specified at their top.

This option lets you specify an explicit delay mode for modules which do not have a delay mode directive in the source file. The argument to the `-default_delay_mode` option is one of the delay modes supported for the `-delay_mode` option.

Argument	Description
`-default_delay_mode path`	Modules with no delay mode specified will simulate in Path delay mode. If no module path delay is defined, distributed delays are used.
`-default_delay_mode distributed`	Modules with no delay mode specified will simulate in Distributed delay mode.
`-default_delay_mode zero`	Modules with no specified delay mode will simulate in zero delay mode.
`-default_delay_mode unit`	Modules with no specified delay mode will simulate in unit delay mode.
`-default_delay_mode none`	Model simulates with the delays specified in the model's source description files.
`-default_delay_mode default`	All modules where delay mode is not specified will behave with default delay mode (without any impact of other delay modes specified in the design or by compilation order).

Argument	Description
`-default_delay_mode` *`full_path`*[...]=*`delay_mode`*	This option applies the specified delay mode to the module instance(s) (with no delay mode specified at the top of the design) specified by the *`full_path`* argument. This argument must be a fully qualified path name starting at the top of the design hierarchy, and it must refer to: ■ A top-level module ■ A module instance ■ An instance array If the path refers to an instance array, the mode is applied to each instance in the array. The argument can be entered in uppercase or lowercase.

Notes:

- `-delay_mode` and `-default_delay_mode` options cannot be used together.
- If more than one `-default_delay_mode` option is specified, the mode with the highest precedence (path, distributed, unit, zero, none, default) is used, regardless of the order of the options on the command line.
- If multiple options with path names are given, they are applied in the order in which they appear on the command line. Therefore, the ones that appear later on the command line will override previous ones to the extent that they refer to the same module instances.
- Wildcard characters and square brackets used in instance-selects must be escaped or enclosed in quotes.

1.2.35 -DEFParam parameter_pathname=value

(Verilog only)

Specifies a value for a Verilog parameter.

The `-defparam` option is used to override the value of a parameter specified in the source. The value specified on the command line overrides the initial value, as well as any value changes made with a `defparam` statement or with an instance value parameter change.

You can pass a value to a parameter at any level of the design hierarchy. Values can be passed across language boundaries. For example, if the design is a mixed VHDL-Verilog-VHDL design, you can assign a value to a parameter of the lower-level Verilog design unit.

The argument to `-defparam` must specify the hierarchical path of the parameter and the value to be assigned.

Note: Hierarchical references terminating in Verilog are allowed to pass through VHDL but must follow Verilog language syntax. For example, suppose that you have a VHDL testbench (top-level entity is called `ent1`) that instantiates a Verilog unit `I1`, and that there is a parameter called `param1` declared inside `I1` that you want to override. The following syntax must be used:

```
-defparam ent1.I1.param1=50
```

You cannot use the following syntax:

```
-defparam :I1.param1=50
```

The value can be an integer, a real, or a string. No spaces are allowed in the argument. For example:

```
% ncelab -defparam top.dut.u1.param4=8 ....
% ncelab -defparam top.dut.u1.param4=-6 ....
% ncelab -defparam top.param2=12.0e45 ....
% ncelab -defparam top.dut.param3=0x5 ....
```

If the *value* part of the argument is a string that begins with a letter, quotation marks are optional. However, if you enclose the string in quotation marks, you must escape the quotation marks with backslash characters. The backslash characters are a requirement of the shell in which the command is issued. For example:

```
% ncelab -defparam top.param1=hello ....
% ncelab -defparam top.param1=\"hello\" ....
```

Backslash characters must be used for strings that do not begin with a letter. For example:

```
-defparam top.param1=\".good.bye\"
-defparam top.TESTPATH=\"./tests\"
```

If the value is a based number (for example, `2'bx`), you must include a backslash character, as shown in the following example.

```
-defparam top.param=12\'bx
```

The following items are not supported:

- Expression evaluation as part of a value

  ```
  -defparam top.param1=3+4       // Illegal
  ```

```
-defparam top.param2=param1+1  // Illegal
```

- Expression evaluation of array of instances

```
-defparam top.aoi[1].param="bob"    // Legal
-defparam top.aoi[3-2].param="bob"  // Illegal
```

Use multiple `-defparam` options to specify values for multiple parameters. For example:

```
% ncelab -defparam top.abc=8 -defparam top.b1.xyz=7 worklib.top
% irun -defparam top.abc=8 -defparam top.b1.xyz=7 test.v
```

The order in which the parameters are specified does not matter. However, if two values for the same parameter are specified, the value specified with the last `-defparam` option on the command line is used.

Note: You can also use the `-gpg` option to change the value of parameter. The `-gpg` option assigns a value to all VHDL generics and Verilog parameters in the design with a specified name.

1.2.36 -DELAY_MODE [full_path[...]=] {path | distributed | unit | zero | none}

(Verilog only)

Use the specified delay mode for the Verilog portions of the design.

The delay mode can be: `path`, `distributed`, `unit`, `zero`, or `none`. Delay mode `none` will set the delay mode to the mode specified in the source. This is the same as not using the `-delay_mode` command-line option.

You can use this option to:

- Set the delay mode for the entire design.
- Set the delay mode for the entire design, and override this delay mode for a specific instance or instances.

Setting the Delay Mode for the Entire Design

To set the delay mode for the entire design, specify the mode with the `-delay_mode` option. For example:

```
ncelab -delay_mode zero .... (or: irun -delay_mode zero ....)
ncelab -delay_mode unit .... (or: irun -delay_mode unit ....)
```

If more than one `-delay_mode` option is specified, the mode with the highest precedence is used, regardless of the order of the options on the command line. The precedence order is as shown above, with `none` having the lowest precedence. In the following example, delay mode `distributed` will be applied because it has a higher precedence:

```
-delay_mode distributed -delay_mode unit
```

Overriding the Delay Mode for Specific Instances

You can override the delay mode for a particular module instance or instances by using the following syntax:

`-delay_mode` *`full_path`*`[...]=`*`delay_mode`*

This option applies the specified delay mode to the module instance(s) specified by the *`full_path`* argument. This argument must be a fully qualified path name starting at the top of the design hierarchy, and it must refer to:

- A top-level module
- A module instance
- An instance array

  If the path refers to an instance array, the mode is applied to each instance in the array. The path can also include an instance-select, in which case the mode will apply only to the specified instance within the instance array.

Include the optional `...` after the path name if you want to apply the mode to the Verilog portions of the design hierarchy below the instance indicated by the path name.

If multiple options with path names are given, they are applied in the order in which they appear on the command line. Therefore, the ones that appear later on the command line will override previous ones to the extent that they refer to the same module instances.

The last element of the full path name can include the `*` and `?` wildcard characters. If the path name consists of only one element (top-level module name), it may not include wildcard characters.

Note: Wildcard characters and square brackets used in instance-selects must be escaped or enclosed in quotes. You can also put quotes around the whole argument, which is easier if there are many characters that need escaping.

If you are running the simulator in single-step mode with *irun*, the elaborator output will include information on which instances received their delay modes from `-delay_mode` options that include path names. For example:

```
% irun -access +rwc -delay_mode board.\*=path board.v counter.v clock.v ff.v
```

```
...
Elaborating the design hierarchy:
        Top level design units:
                board
        Delay mode overrides (subsequent options override previous ones):
                (default: none)
                 board.*=path applied to:
                        board.counter
                        board.clockGen
...
```

If you are running the simulator in multi-step mode, include the `-messages` option on the `ncelab` command line to generate these messages.

Examples:

- Default delay mode is `none`. Use `path` delays for module instance `top.u1`.

  ```
  -delay_mode top.u1=path
  ```

- Default delay mode is `none`. Use path delays for module instance `top.u1` and all hierarchy below it.

  ```
  -delay_mode top.u1...=path
  ```

- Use `path` delays for all instances within `top`, except `top.u2`. You must escape the wildcard character, enclose the wildcard character in quotes, or enclose the whole argument in quotes.

  ```
  -delay_mode top.\*=path -delay_mode top.u2=none
  Or:
  -delay_mode top."*"=path -delay_mode top.u2=none
  Or:
  -delay_mode "top.*=path" -delay_mode top.u2=none
  ```

- Use `unit` delays for all instances in the array `ua`, except `ua[2]`. Square brackets must be escaped or enclosed in quotes. You can also enclose the whole argument in quotes

  ```
  -delay_mode top.ua=unit -delay_mode top.ua\[2\]=none
  -delay_mode top.ua=unit -delay_mode "top.ua[2]=none"
  ```

- Use `unit` delays for the whole design except for module instance `top.u1` and all hierarchy below `top.u1`, which uses `zero` delays.

  ```
  -delay_mode unit -delay_mode top.u1...=zero
  ```

- Use `zero` delays for all hierarchy below `top.u1`, but not for `top.u1` itself.

  ```
  -delay_mode top.u1.\*...=zero
  ```

See "Selecting a Delay Mode" for more information on specifying a delay mode.

1.2.37 -DELAY_MODE_Punit

Disable all timing checks and set all path delays to simulate with the value of one simulation unit. Where a simulation unit is equal to 1 module precision.

Example:

```
irun -delay_mode_punit test.v
```

Or:

```
ncelab -delay_mode_punit tb
```

1.2.38 -DELTa_sequdp_delay

Adds a delta delay to sequential UDPs.

Applying a delta delay to the sequential logic allows the combinational logic to settle before updating the output of the sequential logic.

See "Timing Delays and Race Conditions in Gate-Level Netlists" on page 173 for more information on using the option to avoid race conditions. Also see the -seq_udp_delay, -add_seq_delay, and -sequdp_nba_delay for information on specifying delays.

1.2.39 -DISABLE_Enht

(Verilog only)

Disable enhanced timing features. These timing features are enabled by using special properties in a specify block. Using the properties gives you more control over the selection of a delay when there are multiple inputs that occur either simultaneously or while a path delay output is already scheduled. See "Specify Properties for Module Path Delays" for more information.

This option also disables the enhanced path delay selection algorithm, which is enabled by using the `pathdelay_enhanced` specify block qualifier. See "Enhancing Path Delay Accuracy" for details on the enhanced delay selection algorithm.

1.2.40 -DISCipline discipline_name

(AMS)

Specifies the discipline of discrete nets for which a discipline is otherwise undefined.

See the description of the `-discipline` option in the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for additional information.

1.2.41 -DPI_Void_task

(Verilog only)

Specifies that the return value of exported and imported tasks will be `VOID`.

Prior to the IUS 8.1 release, exported or imported tasks did not have return values, and C and SystemC functions that corresponded to an exported or imported task had to return a `void` type. Due to the addition of support for the `disable` construct within DPI-based designs, C and SystemC functions that correspond to an imported or exported task are required to return an `int` value. For example, the following defines a C task called `imp_task`, which will be imported into SystemVerilog:

```
int imp_task_c (int x, int y){ /* Return type is int */
..int dis_ret;
  dis_ret = exp_task_c(x,y); /*Return type is int */
  return (dis_ret);
}
```

For backward compatibility, use the `-dpi_void_task` option on existing DPI designs. Designs will not be affected by this new requirement and will behave as they did prior to IUS 8.1. However, you must adhere to this new style of DPI function declaration in order to use the `disable` functionality with DPI-based designs.

See “Direct Programming Interface” in the *SystemVerilog Reference* for details on DPI.

1.2.42 -DPIHeader filename

(Verilog only)

Generate a header file for SystemVerilog Direct Programming Interface (DPI) export functions and tasks.

Export tasks and functions are tasks and functions implemented in SystemVerilog that are called from import tasks/functions. The header file contains definitions for all of the C identifiers that correspond to exported tasks and functions contained in the elaborated snapshot. This header file can then be included in all C files from where exported functions have been called.

Example:

```
% ncelab -dpiheader myheader.h worklib.top:module
```

If you are running the simulator in single-step invocation mode with *irun*, use the following command:

```
% irun -dpiheader myheader.h source.v -elaborate
```

Note: If you are running in single-step mode and want to generate a header file, include the `-elaborate` option. After the header file has been generated and included in the C files, and a shared object has been created, you can run *irun* again to simulate the design.

See "Direct Programming Interface" in the *SystemVerilog Reference* for details on DPI.

1.2.43 -DPIImpheader filename

Generate a header file for SystemVerilog Direct Programming Interface (DPI) import functions and tasks.

The generated header file contains all C function prototypes for corresponding imported functions or tasks declared with the keyword `DPI`, `DPI-C`, or `DPI-SC` in SystemVerilog. The header file can be included with your C application code to check the consistency of the function prototypes with their actual declarations or definitions.

See "Direct Programming Interface" in the *SystemVerilog Reference* for details on DPI.

1.2.44 -DResolution

(AMS)

Specifies that the detailed discipline resolution method is to be used to determine the discipline of nets that do not otherwise have defined disciplines.

See the description of the `-dresolution` option in the chapter "Elaborating" in the *Virtuoso AMS Designer Simulator User Guide* for additional information.

1.2.45 -DUmptiming filename

(Verilog only)

Generate a file that contains the design hierarchy along with all the timing information within each scope. This file can be used to check what has actually been annotated from SDF files.

By default, delays and timing checks specified in the HDL are used during simulation. If you update the limits or delays with SDF, the annotated values are used. You can use the `-sdf_verbose` option to include detailed information in the SDF log file to verify that all the

checks and delays specified in the SDF file are correctly applied to the netlist. However, the output does not include the reverse analysis - information on netlist elements that have or have not been annotated.

Use the `-dumptiming` option (`ncelab -dumptiming` or `irun -dumptiming`) to generate an output file that will help you verify what has been actually been annotated from the SDF files.

The output file, which is in XML format, contains the design hierarchy along with all the timing information in each scope. The timing data includes an annotation flag to indicate if it has been annotated or not (`anno='y'` or `anno='n'`). You can find all the items that have not been annotated by scanning the file for `anno='n'`.

The following is an example of the output for a scope from an output file:

```
<scope>
  <decl
    lib="worklib" cell="it" view="v"
    fullname="top.u1b"
    filelineno="2" filepath="./test.v"
    lang="verilog"
    tunit="ns" punit="ps"
  />
  <specify>
    <pathdelay>
      <path="in1 full out"/>
      <pdelay anno='y' delay="1000"/>
    </pathdelay>
    <pathdelay>
      <path="in2 full out"/>
      <pdelay anno='n' delay="1200"/>
    </pathdelay>
    <pathdelay>
      <path="in3 full out"/>
      <pdelay anno='n' delay="1030"/>
    </pathdelay>
    <tcheck type="$setuphold" in1="in1" edge1="posedge" in2="in2"
edge2="negedge" l1val="-500" l1anno='y' l2val="10000" l2anno='n'/>
    <tcheck type="$setuphold" in1="in2" edge1="posedge" in2="in3"
edge2="posedge" l1val="5010" l1anno='n' l2val="5003" l2anno='n'/>
    <tcheck type="$setuphold" in1="in2" edge1="posedge" in2="in3"
edge2="negedge" l1val="5010" l1anno='y' l2val="5003" l2anno='y'/>
    <ntc_nets>
```

```
      <net name="in1" delay="501"/>
    </ntc_nets>
  </specify>
  <sdfinterconnect>
    <portdelay dst="out" delay="4000"/>
  </sdfinterconnect>
</scope>
```

Note: Because the `-dumptiming` option is an elaboration option, the output file will not show any of the timing data from the SDF files that are annotated during simulation time (`-sdf_simtime`).

1.2.46 -DYnvhpi

(VHDL only)

Enable the creation of dynamic drivers.

See “Creating Dynamic Drivers” in the chapter “VHPI Operations” in the *VHPI User Guide* for details on creating dynamic drivers.

1.2.47 -ENable_eto_pulse

Enable accurate output pulse modeling for all modules using the Enhanced Timing Output (ETO) delay algorithm.

You can define ETO modules in one of two ways:

- automatically, using the `-accu_path_delay` option
- manually, using the `pathdelay_enhanced` qualifier in the specify block

The `-enable_eto_pulse` option triggers the `-accu_path_delay` option and assigns a value of X to the output for the duration of the pulse.

Note: Simulation results may change when using this option.

To avoid X-propagation use the `-set_eto_pulse` option.

See Enhancing Path Delay Accuracy for details on the ETO delay algorithm.

1.2.48 -EPULSE_NEg

(Verilog only)

Filter canceled events (negative pulses) to the e state. This option makes canceled events visible. Using this option overrides any `showcancelled` and `noshowcancelled` settings in `specify` blocks. See "Pulse Filtering and Canceled Schedules" for more information.

1.2.49 -EPULSE_NOneg

(Verilog only)

Do not filter canceled events (negative pulses) to the e state. Using this option overrides any `showcancelled` and `noshowcancelled` settings in `specify` blocks. See "Pulse Filtering and Canceled Schedules" for more information.

1.2.50 -EPULSE_ONDetect

(Verilog only)

Use On-Detect filtering of error pulses. This option extends the `e` state back to the edge of the event that caused the pulse to occur.

See "Pulse Filtering Style" for details on On-Detect and On-Event pulse filtering styles.

1.2.51 -EPULSE_ONEvent

(Verilog only)

Use On-Event filtering of error pulses.

See "Pulse Filtering Style" for details on On-Detect and On-Event pulse filtering styles.

1.2.52 -ERrormax integer

Abort after reaching the specified number of errors. By default, there is no limit on the number of error messages.

By using `-errormax`, you can limit the number of errors that are generated, fix those errors, and then rerun to check for other errors. This option is useful when you are running a large design that might contain numerous errors.

Example:

```
% ncelab -errormax 10 worklib.top
```

1.2.53 -EXPand

Expand all vector nets that have been compressed.

For performance and memory capacity reasons, vector Verilog wires and VHDL signals are compressed by default if the model does not require operations on individual bits of the vector.

For Verilog, you cannot perform the following operations on a compressed Verilog wire unless you have expanded the vectors with the `-expand` option:

- Force or release a value on a subelement of a compressed vector.
- Probe a subelement of a compressed vector to an SHM database.
- Set a breakpoint on a subelement of a compressed vector.

Note: For VHDL, you cannot probe a subelement of a compressed vector to an EVCD database unless you have included the `-evcd -mode lfcompat` option on the `probe` command line. For example:

```
ncsim> probe -create :top:cans(0) -evcd -mode lfcompat -database test_default
```

You can use the `value` command to display the value of the vector or the value of a subelement of the vector, and the `describe` command to describe the vector or a subelement of the vector.

Note: Using the `-expand` option can have a severe impact on performance. Try to perform operations on the entire vector, if possible. In Verilog, you can use the `scalared` keyword to expand a specific vector if you must operate on a single bit. For example:

```
wire scalared [31:0] mainbus;
```

For Verilog, the `-expand` option expands vector nets the same way that the `-x` option in Verilog-XL expands vector nets. Using `-expand` will thus eliminate mismatches due to how vectored nets are expanded when you compare databases generated by the two simulators.

1.2.54 -EXTBind bind_file

Specify an external binding file.

The bind file contains SystemVerilog `bind` directives that bind properties to design units. All Verilog and VHDL binds can be specified together in one file.

If you are simulating in multi-step invocation mode, or in single-step invocation mode with *irun*, use the `-extbind` option.

```
% ncelab -extbind bindfile.txt ....
% irun -extbind bindfile.txt ....
```

See the chapter "Using SVA" in the *Assertion Writing Guide* for details on binding SystemVerilog assertions to SystemVerilog and VHDL.

1.2.55 -EXTENDSnap snapshot_name

Extend the specified snapshot by including the additional source files specified on the command line.

The `-extendsnap` option reads an existing snapshot and then builds a new snapshot that includes additional files. This option is typically used to add a test harness to an existing snapshot for a DUT. By using the option, you can avoid rewriting complex scripts or altering the verification environment when adding new files.

Example:

Suppose that you have source files that constitute a DUT, and that you have compiled the files and generated a snapshot using the following `irun` command:

```
irun -c -access +r file1.v file2.v -snapshot mydut
```

To extend the snapshot `mydut` with other files (`file1.e` and `file3.v`) to verify the DUT, invoke *irun* with the `-extendsnap` option and the filenames, as follows:

```
irun -access +rwc file3.v file1.e -extendsnap mydut
```

See "Extending a Snapshot to Include Additional Source Files" for more details on using the `-extendsnap` option.

1.2.56 -EXTEND_TCHECK_Data_limit percent_relaxation

(Verilog only)

Extend the violation regions established by a pair of setuphold or recrem timing checks with negative values in which the timing checks contain two different constraints for posedge and negedge of data with respect to the same reference signal and in which the violation regions do not overlap.

In situations where there are two setuphold or recrem timing checks that establish two different constraints for posedge and negedge of data with respect to the same reference signal, violation regions may not overlap. Because the violation regions created by the timing

checks do not overlap, the negative timing check algorithm does not converge. This results in both of the negative limits being set to zero, thus underestimating the actual speed of the design.

You can avoid this non-convergence by hand-editing the timing checks in the HDL or in the SDF file to create some overlap, or you can use the `-extend_tcheck_data_limit` or the `-extend_tcheck_reference_limit` option to automatically extend the violation regions by the specified percentage to create the overlap.

The `-extend_tcheck_data_limit` option changes the hold or recovery limit in the timing checks so that the violation regions overlap by at least two units of simulation precision. The *`percent_relaxation`* argument is the maximum percentage increase allowed in the timing violation window to achieve the overlap.

You cannot use both `-extend_tcheck_data_limit` and `-extend_tcheck_reference_limit` on the command line. Using these options automatically turns on the `-ntc_warn` option.

When you use either of these options, the elaborator issues a warning message `NTCRLX` to let you know that a pair of signals had non-overlapping two limit constraints for different edges, that this situation caused non-convergence, and that the limits are being relaxed to make the constraints overlap.

Example:

```
% ncelab -extend_tcheck_data_limit 100 worklib.test:module
```

See "Negative Timing Check Limits in $setuphold and $recrem" for more information.

1.2.57 -EXTEND_TCHECK_Reference_limit percent_relaxation

(Verilog only)

Extend the violation regions established by a pair of setuphold or recrem timing checks with negative values in which the timing checks contain two different constraints for posedge and negedge of data with respect to the same reference signal and in which the violation regions do not overlap.

In situations where there are two setuphold or recrem timing checks that establish two different constraints for posedge and negedge of data with respect to the same reference signal, violation regions may not overlap. Because the violation regions created by the timing checks do not overlap, the negative timing check algorithm does not converge. This results in both of the negative limits being set to zero, thus underestimating the actual speed of the design.

You can avoid this non-convergence by hand-editing the timing checks in the HDL or in the SDF file to create some overlap, or you can use the `-extend_tcheck_data_limit` or the `-extend_tcheck_reference_limit` option to automatically extend the violation regions by the specified percentage to create the overlap.

The `-extend_tcheck_reference_limit` option changes the setup or removal limit in the timing checks so that the violation regions overlap by at least two units of simulation precision. The *`percent_relaxation`* argument is the maximum percentage increase allowed in the timing violation window to achieve the overlap.

You cannot use both `-extend_tcheck_data_limit` and `-extend_tcheck_reference_limit` on the command line. Using these options automatically turns on the `-ntc_warn` option.

When you use either of these options, the elaborator issues a warning message `NTCRLX` to let you know that a pair of signals had non-overlapping two limit constraints for different edges, that this situation caused non-convergence, and that the limits are being relaxed to make the constraints overlap.

Example:

```
% ncelab -extend_tcheck_reference_limit 100 worklib.test:module
```

See "Negative Timing Check Limits in $setuphold and $recrem" for more information.

1.2.58 -FIle arguments_filename

Use the command-line arguments contained in the specified arguments file.

You can store frequently used or lengthy command lines by putting command-line arguments (command options and top-level design unit names) in a text file. When you invoke the elaborator with the `-file` option, the arguments in the arguments file are incorporated with your command as if they had been entered on the command line.

The arguments file can contain command options, including other `-file` options, and top-level design unit names. The individual arguments within the arguments file must be separated by white space or comments.

Example:

In the following example, the file called `ncelab.args` contains `ncelab` command-line options and the name of the top-level design unit.

```
-messages
-access +rwc
```

```
worklib.top:module
```

You can invoke the elaborator with the following command:

```
% ncelab -file ncelab.args
```

You can also use the `NCELABOPTS` variable in an `hdl.var` file to include command-line options.

Note: If you are running the simulator in single-step invocation mode with *irun*, you cannot include the name of the top-level module(s) in the arguments file. In *irun*, the top-level modules are passed internally from the parser to the elaborator.

1.2.59 +FSmdebug

Generate state information in the snapshot.

This option enables the FSM debug flow in the SimVision FSM window. In order to view state machine labels and state variables in SimVision, you must use the `-access +rwc` and `+fsmdebug` options to generate state information in the snapshot.

The following example command-line uses *irun* to generate the state machine information in the snapshot and then starts SimVision with the design in the Design Browser.

```
irun -gui -access +rwc +fsmdebug *.v
```

The following example command-lines use the multi-step invocation mode to generate state machine information in the snapshot and start SimVision.

```
ncvlog *.v
ncelab -access +rwc +fsmdebug top
ncsim -gui top
```

1.2.60 -GAteloopwarn

(Verilog only)

Enable potential zero-delay gate loop warning.

This option can help to identify zero-delay gate oscillations in gate-level designs. The option sets a counter limit on continuous zero-delay loops. When the limit is reached, simulation stops and a warning is generated stating that a possible zero-delay gate oscillation was detected. You can then use the Tcl `drivers -active` command to identify the active signals and trace these signals to the zero-delay loop.

1.2.61 -GENAfile access_filename

Generate an access file that has the specified filename.

This option creates an access file based on the objects that were accessed during simulation by Tcl commands or by a PLI application and on the type of access that was required. You can then use this access file in a subsequent run by including it with the `-afile` option. See "Using -genafile to Generate an Access File" for more information.

See "Using an Access File" for information on the access file.

1.2.62 -GENEric generic_name => value

(VHDL only)

Specifies a value for a VHDL generic.

This option associates a value with a generic on the command line. You can pass a value to a generic at any level of the design hierarchy. Values can be passed across language boundaries. For example, if the design is a mixed VHDL-Verilog-VHDL design, you can assign a value to a generic of the lower-level entity.

The *value* is an appropriate value for the declared data type of the generic. To pass a value to a top-level generic, the *generic_name* is the name of the generic as it appears in the VHDL source. For example, assume that you have a top-level generic called `gen`, as in the following top-level entity:

```
entity test is
  generic (gen : integer);
  port (x : in bit;
        sum : out bit);
end test;
```

To pass a value to this top-level generic, specify the name of the generic, or the hierarchical path of the generic, on the command line.

```
% ncelab -generic "gen => 4" top_level_design_unit
```

or:

```
% ncelab -generic ":gen => 4" top_level_design_unit
```

To pass a value to a lower-level generic, provide the full hierarchical path to the generic. For example:

```
% ncelab -generic ":inst_full_add:gen => 4" top_level_design_unit
```

When passing a value across language boundaries, you can use either the Verilog hierarchy separator (.) or the VHDL separator (:). For example:

```
% ncelab -generic "vlog_top:vhdl_inst1:gen => 4" top_level_design_unit
% ncelab -generic "vlog_top.vhdl_inst1.gen => 4" top_level_design_unit
```

To pass a value to a case-sensitive generic declared with the escape character, use the escaped name in the argument. For example:

```
% ncelab -generic "\gen\ => 4" top_level_design_unit
% ncelab -generic "\GEN\ => 8" top_level_design_unit
```

Use multiple `-generic` options to specify values for multiple generics. For example:

```
% ncelab -generic "GNRC_INTEGER => -99" -generic "GNRC_TIME => 1us" \
  -generic 'GNRC_STRING => "abc"' E:A
```

The order in which the generics are specified does not matter. However, if two values for the same generic are specified, the value specified with the last `-generic` option on the command line overrides the former value.

The generic subtypes are limited to:

```
(generic G: INTEGER := ...)
(generic G: REAL := ...)
(generic G: STD_LOGIC := ...)
(generic G: STD_LOGIC_VECTOR := ...)
(generic G: BIT := ...)
(generic G: BIT_VECTOR := ...)
(generic G: TIME := ...)
(generic G: STRING := ...)
(generic G: NATURAL := ...)
(generic G: POSITIVE := ...)
(generic G: BOOLEAN := ...)
(generic G: user-defined enumerated type)
```

The default value expression is optional. The shown type marks must be used.

Integers must be decimal literals (not based literals). The literals can contain underscores, but the restrictions on leading, trailing, and double underscores are not enforced.

Time literals must be decimal physical literals. The abstract literal portion (number) must be present; the preceding comments for integers apply to TIME literals. The unit portion of the physical literal is required and can be any time unit name from `fs` through `hr`.

Strings must be delineated by double quotes. Colons are not allowed as alternate delimiters. You can embed double quotes in the string.

Values for BIT and STD_LOGIC types must be enclosed in single quotes.

The STRING, BIT_VECTOR, and STD_LOGIC_VECTOR types have constraints and directions.

- If the constraint and direction is specified in the HDL code, the size of the generic being passed with the -generic option must match the size specified in the HDL code. If the size does not match, a warning message is generated. The elaboration continues, but the generic association is ignored.

  Note: If you include the -relax option on the ncelab command line, the warning message is not generated. If the length of the value specified with the -generic option is smaller than the length specified in the HDL code, the generic value is padded with 0s for BIT_VECTOR or with Us for STD_LOGIC_VECTOR. If the length of the value specified with the -generic option is larger than the length specified in the HDL code, the generic value is truncated.

- If the constraint and direction is not specified in the HDL code, the constraint is determined from the value of the generic specified on the command line and the direction is set to TO. The left constraint is 0, and the right constraint is the length of the vector minus 1 (length - 1). The constraint for generics of type STRING is taken from the associated value).

Example

Given the following entity and architecture declarations:

```
library ieee;
use ieee.std_logic_1164.all;
library std;
use std.textio.all;

entity E is
    generic (GNRC_INTEGER : INTEGER := 0;
             GNRC_TIME: TIME := 0 ns;
             GNRC_STRING : STRING := "";
             GNRC_BOOLEAN : BOOLEAN := FALSE;
             GNRC_NATURAL : NATURAL := 10;
             GNRC_POSITIVE : POSITIVE := 5;
             GNRC_STD_LOGIC : STD_LOGIC := '1';
             GNRC_STD_LOGIC_VEC_1 : STD_LOGIC_VECTOR (0 TO 3) := "1111";
             GNRC_STD_LOGIC_VEC_2 : STD_LOGIC_VECTOR;
             GNRC_BIT : BIT := '1';
             GNRC_BIT_VEC : BIT_VECTOR := "0011";
```

```
            GNRC_REAL : REAL := 2.0);
end;

architecture A of E is
    begin
    ...
end;
```

The following `ncelab -generic` options are legal:

```
-generic "GNRC_INTEGER => 99"
-generic "GNRC_INTEGER => -99"
-generic "GNRC_TIME => 17 ns"
-generic 'GNRC_STRING => "abc"'
-generic 'GNRC_BOOLEAN => "TRUE"'
-generic "GNRC_NATURAL => 12"
-generic "GNRC_POSITIVE => 7"
-generic "GNRC_STD_LOGIC => 'U'"
-generic "GNRC_STD_LOGIC_VEC_1 => U100"
-generic "GNRC_STD_LOGIC_VEC_2 => UUUU"
-generic "GNRC_BIT => '0'"
-generic "GNRC_BIT_VEC => 1100"
-generic "GNRC_REAL => 9.0"
```

The following `-generic` options are not legal:

```
-generic "GNRC_INTEGER => 16#FF#"      -- No based literals
-generic "GNRC_TIME => 0.1 ns"         -- No real literals. Use 100 ps.
-generic "GNRC_STRING => abc"          -- No quotes around abc
-generic "GNRC_BOOLEAN => 0"           -- Value must be "true" or "false"
-generic "GNRC_BIT => 1"               -- Value must be in single quotes
-generic "GNRC_STD_LOGIC_VEC_1 => U1"  -- Ignored with a warning.Value is two bits,
                                       -- but GNRC_STD_LOGIC_VEC_1 declared as
                                       -- (0 TO 3).
```

You can include `-generic` command-line options in an arguments file that you include with the `-file` option. However, the syntax used for assigning a string value to a generic, and for assigning a value to a boolean, is different from the syntax used on the `ncelab` command line. For example, on the `ncelab` command-line, the following syntax is used:

```
-generic 'GNRC_STRING => "abcd.txt"'
-generic 'GNRC_BOOLEAN => "TRUE"'
```

If you put the `-generic` options in an arguments file, the syntax is as follows:

```
-generic "GNRC_STRING => \"abcd.txt\""
-generic "GNRC_BOOLEAN => \"TRUE\""
```

Note: You can also use the -gpg option to change the value of a generic. The -gpg option assigns a value to all VHDL generics and Verilog parameters in the design with a specified name.

1.2.63 -GENHref filename

Generate a hierarchical reference permission file (href file).

Additionally, if a design unit in a primary partition needs access to some other part of the design or if the design needs access to some shared object that is in the incremental or another primary partition, this option creates a *primary*_externs.v file for that primary.

This option (ncelab -genhref or irun -genhref) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.64 -GNoforce

Do not assign the value specified with the -gpg option to generics with the specified name if there is a default value for the generic in the VHDL source code.

You can use the -gpg option to assign a value to all VHDL generics and Verilog parameters with a specified name. For example, the following option assigns the value 10 to all generics/parameters called obj1:

```
-gpg "obj1 => 10"
```

If you include the -gnoforce option, only generics that do not have a default value will be assigned the value 10.

1.2.65 -GPg { "object_name => value" | "instance_name.object_name => value" | "hierarchical_object_path => value" }

Assign the specified value to all VHDL generics and Verilog parameters with the given name.

You can use the -generic option to assign values to VHDL generics from the command line. For Verilog parameters, you can use the -defparam option. For both of these options, you must specify the complete hierarchical path of the generic/parameter. This means that, if you want to specify a value for all objects that have the same name, you must identify all hierarchical paths and write multiple -generic and/or -defparam options.

Use the -gpg option to assign a value to all generics and parameters in the design with a specified name.

The argument to the -gpg option is in one of the following formats:

- -gpg "*object_name => value*"

  Assigns the value to all generics and parameters with the specified name.

  ```
  -gpg "obj1 => 2"
  ```

- -gpg "*instance_name.object_name => value*"

  Assigns the value to all generics and parameters with the specified name in all instances with the specified instance name.

  Note: The Verilog dot (.), the VHDL colon (:), or a slash (/) can be used interchangeably as the hierarchy separator.

  ```
  -gpg "u1.obj1 => 2"
  -gpg "dkm_i/can_counter_i/canbb => TRUE"
  -gpg "vlog_module.u1/u2:generic1 => 10"
  ```

- -gpg "*hierarchical_object_path => value*"

  Assigns the value to the generic or parameter specified by the hierarchical path.

  Note: The Verilog dot (.), the VHDL colon (:), or a slash (/) can be used interchangeably as the hierarchy separator. When using / as the separator:

  - If the top-level unit is Verilog, the path must start with /top_verilog_module.
  - If the top-level unit is VHDL, the path must start with /top_vhdl_entity.

  Examples:

  ```
  -gpg "/top_vlog/vhdl_unit/u1/gen1 => 2"
  -gpg "/top_vhdl_entity/vlog_unit/u1/obj1 => 2"
  ```

```
-gpg "/dkm_test/dkm_i/can_counter_i/canbb => TRUE"
-gpg "top_vlog.m10:gen1 => 35"
-gpg ":i1:i2:u1:obj1 => 2"
```

The *value* must be a simple constant value or a string. The *value* is interpreted as a string if it starts with a letter or if it is enclosed in quotes. Quotes are required for strings that do not start with a letter. Quoted strings must be escaped with backslash characters. For example:

```
-gpg "obj1 => 2"          Correct, integer
-gpg "obj1 => abc"        Correct, string
-gpg "obj1 => \"555\""   Correct. String does not start with a letter, so
                          quotes are required. Quoted strings must be escaped.
-gpg "obj1 => "555""      Incorrect. Quoted string is not escaped.
```

Because multiple formats can be used, there is a priority encoding built into this functionality. The more granular the specification, the higher the precedence. For example, if you use the following command line:

```
% ncelab -gpg "obj1 => 2" -gpg "u1.obj1 => 3" -gpg ":i1:i2:u1:obj1 => 4" ....
```

then:

- `:i1:i2:u1:obj1` gets a value of `4`.
- All generics/parameters named `obj1` in the instance `u1` get a value of `3`, except for `:i1:i2:u1:obj1`.
- All other generics/parameters named `obj1` get a value of `2`.

Values assigned with the `-generic` and `-defparam` options override the value assigned with `-gpg`. For example:

```
-gpg "obj1=>2"   (Value of 2 assigned to all generics and parameters called obj1.)


-gpg "obj1=>2" -generic ":i2:obj1=>9"   (Value of 2 assigned to all generics and
                                         parameters called obj1, but value of 9
                                          assigned to generic :i2:obj1.)


-gpg "obj1=>2" -defparam top.i1.obj1=9  (Value of 2 assigned to all generics and
                                         parameters called obj1, but value of 9
                                          assigned to parameter top.i1.obj1.)
```

Note: You cannot use the `-gpg` option (or the `-generic` or `-defparam` option) to change the value of a generic/parameter within protected code.

1.2.66 -GVerbose

Display messages that show the value assignments to Verilog parameters and VHDL generics from the `-gpg` option.

You can use the `-gpg` option to assign a value to all VHDL generics and Verilog parameters with a specified name. For example, the following option assigns the value `10` to all generics/parameters called `p1`:

```
-gpg "p1 => 10"
```

If you include the `-gverbose` option, messages like the following are displayed:

```
top.p1 assigned a value : 10 (-gpg)
top.m10:v1.p1 assigned a value : 10 (-gpg)
top.m32:v1.p1 assigned a value : 10 (-gpg)
```

1.2.67 -HDlvar hdlvar_pathname

Use the specified `hdl.var` file. See "The hdl.var File" for details on the `hdl.var` file.

All tools and utilities that require an `hdl.var` file use a default search mechanism to find the `hdl.var` file. See "The setup.loc File" for information on this search mechanism. Use the `-hdlvar` option to override the default search order and force the elaborator to use the specified `hdl.var` file.

Example:

```
% ncelab -hdlvar ~/hdl.var alu_16
```

You cannot include the `-hdlvar` option with the `NCELABOPTS` variable in an `hdl.var` file.

1.2.68 -HElp

Display a list of the `ncelab` command options with a brief description of each option.

This option is ignored if you include it with the `NCELABOPTS` variable in an `hdl.var` file.

1.2.69 -HRef filename

Use the specified hierarchical reference permission file.

This option (`ncelab -href` or `irun -href`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.70 -IEEe1364

(Verilog only)

Check for compatibility with the IEEE1364 standard.

Use the `-ieee1364` option to check for compatibility with the *IEEE-1364 Verilog Hardware Description Language Reference Manual*. Messages generated by the elaborator contain references to relevant sections of the IEEE-1364 LRM.

Using this option is important if you are going to use other tools, such as a second simulator or a synthesis tool, that are compatible only with a particular standard or specification.

Most compatibility checks are performed during compilation. The `-ieee1364` option should be used when you invoke *ncvlog* to compile your Verilog source files.

Example:

```
% ncelab -ieee1364 top_mod
```

1.2.71 -IEReport

(AMS)

Generates a report on interface elements.

See the description of the `-iereport` option in the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for additional information.

1.2.72 -INCRBind module_name

Identify a module whose instances shall be left unbound in the primary partition so that they can be bound to other partitions in the final model.

For example:

```
% ncelab tb_top -incrbind dut
% ncelab tb_top -primbind
```

The unbound instance `dut` in an elaborated primary snapshot is identified using the `-incrbind` option. This instance is then bound to the simulation snapshot using the above command.

This option (`ncelab -incrbind` or `irun -incrbind`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.73 -INCRPath [top_level_unit@]path

Specify the hierarchical path to the top of this primary in the final model so that hierarchical paths starting at the top of the model and referring to an object in this partition can be resolved.

Consider the following:

```
% ncelab -messages -incrpath top.prim_inst prim -mkprimary
```

This example shows that the top-level unit of the primary partition being elaborated will be instanced at `top.prim_inst` when it is connected to the incremental partition. This allows hierarchical names that begin with `top.prim_inst` to resolve correctly during primary elaboration.

```
% ncelab -mess -incrpath top1@tb.top1_inst.mid_inst.prim_inst -incrpath
top2@tb.top2_inst.mid_inst.prim_inst prim -mkprimary -access +rwc
```

This example specifies the paths of multiple top-level units during primary elaboration.

The `-incrpath` option (`ncelab -incrpath` or `irun -incrpath`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.74 -INCRTop module_name

Specify a top level of the incremental partition when using the `-genhref` option to generate a hierarchical permission file.

For example:

% irun -f dutsrc.f tb_top.v test1.v -genhref href.txt -incrtop tester

This option (`ncelab -incrtop` or `irun -incrtop`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.75 -INITBIopz

Initialize boundary inout ports to ‘Z’.

A mixed Verilog-VHDL design may contain VHDL pass-through networks (a mixed-language network with only VHDL drivers). For pass-through networks, the value of a pass-through net is resolved according to the semantics of the driving language domain. Components in the other language domain read the value through an implicit type conversion.

In VHDL, when a signal is connected to an `inout` port of a component instantiation, the `inout` port becomes one of the sources of the signal. A resolution function resolves the value of the `inout` port and the values of the other drivers of the signal to determine the value of the signal. If the `inout` port does not have drivers or a default expression, it is given the default value 'U'. According to the resolution function for `std_logic`, any value resolved with 'U' results in 'U'. This means that the value of the signal will remain at 'U'.

Use the `-initbiopz` option to initialize these VHDL language boundary inout ports to 'Z'.

See "Mixed-Language Networks and Signal Resolution" for information on signal resolution in mixed-language designs and for an example of using the `-initbiopz` option.

1.2.76 -INITBPx

In a mixed-language design, initialize all VHDL boundary objects (VHDL ports and signals connected to Verilog ports) to 'X'.

By default, VHDL objects are initialized to 'U'. Use the `-initbpx` option to initialize VHDL ports and signals that are connected to Verilog ports to 'X'.

The VHDL boundary object:

- Must be of type `std_logic` or `std_ulogic`.
- Must not have an initial value. The `-initbpx` option has no effect on a VHDL boundary object that has an initial value.

The VHDL boundary object can be of any mode (IN, OUT and INOUT), with or without a driver.

The `-initbiopz` option can be used to initialize VHDL boundary objects of INOUT mode to 'Z'. If the `-initbiopz` and `-initbpx` options are both specified, the `-initbiopz` option is ignored, and the boundary objects are initialized to 'X'.

1.2.77 -INITMEM0

Initialize all array variables to zero instead of X.

1.2.78 -INITMEM1

Initialize all bits in array variables to one instead of X.

1.2.79 -INITREG0

Initialize all non-array variables to zero instead of X.

1.2.80 -INITREG1

Initialize all bits in non-array variables to one instead of X.

1.2.81 -INTermod_path

For Verilog, enable the ability to specify unique delays and unique pulse control limits for each source-load path. See “Interconnect Delays” for details on interconnect delays.

For VHDL, enable the ability to specify unique delays for each source-load path during VITAL SDF annotation. See “VITAL SDF Annotation” for details on VITAL SDF annotation.

You cannot use the `-intermod_path` option with the `-vipdmax` or `-vipdmin` options.

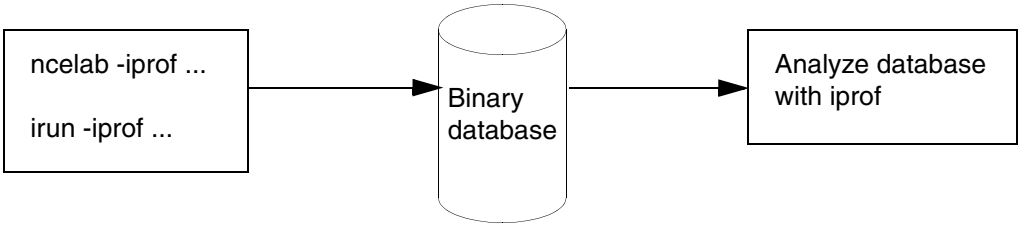
1.2.82 -IProf

Run the Advanced Profiler, a tool that measures where CPU time is spent during simulation.

The `-iprof` option (`ncelab -iprof` or `irun -iprof`) performs the required instrumentation and causes the simulator to dump a binary database containing the profile information. The binary database, called `ncprof.ucd`, is stored in the following directory:

invocation_directory`/iprof_report_dir/`*prof_cpu_nnnn*

The binary database can then be analyzed using the *iprof* analysis tool.



See the Advanced Profiler for details.

1.2.83 -LIB_Binding

(VHDL only)

Relax the strict default binding search order.

By default, the elaborator adheres to a strict interpretation of the VHDL LRM, which states that you must use `LIBRARY` statements with corresponding `USE` clauses in the source code to provide visibility to the declarative region that an unbound instance resides in. To bind component instances to compiled design units in the libraries, the elaborator:

1. Uses explicit binding indications.
2. If there is no explicit binding indication, the elaborator tries to bind the component to (in order):
   a. A design unit made visible with a `USE` clause given to the architecture instantiating the component.
   b. A design unit made visible with a `USE` clause given to the entity of the architecture instantiating the component.
   c. A design unit available in the library into which the component was compiled. For example, if you have the following instantiation statement:

      ```
      inst1 : DUT port map (......)
      ```

      and the component `DUT` was compiled into library `LIB_COMP`, the elaborator will search for entity `DUT` in the library `LIB_COMP`.
   d. A design unit in the work library.

If a binding cannot be found, the elaborator generates an error.

The `-lib_binding` option extends the set of binding rules followed when a component is being instantiated using default binding. The search order used with the `-lib_binding` option is as follows:

1. A design unit made visible with a `USE` clause given to the architecture instantiating the component.
2. A design unit made visible with a `USE` clause given to the entity of the architecture instantiating the component.
3. A design unit available in the library into which the component was compiled.
4. A design unit in the work library.
5. A design unit made visible with a `LIBRARY` clause given to the architecture instantiating the component (no corresponding `USE` clause).
6. A design unit made visible with a `LIBRARY` clause given to the entity of the architecture instantiating the component (no corresponding `USE` clause).

Note: The `ncelab -relax` option can also be used to relax the strict default binding search order. The search order used with `-relax` is the same as that used with `-lib_binding`, except that `-relax` will also search for a design unit in a library defined in the `cds.lib` file. That is, if a binding has not been found, the elaborator opens the `cds.lib` file and searches all of the libraries that are defined in the file that have not already been searched. The `-relax` option also enables a looser interpretation of other VHDL rules specified in the LRM besides the default binding order.

1.2.84 -LIBMap library_map_file [library_map_file ...]

(Verilog only)

Use the specified Verilog library mapping file(s).

A library map file can contain:

- Library declarations

  A library declaration associates a logical library name with a source file or set of source files. The specified source files are compiled into the associated library. For example:

  ```
  library rtlLib "*.v";
  library gateLib "adder.vg";
  ```

  In this example, all design units in files with a `.v` extension will be compiled into the library called `rtlLib`, and design units in the file called `adder.vg` will be compiled into the library called `gateLib`.

Because the simulator has always provided several ways to control the compilation of design units into libraries, these library declarations are optional. If used, they provide an additional way to control which libraries design units are compiled into.

If you use a library map file with library declarations to control the compilation of design units into libraries, you must specify the `-libmap` option when the source files are compiled (`ncvlog -libmap` or `irun -libmap`). You must also specify the `-libmap` option at elaboration time (`ncelab -libmap`) because the elaborator will search the libraries in the order they are specified in the library declarations.

- References to other library map files

  An `include` statement can be used to include the contents of a library map file in another library map file.

- Configuration declarations

  These are configuration blocks that control the binding of instances during elaboration. If you specify the Verilog configuration in a library map file, you must specify the library map file with the `-libmap` option. In the following example command line, `lib.map` is the name of the library map file that contains the configuration rules, and `cfg` is the name of the configuration:

  ```
  % ncelab -libmap lib.map [other_options] cfg
  ```

If you are using *irun*, and want to use a configuration that is in a library map file, you must use the `-libmap` option to specify the library map file. You must use the `-top` option to specify the top-level unit. For example:

```
% irun -libmap lib.map -top cfg [other_options] source_files
```

See "Using a Verilog Configuration" for details on Verilog configurations.

See the description of the `-use5x4vlog` option for information on using 5.X configurations with library map files.

1.2.85 -LIBName library_name

(Verilog only)

Search the specified library first, instead of searching the libraries in the order listed in the library declarations in a library map file.

A library map file can contain either or both of the following:

- Library declarations, which associate a source file, or set of source files, with a library logical name. For example:

```
library rtlLib  "top.v";
library aLib    "adder.v";
library gateLib "adder.vg";
```

- A configuration, which contains the explicit rules to be used for binding instances.

If the library map file does not contain a configuration, and if you use the `-libmap` option to specify the name of the library map file, the libraries declared in the library declarations are searched to find a binding. By default, the libraries are searched in the order in which they are declared. Use the `-libname` option to override the default library search order.

For example, suppose that you have a library map file that contains the library declarations shown above. By default the libraries are searched in the following order: `rtlLib`, `aLib`, `gateLib`. To override this order so that `gateLib` is searched first, use the following command:

```
% ncelab -libmap library_map_file -libname gateLib top_level_unit
```

If you are running the simulator in single-step invocation mode with *irun*, use the `-libname` *`library_name`* option. You must also include the `-top` *`top_level_unit`* option to specify the top-level unit. This can be the top-level module if you are not using a configuration, or the name of a configuration.

See "Default Configuration Using a Library Map File" for more information and an example.

1.2.86 -LIBVerbose

Display messages during design unit binding.

1.2.87 -LICqueue

Queue the request for a license if one is not currently available and run the elaborator when a license becomes available.

Enabling Polling for Available Licenses on Multiple Servers

When a job is submitted, the servers listed in the `CDS_LIC_FILE` variable (or the `LM_LICENSE_FILE` variable) are searched in order for an available license. If a license is not currently available, and if the job was submitted with the `-licqueue` option (`ncelab -licqueue` or `irun -licqueue`), the elaborator queues the request. The request is queued to the first server in the list that contains the feature, and the request is bound to this server. If a license on another server is released and is then available, the elaborator will not check out that license.

Set the CDS_LIC_QUEUE_POLL environment variable to 1 to enable polling of the other servers in the search path for an available license.

```
setenv CDS_LIC_QUEUE_POLL 1
```

While the request is queued to the first server in the path that contains the feature, the other servers will be polled for a license. If a license that satisfies the queued request becomes available, that license is checked out. The queued license request is then dequeued.

If server polling has been enabled, you can set a second environment variable, CDS_LIC_QUEUE_POLL_INT, to set the polling interval. The default (and minimum) is 30 seconds. You can set the interval to a higher number of seconds. If you set the variable to a value less than 30, the default is used.

```
setenv CDS_LIC_QUEUE_POLL_INT 40
```

When polling is enabled, it is possible that a queued license is checked out at the same time that another license is checked out due to the polling. In this case, the license from the queued request is checked in, and the license checked out from the polling is used. This maximizes license usage because non-polling requests still queue on the first server when users have the same CDS_LIC_FILE.

Enabling polling of multiple servers has a performance impact, as servers must be checked for available licenses, licenses must be checked out if the polling is successful, and queued requests must be dequeued. Performance also depends on the number of queued requests and the number and location of servers in the path.

Additional logging is performed when polling is enabled as the attempted checkouts (both failed and successful) and dequeuing of the queued license is logged.

Polling is disabled if a job is suspended. If the job is then resumed, all servers are checked for an available license. If no license is available, the request is queued to the first server in the list that contains the feature.

1.2.88 -LOADPli1shared_lib_name:boot_func_name [:export][,boot_func_name ...]

(Verilog only)

Dynamically load the specified PLI1.0 application.

If a PLI application has already been compiled into a dynamic shared library, you can use -loadpli1 to load the library and to register the system tasks defined in the application at run time.

The argument to this option is the name or full path of the shared library that contains the PLI application followed by the name of the function that registers the new system tasks. This function, called the *bootstrap* function, is part of the PLI application, and is defined in the shared library.

You can load any number of applications in the same statement by separating the names of the bootstrap functions with a comma. No spaces are allowed in the argument.

The file extension of the shared library is optional. The elaborator appends a suffix that is consistent with the OS that you are running. For example, if you are running on the SUN4v platform and enter the following command, the elaborator searches for a library called `SSI.so`.

```
% ncelab -loadpli1 SSI:ssi_boot top
```

For PLI1.0 applications, the simulator always loads the shared library and executes any bootstrap function(s) that is passed to the elaborator.

Examples:

1. `% ncelab -loadpli1 mylib:boot1`

   The elaborator loads the shared library and executes `boot1`. The simulator loads the shared library and executes `boot1`.

2. `% ncelab -loadpli1 mylib:boot1,boot2`

   The elaborator loads the shared library and executes `boot1` and `boot2`. The simulator loads the shared library and executes `boot1` and `boot2`.

In some cases, you may want to execute different functions in the elaborator and in the simulator. For example, you might have a PLI application that you want to run at simulation time only. Such an application may perform tasks such as recording how long a simulation runs or opening communication with another tool. You can do this by using a period to separate the function that you want to execute in the elaborator from the function that you want to execute in the simulator. For example:

3. `% ncelab -loadpli1 mylib:boot1.boot2`

   In this example, the argument contains one *`boot_func_name`*, which is divided into two parts. The elaborator loads `mylib` and executes `boot1`. The simulator loads `mylib` and executes `boot2`.

   Note: `boot2` must register the same system tasks that are registered by `boot1`.

4. `% ncelab -loadpli1 mylib:boot1,boot2.boot3,boot4`

In this example, the argument contains three boot functions. The second is divided into two parts. The elaborator loads `mylib` and executes `boot1`, `boot2`, and `boot4`.

The simulator loads `mylib` and executes `boot1`, `boot3`, and `boot4`.

In the IUS 5.4 and earlier releases, symbols in dynamic libraries were exported by default. Because this sometimes resulted in symbol collisions if the same symbols were defined in multiple applications, this default export of symbols has been turned off. To enable the export of symbols from dynamic libraries, you can add the `:export` qualifier to the command-line option. For example, suppose that a dynamic library called `libddr.so` has a dependency on a dynamic library called `libdigeo.so`. Use the following command-line option:

```
% ncelab -loadpli1 libdigeo:digeo_boot:export -loadpli1 libddr:ddr_boot
top_level_design_unit
```

Note: Instead of adding the `:export` qualifier, you can use the `-pli_export` option.

1.2.89 -LOADSc library_name

(NC-SC only)

Dynamically load the specified SystemC® library.

See “Elaborating SystemC Designs” in the chapter called “Simulating SystemC Models” in the *SystemC Simulation User Guide* for details on elaborating designs containing SystemC models.

1.2.90 -LOADVpishared_lib_name:boot_func_name [:export][,boot_func_name ...]

(Verilog only)

Dynamically load the specified VPI application.

If a VPI application has already been compiled into a dynamic shared library, you can use `-loadvpi` to load the library and to register the system tasks and VPI callbacks defined in the application at run time.

The argument to this option is the name or full path of the shared library that contains the VPI application followed by the name of the function that returns a pointer to either a `vpi_register_systf()` or a `vpi_register_cb()` function call that contains the definitions of system tasks and functions. This function, called the *bootstrap* function, is part of the VPI application, and is defined in the shared library.

You can load any number of applications in the same statement by separating the names of the bootstrap functions with a comma. No spaces are allowed in the argument.

The file extension of the shared library is optional. The elaborator appends a suffix that is consistent with the OS that you are running. For example, if you are running on the SUN4v platform and enter the following command, the elaborator searches for a library called `SSI.so`.

```
% ncelab -loadvpi SSI:ssi_boot top
```

Examples:

1. `% ncelab -loadvpi mylib:boot1`

   The elaborator loads the shared library and executes `boot1`. The simulator loads the shared library and executes `boot1`.

2. `% ncelab -loadvpi mylib:boot1,boot2`

   The elaborator loads the shared library and executes `boot1` and `boot2`. The simulator loads the shared library and executes `boot1` and `boot2`.

In some cases, you may want to execute different functions in the elaborator and in the simulator. For example, you might have a PLI application that you want to run at simulation time only. Such an application may perform tasks such as recording how long a simulation runs or opening communication with another tool. You can do this by using a period to separate the function that you want to execute in the elaborator from the function that you want to execute in the simulator. For example:

3. `% ncelab -loadvpi mylib:boot1.boot2`

   In this example, the argument contains one *`boot_func_name`*, which is divided into two parts. The elaborator loads `mylib` and executes `boot1`. The simulator loads `mylib` and executes `boot2`.

   Note: `boot2` must register the same system tasks that are registered by `boot1`.

4. `% ncelab -loadvpi mylib:boot1,boot2.boot3,boot4`

   In this example, the argument contains three *`boot_func_names`*. The second is divided into two parts. The elaborator loads `mylib` and executes `boot1`, `boot2`, and `boot4`.

   The simulator loads `mylib` and executes `boot1`, `boot3`, and `boot4`.

If your VPI application does not contain user-defined system tasks or functions and you use other VPI callbacks (that is, `vpi_register_cb` instead of `vpi_register_systf`), the

code only needs to be run at simulation time. In this case, you can use the `-loadvpi` option on the `ncsim` command line as follows:

```
% ncsim -loadvpi mylib:boot1
```

In the IUS 5.4 and earlier releases, symbols in dynamic libraries were exported by default. Because this sometimes resulted in symbol collisions if the same symbols were defined in multiple applications, this default export of symbols has been turned off. To enable the export of symbols from dynamic libraries, you can add the `:export` qualifier to the command-line option. For example, suppose that a dynamic library called `libddr.so` has a dependency on a dynamic library called `libdigeo.so`. Use the following command-line option:

```
% ncelab -loadvpi libdigeo:digeo_boot:export -loadvpi libddr:ddr_boot
top_level_design_unit
```

Note: Instead of adding the `:export` qualifier, you can use the `-pli_export` option.

1.2.91 -LOCalbind

Restrict the binding of design units to the IP level specified with the `-renameip` option when a package is installed with the *ncrelocate* utility.

When you install a package with *ncrelocate*, you can automatically modify all logical library names while installing the shipped package by using a `ncrelocate -install -renameip` command. You can include the `-localbind` option on the `ncrelocate` command to restrict the binding of design units to the IP level specified with the `-renameip` option.

The `-localbind` option must be included on the `ncrelocate -install` command line, as well as on the `ncelab` command line (`ncelab -localbind` or `irun -localbind`) when the integrated system is elaborated.

See ncrelocate for details on using the *ncrelocate* utility.

1.2.92 -LOGfile filename

Use the specified name for the log file instead of the default name `ncelab.log`.

Example:

```
% ncelab -logfile mylog.log top
```

Use `-nolog` if you do not want a log file. If you use both `-logfile` and `-nolog` on the command line, `-logfile` overrides `-nolog`.

Use `-append_log` if you are going to run *ncelab* multiple times and you want all log information appended to one log file. If you do not use this option, the log file is overwritten each time you run *ncelab*.

Because the log file is opened before variables in the `hdl.var` file are read, the `-logfile` option is ignored with a warning if you define it with the `NCELABOPTS` variable in an `hdl.var` file.

Redirecting the output of an NC tool to the same log file that the tool creates can result in corrupted information. For example, with the following command, *ncelab* generates `ncelab.log` and then the output is redirected to the same file.

```
% ncelab -messages worklib.top:arch > ncelab.log
```

Instead of using redirection, use the `-logfile` option to specify where the output is to be recorded. If file redirection is absolutely needed, include the `-nolog` option to suppress the generation of the log file that the tool would normally create.

1.2.93 -LPS_Assign_ft_buf

Specify that a continuous assignment is to be treated as a buffer.

See the description of `-lps_assign_ft_buf` in the *Low-Power Simulation User Guide* for details on this option.

1.2.94 -LPS_Blackboxmm

Treat all macro models as a black box.

See the description of `-lps_blackboxmm` option in the *Low-Power Simulation User Guide* for details on this option.

1.2.95 -LPS_CEllrtn_off

Suppress implicit state retention insertion from primitive cells.

See the description of `-lps_cellrtn_off` in the *Low-Power Simulation User Guide* for details on this option.

1.2.96 -LPS_COnst_aon

Disable corruption for any tie-high or tie-low constant in a switchable domain, if that constant drives the load in an always-on domain.

See the description of `-lps_const_aon` in the *Low-Power Simulation User Guide* for details on this option.

1.2.97 -LPS_CPf cpf_filename

Specify a CPF file for low-power simulation.

See the description of `-lps_cpf` in the *Low-Power Simulation User Guide* for details on this option.

1.2.98 -LPS_Dtrn_min

Treat the value specified for the transition slope or transition latency with an `update_power_domain` command as the minimum transition time.

See the description of `-lps_dtrn_min` in the *Low-Power Simulation User Guide* for details on this option.

1.2.99 -LPS_Force_reapply

Reapply forces at power up that you applied to a signal affected by a power domain that was being corrupted.

See the description of `-lps_force_reapply` in the *Low-Power Simulation User Guide* for details on this option.

1.2.100 -LPS_IMPLICITPSO_char 'value'

When power is shut off, corrupt objects of VHDL user-defined enumerated types that have character literals specified in the definition with the character specified by the *`value`* argument.

See the description of `-lps_implicitpso_char` in the *Low-Power Simulation User Guide* for details on this option.

1.2.101 -LPS_IMPLICITPSO_nonchar value

When power is shut off, corrupt objects of VHDL user-defined enumerated types that have identifiers specified in the definition with the identifier specified by the *value* argument.

See the description of -lps implicitpso nonchar in the *Low-Power Simulation User Guide* for details on this option.

1.2.102 -LPS_INT_index_nocorrupt

Disable the corruption of VHDL integers that are used as an array index.

See the description of -lps int index nocorrupt in the *Low-Power Simulation User Guide* for more information.

1.2.103 -LPS_INT_nocorrupt

Disable the corruption of all VHDL integers in the design.

See the description of -lps int nocorrupt in the *Low-Power Simulation User Guide* for details on this option.

1.2.104 -LPS_ISO_Off

Turn off port isolation in low-power simulation.

See the description of -lps iso off in the *Low-Power Simulation User Guide* for details on this option.

1.2.105 -LPS_ISO_Verbose

Enable reporting of isolation information.

See the description of -lps iso verbose in the *Low-Power Simulation User Guide* for details on this option.

1.2.106 -LPS_ISOFilter_verbose

Enable reporting of isolation filtering information.

See the description of `-lps isofilter verbose` in the *Low-Power Simulation User Guide* for details on this option.

1.2.107 -LPS_ISORuleopt_warn

Print a warning message if an isolation rule has been optimized away.

See the description of `-lps isoruleopt warn` in the *Low-Power Simulation User Guide* for details on this option.

1.2.108 -LPS_LOG_verbose filename

Write the output of the `-lps_verbose` option to a log file with the specified name. You must use the `-lps_verbose` option with `-lps_log_verbose`.

See the description of `-lps_log_verbose` in the *Low-Power Simulation User Guide* for details on this option.

1.2.109 -LPS_LOGfile filename

Write the low-power simulation information to a log file with the specified name.

See the description of `-lps_logfile` in the *Low-Power Simulation User Guide* for details on this option.

1.2.110 -LPS_MOdules_wildcard

Enable the use of wildcard characters in the *`module_list`* argument of the CPF `set_sim_control -modules` option.

See the description of the `set_sim_control` command in the *Low-Power Simulation User Guide* for details.

1.2.111 -LPS_MTr_min

Treat the value specified for the transition time with a `create_mode_transition -latency` option as the minimum latency.

See the description of `-lps mtrn min` in the *Low-Power Simulation User Guide* for details on this option.

1.2.112 -LPS_MVs

Turn on voltage scaling simulation and voltage tracking.

See the description of `-lps_mvs` in the *Low-Power Simulation User Guide* for details on this option.

1.2.113 -LPS_NO_xzshutoff

Ignore an unknown value on the power domain shutoff control signal.

See the description of `-lps_no_xzshutoff` in the *Low-Power Simulation User Guide* for details on this option.

1.2.114 -LPS_NOtlp

Turn off special treatment for top-level ports.

See the description of `-lps_notlp` in the *Low-Power Simulation User Guide* for details on this option.

1.2.115 -LPS_PA_model_on

(Verilog only)

Turn on power-aware functionality.

See the description of `-lps pa model on` in the *Low-Power Simulation User Guide* for details on this option.

1.2.116 -LPS_PMCheck_only

Use the domain-level controls (active state conditions specified with `create_power_domain -active_state_conditions`) to drive the power domain nominal condition transitions. Power mode/mode transition specifications are used for verification, not control.

See the description of `-lps pmcheck only` in the *Low-Power Simulation User Guide* for details on this option.

1.2.117 -LPS_PMOde

Turn on power mode simulation, mode tracking, and built-in mode checking, in addition to voltage tracking.

See the description of `-lps pmode` in the *Low-Power Simulation User Guide* for details on this option.

1.2.118 -LPS_PSn_verbose

Enable reporting of information on the power supply network. This includes information on supply sets, power switches, and power domains.

See the description of `-lps_psn_verbose` in the *Low-Power Simulation Guide (IEEE 1801*) for details on this option.

1.2.119 -LPS_RTN_Lock

Lock the value of state retention elements so that the value does not change between:

- The save operation and power down
- Power up and state restoration

See the description of `-lps rtn lock` in the *Low-Power Simulation User Guide* for details on this option.

1.2.120 -LPS_RTN_Off

Turn off state retention in low-power simulation.

See the description of `-lps rtn off` in the *Low-Power Simulation User Guide* for details on this option.

1.2.121 -LPS_SImctrl_on

Enable simulation-time control over low-power simulation. This option lets you elaborate the design with power and then control the power simulation without re-elaborating the design.

See the description of `-lps_simctrl_on` in the *Low-Power Simulation User Guide* for details on this option.

1.2.122 -LPS_SRFilter_verbose

Enable reporting of state retention filtering information.

See the description of `-lps_srfilter_verbose` in the *Low-Power Simulation User Guide* for details on this option.

1.2.123 -LPS_SRRuleopt_warn

Print a warning message if a state retention rule has been optimized away.

See the description of `-lps_srruleopt_warn` in the *Low-Power Simulation User Guide* for details on this option.

1.2.124 -LPS_STDby_nowarn

Suppress the warning messages that are generated when a transition occurs at an input of a power domain that is in standby mode.

See the description of `-lps stdby nowarn` in the *Low-Power Simulation User Guide* for details on this option.

1.2.125 -LPS_STIme time

Specify the time to start low-power simulation.

See the description of `-lps stime` in the *Low-Power Simulation User Guide* for details on this option.

1.2.126 -LPS_STL_off

Turn off state loss in low-power simulation.

See the description of `-lps stl off` in the *Low-Power Simulation User Guide* for details on this option.

1.2.127 -LPS_Upcase

Convert all identifiers in the CPF file to uppercase.

See the description of `-lps upcase` in the *Low-Power Simulation User Guide* for details on this option.

1.2.128 -LPS_VERBose {1 | 2 | 3 | 4}

Enable reporting of power information and specify the level of information you want reported.

You must use this option to log power domain information. If this option is not specified, the simulator does not report power domain information.

See the description of `-lps_verbose` in the *Low-Power Simulation User Guide* for details on this option.

1.2.129 -LPS_VERIfy

Note: This option is currently supported only for RTL simulations.

Enable the following advanced low-power verification features:

- Automatic generation of assertions that check properties of control signals and their correct sequencing
- Automatic generation of a SystemVerilog coverage model for low-power control signals

See "Advanced Low-Power Verification Features" in the *Low-Power Simulation User Guide* for details on this option.

1.2.130 -LPS_VPlan vplan_filename

Generate a verification plan (vPlan) for power coverage for use by vManager. The generated vPlan provides a more organized, easier to read, and better documented view of the automated coverage data generated by the simulator.

See "Advanced Low-Power Verification Features" in the *Low-Power Simulation User Guide* for details on this option.

1.2.131 -MAxdelays

Apply the maximum delay value from a timing triplet in the form min:typ:max that appears in a specify block in the Verilog description.

This option also selects the maximum delay value if the min:typ:max value appears in the SDF file while annotating to Verilog or to VITAL unless an SDF-specific construct is used to override it. For example, if you use `-maxdelays` on the command line, but specify MINIMUM in an SDF command file (`MTM_CONTROL = "MINIMUM"`), in an SDF configuration file (`MTM = MINIMUM;`), or in the `$sdf_annotate` task, the maximum values in the specify block will be used, but the minimum values in the SDF file will be used.

Example:

```
% ncelab -maxdelays top_mod
```

1.2.132 -MEMdetail

Enable instrumentation for stream profiling in the simulator.

Cadence provides a memory profiler that reports on memory usage based on native streams. The report generated by the profiler is a collection of the following profiles:

- Memory usage based on native streams
- Heap usage of SystemVerilog dynamic objects
- Details of memory used by shared objects

The profiler report is contained in an output file called `ncprofmem.out`.

The profiler keeps track of all memory allocations and de-allocations and reports them in tabular form with details such as: *memory in use*, *total memory allocated*, *total memory de-allocated*, *filename* and *line numbers*.

Note: The memory profiler is supported on Linux platforms only.

To enable stream based profiling, the `-memdetail` option must be used on the `ncelab` command when you elaborate the design. You must also include the `-memdetail` option on the `ncsim` command to generate the profile report.

```
% ncelab -memdetail [other_options] top_level_design_unit
% ncsim -memdetail [other_options] snapshot_name
```

If you are using single-step invocation mode with *irun*, the `-memdetail` option is automatically passed to both the elaborator and the simulator.

```
% irun -memdetail [other_options] source_files
```

See the SystemVerilog Reference for details on the memory profiler.

1.2.133 -MEssages

Print informative messages during execution, including a list of command-line options used in the run. During elaboration, the messages also provide some statistical information about the design hierarchy.

If you are running in single-step mode with *irun*, messages are displayed by default. There is no `irun -messages` option.

By default, elaborator messages are printed to a log file called `ncelab.log` (or to `irun.log` if you are using *irun*). Use `-logfile` to rename the log file. Use `-nolog` if you do not want a log file.

Messages are also printed to the screen by default, except for the command-line options. Use `-nostdout` if you want to suppress printing to the screen.

For Verilog portions of the design, you can use the `-libverbose` option to display additional messages that provide information on the binding of instances to compiled design units.

1.2.134 -MINdelays

Apply the minimum delay value from a timing triplet in the form min:typ:max that appears in a specify block in the Verilog description.

This option also selects the minimum delay value if the min:typ:max value appears in the SDF file while annotating to Verilog or to VITAL, unless an SDF-specific construct is used to override it. For example, if you use `-mindelays` on the command line, but specify MAXIMUM in an SDF command file (`MTM_CONTROL = "MAXIMUM"`), in an SDF configuration file (`MTM = MAXIMUM;`), or in the `$sdf_annotate` task, the minimum values in the specify block will be used, but the maximum values in the SDF file will be used.

Example:

```
% ncelab -mindelays top_mod
```

1.2.135 -MIXesc

Elaborate mixed-language designs in which escaped names are used for VHDL entities, port names, or generics.

The `-mixesc` option is required when elaborating a mixed-language design if you have used escaped names for VHDL entities that are instantiated in a Verilog module, or if you have used escaped names for VHDL ports or generics.

Note: Cadence strongly recommends that you avoid using escaped names for VHDL entities, ports, or generics. See "Importing VHDL into Verilog" for more information.

1.2.136 -MKprimsnap

Create a primary snapshot.

This option (`ncelab -mkprimsnap` or `irun -mkprimsnap`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.137 -MODELIncdir pathname [:pathname]

(AMS)

Specifies a list of directories to be searched for model files, included files, or files that are passed as instance parameter values.

See the description of the `-modelincdir` option in the chapter "Elaborating" in the *Virtuoso AMS Designer Simulator User Guide* for details on this option.

1.2.138 -MODELPath argument

(AMS)

Specifies SPICE or Spectre source files for the models to be used in a specified scope and in scopes below the specified scope. If a source file is a library file (which begins with the `library` keyword), you must specify a *`section`*.

See the description of the `-modelpath` option in the chapter "Elaborating" in the *Virtuoso AMS Designer Simulator User Guide* for details on this option.

1.2.139 -NAmemap_mixgen

Use name mapping when mapping from VHDL component generics to Verilog parameters.

By default, when a Verilog module is instantiated inside a VHDL design unit and default binding is done, VHDL generics are mapped to Verilog parameters using positional mapping. For example, suppose that the Verilog module contains the following two parameters:

```
parameter abc = 1;
parameter xyz = 3;
```

The VHDL component declaration declares the following two generics:

```
component mymod
  generic (xyz, abc : integer);
end component
```

The Verilog module is instantiated as follows:

```
i1 : mymod generic map(xyz => 5, abc =>9);
```

A component instantiation statement with a generic map associates actual values with the local generics of the component. As the component is VHDL, named association is done. In this example, component generic `abc` gets the value `9`, and component generic `xyz` gets the value `5`.

While binding, the local values of the component get associated with the parameters of the module using positional mapping. Therefore, in this example, the VHDL component generic `xyz` (value=5) is mapped to the Verilog parameter `abc`. The VHDL component generic `abc` (value=9) is mapped to the Verilog parameter `xyz`.

Use the `-namemap_mixgen` option if you want to use name mapping instead of positional mapping when mapping VHDL component generics to Verilog parameters. If the example above is elaborated with this option, the VHDL component generic `abc` (value=9) is mapped to the Verilog parameter `abc`, and the VHDL component generic `xyz` (value=5) is mapped to the Verilog parameter `xyz`.

1.2.140 -NCError warning_code[:warning_code ...]

Increase the severity level of the specified warning message from warning to error. The *`warning_code`* argument is the message code (mnemonic) that appears in the warning message following the severity code. You can enter the *`warning_code`* in uppercase or in lowercase.

Example:

By default, the elaborator issues the following warning message if you have a `$sdf_annotate` system task in your Verilog HDL, but the SDF file cannot be found:

```
ncelab: *W,CUSFNF: The SDF file "test.sdf" not found..
```

If you want to upgrade this warning message to an error message so that the elaborator will stop, use the following option on the command line:

```
% ncelab -ncerror CUSFNF worklib.top:module
```

You can increase the severity level of multiple warning messages either by using multiple `-ncerror` options or by using one `-ncerror` option and separating the *`warning_code`* arguments with a colon. For example,

```
% ncelab -ncerror INTOVF -ncerror CUVWSP worklib.top:module
% ncelab -ncerror INTOVF:CUVWSP worklib.top:module
```

Using this option can change the behavior of the tool because functions that return errors instead of warnings may behave differently. Warnings that are changed to errors are counted in the error count limit that you specify with the `-errormax` option.

1.2.141 -NCFatal {warning_code | error_code} [:{warning_code | error_code} ...]

Increase the severity level of the specified warning message or error message from warning or error to fatal. The *`warning_code`* or *`error_code`* argument is the message code (mnemonic) that appears in the message following the severity code.

Example:

```
% ncelab -ncfatal LMNOPQ worklib.top:module
```

You can increase the severity level of multiple warning messages or error messages to fatal either by using multiple `-ncfatal` options or by using one `-ncfatal` option and separating the *`warning_code`* or *`error_code`* arguments with a colon. For example,

```
% ncelab -ncfatal DLCPTH -ncfatal CUVWSP worklib.top:module
% ncelab -ncfatal DLCPTH:CUVWSP worklib.top:module
```

1.2.142 -NCInitialize

(Verilog only)

Enable the initialization of all Verilog variables to a specified value at the start of simulation.

Note: A *variable* is an abstraction of a data storage element (see Section 3.2.2 of the IEEE 1364-2001 standard). This includes `reg`, `integer`, `time`, `real`, `realtime`, and memories.

It also includes the new variable data types introduced by SystemVerilog, such as `logic`, `bit`, `byte`, `int`, `shortint`, `longint`, structs, enums, and arrays of variables (see Section 27.14 of the IEEE 1800 SystemVerilog LRM).

This option provides a convenient way to initialize Verilog variables in the design instead of writing code in an `initial` block, using Tcl `deposit` commands at time zero, or writing a VPI application to do the initialization.

The elaborator (`ncelab -ncinitialize`) option enables initialization of Verilog variables. The value that the variables are to be initialized to is specified with the `-ncinitialize` option when you invoke the simulator (`ncsim -ncinitialize`). This lets you create a single snapshot that can be used in different simulation runs with or without initialization of variables.

If you are running in single-step invocation mode with *irun*, use the simulator `-ncinitialize` *`value`* option. Using this option on the `irun` command line will automatically enable initialization of variables by passing `-ncinitialize` to the elaborator.

For example:

```
% ncvlog test.v
% ncelab -ncinitialize worklib.top   // Enable initialization
% ncsim -ncinitialize 0 worklib top   // Initialize all variables in the design to 0
```

Or:

```
% irun -ncinitialize 0 test.v
```

When you invoke the simulator:

- All variables can be initialized to `0`, `1`, `x`, or `z`. For example:

  ```
  ncsim -ncinitialize 0
  ```

- Different variables can be initialized to `0`, `1`, `x`, or `z` randomly using `rand:`*`n`*, where *`n`* is a 32-bit integer used as a randomization seed. For example:

  ```
  ncsim -ncinitialize rand:56
  ```

- Different variables can be initialized to 0 or 1 randomly using `rand_2state:`*`n`*. For example:

  ```
  ncsim -ncinitialize rand_2state:56
  ```

Note: You can also enable initialization by specifying global write access to all simulation objects with the `-access +w` option. However, this option provides both read and write access to all simulation objects in the design, which can affect performance. The `-ncinitialize` option provides read and write access only to Verilog variables.

1.2.143 -NEGDelay

Enable adjustment of negative interconnect and module delays.

By default, the elaborator will zero out all negative delays and issue a warning. Use this option (`ncelab -negdelay` or `irun -negdelay`) when you want to support negative delay adjustments to single-level driver and loads.

For more details, see Negative SDF Delays.

1.2.144 -NEG_Verbose

Print detailed information on all negative delay adjustments made during elaboration.

By default, negative delay adjustments are printed to the log file (`ncelab.log` or `irun.log`). Use `-logfile` to rename the logfile. Use `-nolog` if you do not want a log file.

Negative delay adjustments are also printed to the screen by default. Use `-nostdout` if you want to suppress printing to the screen.

1.2.145 -NEVerwarn

Disable printing of all warning messages.

```
% ncelab -neverwarn worklib.top
```

To turn off one or more specific warning messages, use `-nowarn`.

1.2.146 -NOASsert

Disable PSL assertions in the snapshot.

During development, the individual blocks that comprise a chip are probably simulated with assertion checking turned on. When the compiled blocks are integrated at the chip level, assertion checking is no longer required.

Use the `-noassert` option to turn off assertion checking in the elaborator. This option disables all assertion properties in the snapshot.

See *Assertion Checking in Simulation*, Chapter 2, "Compiling and Elaborating a Design with Assertions" for details.

1.2.147 -NOAUtosdf

(Verilog only)

Do not perform automatic SDF annotation.

The elaborator recognizes `$sdf_annotate` system tasks in the design source files, and if the `$sdf_annotate` system tasks are scheduled to run at time 0 and meet other requirements, annotation is performed automatically. Use the `-noautosdf` option if you do not want to annotate the design.

See "SDF Timing Annotation" for details on SDF annotation.

1.2.148 -NOBinding design_unit_name

Exclude instances of the specified design unit when elaborating the design.

Note: The `-nobinding` option can be used only with the `-partialdesign` option.

If your design contains design units that are simulation-specific or that are not suitable for synthesis and/or formal verification, you can use this option to exclude the design units from the elaboration. When elaborating the design hierarchy, instances of the specified design unit are not bound, and the elaborator generates a snapshot of the partial design. You can then perform formal verification (connectivity checks, for example) on the parts of the design for which it makes sense.

For example, the following command specifies that no binding should be done for any instance of design unit `foo`.

```
% ncelab -partialdesign -nobinding foo worklib.top:module
```

Use multiple `-nobinding` options to exclude multiple design units.

1.2.149 -NOCopyright

Suppress the printing of the copyright banner.

Because the copyright banner is displayed before any variables in the `hdl.var` file are processed, this option is ignored if you include it with the `NCELABOPTS` variable in an `hdl.var` file.

1.2.150 -NODEAdcode

Turn off the dead code optimization.

The simulator includes an optimization that prevents code that does not contribute in any way to the output of the model from running. The `-nodeadcode` option turns off this optimization.

Use this option if you want to be sure to exercise all the code that is in the model. There may be cases where you want to test that all your code executes without error, even though it does not affect any simulation output. This might happen when you don't have the design completed yet.

Because this "dead" code does not run, any run-time errors, such as constraint errors or null access dereferences, that would be generated by the code are not generated. Other simulation differences (for example, with delta cycle counts and active time points) can also occur.

The `-nodeadcode` option can affect the time that simulation terminates. A process that is deadcoded might otherwise keep simulation alive and running for longer, even though it generates no visible effect on simulation results.

Using `-nodeadcode` can slow down simulation and elaboration performance.

1.2.151 -NODEFbopen

(VHDL only)

Disable default binding for any items marked with the `use open` keywords in the configuration.

By default, the elaborator ignores the `use open` keywords in the VHDL configuration and uses default binding if a match is available in the visible compiled libraries. For example, consider the following two files:

```
-- File: test.vhd
library ieee;
use ieee.std_logic_1164.all;
library std;
use std.textio.all;

entity test is
end;

architecture rtl of test is
```

```
  component sub is
    port (my_in: in std_logic);
  end component;

  signal a : std_logic;
  begin
    s1 : sub port map (my_in => a);
    s2 : sub port map (my_in => a);
end rtl;

-- File: cfg.vhd
library lib1;
library lib2;

configuration cfg of test is
for rtl
  for s1 : sub use entity lib1.sub;
  end for;
  for s2 : sub use open;
  end for;
end for;
end cfg;
```

In this example, instance `s1` will be bound to `lib1.sub`. However, if component `sub` has also been compiled into `worklib`, the elaborator will bind it to `s2`, as that library is always visible.

```
ncelab: *W,CUDEFB: default binding occurred for component instance (:test(rtl):s2)
with design unit (WORKLIB.SUB:RTL)
```

If component `sub` has been compiled into `lib1` or `lib2`, and the file `test.vhd` contains `library` and `use` clauses for `lib1` and `lib2`, the elaborator will bind it to `s2` with the following warning:

```
ncelab: *W,CUDEFB: default binding occurred for component instance (:test(rtl):s2)
with design unit (LIB1.SUB:RTL).
```

Use the `-nodefbopen` option (`ncelab -nodefbopen` or `irun -nodefbopen`) to leave instances specified with `use open` unbound, regardless of where the component is compiled.

This option is useful in complex designs that can be partially simulated before all subsystems are complete. Specific component instances can be left unbound until later in the design cycle. As the design of each subsystem is completed, the corresponding component configuration is updated to bind to the new entity.

1.2.152 -NOEsp

(Verilog only)

Ignore edge-sensitive path delays in `specify` blocks.

When describing a module path in a `specify` block, you can specify an edge transition at the source to model the timing of input to output delays that occur only when a specified edge occurs at the source signal. This is called an *edge-sensitive path delay*. For example:

```
specify
  (posedge a => (z:a)) = 5;
  (negedge a => (z:a)) = 10;
endspecify
```

In this example, when signal `a` transitions from `0` to `1`, the signal `z` will be updated 5 time units later. When signal `a` transitions from `1` to `0`, the signal `z` will be updated 10 time units later.

Edge-sensitive path delays are on by default. Use the `-noesp` option to disable edge-sensitive path delays. You might do this for backward compatibility with Verilog-XL.

If you use the `-noesp` option, the edge-qualifier is ignored. The `specify` block shown above is interpreted as follows:

```
specify
  (a => z) = 5;
  (a => z) = 10;
endspecify
```

This results in all transitions from signal `a` to signal `z` getting the smallest delay of the active path delays (5, in this example).

See "Edge-Sensitive Module Paths" for more information on edge-sensitive path delays.

1.2.153 -NOIpd

(VHDL only)

Ignore the input path delays in a VITAL level 1 cell and read the non-delayed input signals directly.

This option causes the elaborator to ignore the lumped interconnect delays that are specified on input ports in the `WIREDELAY` block. Use this option to speed up the simulation of circuits when the input port delays should be ignored.

For distributed delay models (VITAL primitive procedures that have delays on them), VITAL lets you specify a different delay from each input to the output. You may not need this level of accuracy for all of your simulation runs and you may want to compromise on it to improve speed and/or memory in your simulation.

1.2.154 -NOLog

Do not generate a log file. By default, *ncelab* generates a log file called `ncelab.log`.

The `-nolog` option is overridden by the `-logfile` option.

If you include the `-nolog` option in the `hdl.var` file, a log file will be created because the file is opened before any variables in the `hdl.var` file are processed. However, once the option is processed in the `hdl.var` file, output to the log file is discontinued, and the log file will contain only header information.

1.2.155 -NOMxindr

For a mixed Verilog-VHDL design, create a network topology that silently ignores an illegal connection of a VHDL port of mode `in` to a lower-level Verilog port that has a driver. Ignoring this illegal connection eliminates the `MXINDR` error message, which was introduced in the LDV 4.1 release.

In Verilog, a port that has drivers is treated as an output port regardless of how the port is declared. VHDL does not allow an input port connection to a lower-level output port. Therefore, connecting a VHDL port of mode `in` to a lower-level Verilog port that has a driver is illegal in VHDL.

In the LDV 4.0 release, or earlier, this illegal connection was not detected, and the contribution of the Verilog driver to the net was silently ignored in the VHDL component. The net was effectively split and had completely different values on either side of the connection.

In the LDV 4.1 release, a new error message `MXINDR` was introduced to address this particular situation. The error message provides the path to the VHDL input port, and source file information so that you can make the appropriate corrections in your design.

Use the `-nomxindr` command-line option if you encounter the `MXINDR` error and do not want to make the required changes in your design. If you use the option, the elaborator will create the network topology as in releases prior to LDV 4.1, and the illegal connection will be ignored.

See "Port Mode Mismatch Errors" for more information.

1.2.156 -NONEg_tchk

Do not allow negative values in `$setuphold` and `$recrem` timing checks in the Verilog description and in `SETUPHOLD` and `RECREM` timing checks in SDF annotation. If you use this option, any negative values in the description or in the SDF annotation are set to 0 and a warning is issued.

See "$setuphold" and "$recrem" for more information on negative timing checks.

1.2.157 -NONOtifier

(Verilog only)

Ignore notifiers in timing checks.

See "Using Notifiers" for information on using notifiers.

1.2.158 -NOParamerr

(AMS)

Allow undeclared parameters to be overridden. By default, the elaborator reports an error and stops when it encounters a value override for an undeclared parameter.

See the description of the `-noparamerr` option in the chapter "Elaborating" in the *Virtuoso AMS Designer Simulator User Guide* for additional information.

1.2.159 -NORtis

(Verilog only)

Disable the input sense feature of SDF `RETAIN` annotated paths.

Note: With *irun*, you can use the `-nortis` or `+ncsdf_nortis` option.

An SDF `IOPATH` construct can contain a `RETAIN` construct, which specifies the time for which an output or inout port retains its previous logic value after a change at a related input or inout port. For example:

```
(IOPATH addr[13:0] dout[7:0]
    (RETAIN (4:5:7) (5:6:9))     // RETAIN delays
    (15:20:25) (18:22:27)        // IOPATH delays
)
```

In response to a transition on bus addr, output dout will transition to the X state. The first RETAIN value (4:5:7) is the rise time used for dout going from low to X. The second RETAIN value (5:6:9) is the fall time used for dout going from high to X. Output dout will next transition from X to its final state. The first IOPATH value (15:20:25) is used when dout transitions from X to high. The second IOPATH value (18:22:27) is used when dout transitions from X to low.

By default, an IOPATH annotated with a RETAIN delay is sensitive to the input, even though the output may not change. Use the -nortis option to disable this input sense feature. Transitions to the X state will be visible after the RETAIN delays only if the output has changed. The X values will not be visible if the output does not change.

1.2.160 -NOSOurce

Ignore source file timestamps when using the -update option.

1.2.161 -NOSPecify

(Verilog only)

Note: The -nospecify option is intended to be used for pure Verilog designs. It is not intended for VHDL. Using this option with VHDL will disable SDF annotation.

The -nospecify command-line option is a convenient way to disable several timing features that are usually not required for functional verification. This option affects timing information contained in specify blocks and SDF annotation, as follows:

- Timing information described in specify blocks
  - Module paths and delays described in specify blocks are ignored.
  - Timing checks described in specify blocks are ignored. For negative timing checks, delayed signals are processed to establish correct logic connections, with zero delays between the connections, but the timing checks are ignored.
  - DelayOverride$ specparams in specify blocks are ignored.
- SDF annotation

  All SDF delays and timing checks are ignored.

The -nospecify option can be used with the -delay_mode command-line option to specify the delay mode (zero, unit, path, or distributed), or with the `delay_mode_* compiler directives. The order of precedence in delay mode selection from highest to lowest is as follows:

1. Command-line option
2. Compiler directives
3. Default — no delay mode

1.2.162 -NOSTdout

Suppress the printing of output to the screen.

The `-nostdout` option does not change what is written to the log file.

1.2.163 -NOTImingchecks

Do not execute timing checks.

This option turns off both Verilog and accelerated VITAL timing checks.

Note: The `-notimingchecks` option turns off all timing checks. Because the timing checks have been turned off, any calculation of delays that would normally occur because of negative limits specified in timing checks is disabled. If your design requires that these delays be calculated in order for the design to simulate correctly, use the `-ntcnotchks` option.

1.2.164 -NOVitalaccl

(VHDL only)

Suppress the acceleration of VITAL level 1 compliant cells. Using this option may help you to debug a problem if you are seeing unexpected simulation results. For example, elaborating with this command-line option will halt the simulation for all VITAL ASSERT/WARNING statements.

1.2.165 -NOWarn warning_code[:warning_code ...]

Disable the printing of the warning or note with the specified code.

For example, when elaborating, you may know about unconnected signals in your model. While the individual design units or source files may compile without error, the elaborator will generate port mismatch warning messages. If you are not interested in seeing these messages, use `-nowarn` to turn them off.

The *warning_code* argument is the message code (mnemonic) that appears in the warning message that follows the error severity code.

Example:

```
% ncelab -nowarn CUVWSP worklib.top
```

You can disable the printing of multiple warning messages either by using multiple `-nowarn` options or by using one `-nowarn` option and separating the *warning_code* arguments with a colon. For example,

```
% ncelab -nowarn INTOVF -nowarn CUVWSP worklib.top
% ncelab -nowarn INTOVF:CUVWSP worklib.top
```

1.2.166 -NOXilinxaccl

Disable the acceleration of Xilinx library functions.

Some packages in the Xilinx Integrated Software Environment versions ISE4.2i and ISE5.1i have been accelerated in the Incisive simulator. In the current release, the following libraries are accelerated:

- UNISIM library (VPKG package)
- SIMPRIM library (VPACKAGE package)
- Standard Xilinx cells like RAMS and Flip-Flops in the SIMPRIM library

Acceleration of Xilinx libraries is on by default. Use the `-noxilinxaccl` command-line option to disable acceleration.

1.2.167 -NO_Sdfa_header

Do not print elaborator messages that display information about arguments used in a `$sdf_annotate` system task or in an SDF command file.

By default, the elaborator prints out messages showing these arguments. For example:

```
% ncelab -nocopyright -mess worklib.top
        Elaborating the design hierarchy:
                Caching library 'worklib' ....... Done
        Annotating SDF timing data:
                Compiled SDF file:     my1.sdf.X
                Log file:
                Backannotation scope:  top.testand.insta
                Configuration file:
```

```
                MTM control:
                Scale factors:
                Scale type:
        Annotation completed successfully...
```

Use the `-no_sdfa_header` option to suppress these messages.

```
% ncelab -nocopyright -mess -no_sdfa_header worklib.top
        Elaborating the design hierarchy:
                Caching library 'worklib' ....... Done
        Annotating SDF timing data.
        Annotation completed successfully...
```

1.2.168 -NO_TCHK_Msg

Do not display timing check warning messages.

This option turns off messages for both Verilog and accelerated VITAL timing checks.

1.2.169 -NO_TCHK_Xgen

(VHDL only)

Turn off X generation in accelerated VITAL timing checks.

This option has no effect if you use the `-novitalaccl` option.

1.2.170 -NO_VPD_Msg

(VHDL only)

Turn off glitch messages from accelerated VITAL pathdelay procedures.

This option has no effect if you use the `-novitalaccl` option.

1.2.171 -NO_VPD_Xgen

(VHDL only)

Turn off X generation in accelerated VITAL pathdelay procedures.

This option has no effect if you use the `-novitalaccl` option.

1.2.172 -NTC_Level ntc_level

(Verilog only)

Specify the behavior of the algorithm used for negative timing checks (NTC).

The *ntc_level* argument can be:

- 1

  In level 1, the NTC algorithm assumes that a given NTC topology can be made to converge by calculating a single delay for the different nets. This is how NTC works in Verilog-XL. Level 1 functionality is provided primarily for backward compatibility.

- 2

  Level 2 is the default NTC algorithm. The algorithm is sensitive to the posedge and negedge controls of the timing check inputs. This allows the algorithm to calculate, when needed, delays that have a rise and a fall value, instead of calculating a single delay for the different nets.

- 3

  Level 3 is the new NTC algorithm. This algorithm increases the likelihood that a given NTC topology can be made to converge, especially in the presence of timing checks with multiple conditions. This allows the algorithm to calculate, when needed, conditional delays whose values depend on the timing check conditions.

Example:

```
% ncelab -ntc_level 3 [other_options] worklib.top:module
% irun -ntc_level 3 [other_options] source1.v source2.v
```

See "Negative Timing Check Limits in $setuphold and $recrem" for details on negative timing checks.

1.2.173 -NTC_NEglim

(Verilog only)

Adjust the negative limit for an invalid negative timing check timing window.

In `$setuphold` timing checks, you can specify a negative value for the setup or hold limit. In `$recrem` timing checks, you can specify a negative value for the recovery or removal limit. In both cases, the sum of the two arguments must be 0 or greater. If the value of the negative limit is greater than the value of the positive limit, the negative limit is set to 0 by default. For

example, in the following `$setuphold` timing check, the negative hold value is set to 0 by default:

```
$setuphold (posedge clk, data, 8, -10, ntfy_reg);
```

Use the `-ntc_neglim` option to adjust the negative limit to match the positive limit. For example, if you include `-ntc_neglim`, the timing check shown above is changed to:

```
$setuphold (posedge clk, data, 8, -8, ntfy_reg);
```

Because the timing violation window size is now zero, no violations will be reported. However, the negative values in the timing checks are preserved and used in the calculation of delayed signals.

1.2.174 -NTCNOtchks

(Verilog only)

Generate negative timing check (NTC) delays, but do not execute timing checks.

You can use the `-notimingchecks` option to turn off all timing checks in your design. However, if you have negative timing checks in the design, this option also disables the generation of delayed internal signals, and you may get wrong simulation results if the design requires these delayed signals to function correctly. That is, if you have negative timing checks, simulation results may be different with `-notimingchecks` and without `-notimingchecks`.

Use the `-ntcnotchks` option instead of the `-notimingchecks` option if you want the delayed signals to be generated but want to turn off timing checks. This option removes the timing checks from the simulation after the NTC delays have been generated.

See "Negative Timing Check Limits in $setuphold and $recrem" for details on negative timing checks.

You can enable the computation of negative timing delays for specific instances with the `ntcnotchks` specification in a timing file. See Writing a Timing File for details.

1.2.175 -NTC_Poslim

(Verilog only)

Adjust the positive limit for an invalid negative timing check timing window.

In `$setuphold` timing checks, you can specify a negative value for the setup or hold limit. In `$recrem` timing checks, you can specify a negative value for the recovery or removal limit. In

both cases, the sum of the two arguments must be 0 or greater. If the value of the negative limit is greater than the value of the positive limit, the negative limit is set to 0 by default. For example, in the following `$setuphold` timing check, the negative hold value is set to 0 by default:

```
$setuphold (posedge clk, data, 8, -10, ntfy_reg);
```

Use the `-ntc_poslim` option to adjust the positive limit to match the negative limit. For example, if you include `-ntc_poslim`, the timing check shown above is changed to:

```
$setuphold (posedge clk, data, 10, -10, ntfy_reg);
```

Because the timing violation window size is now zero, no violations will be reported. However, the negative values in the timing checks are preserved and used in the calculation of delayed signals.

1.2.176 -NTC_Tolerance tolerance_value

(Verilog only)

Specify the tolerance value for a negative timing check timing window.

The *`tolerance_value`* argument is an absolute time value followed by a time unit. Valid time units are: `fs`, `ps`, `ns`, `us`, `ms`, and `s`. The default is `ns`.

The `-ntc_tolerance` option does two things:

- Filter out `$setuphold` and `$recrem` timing checks in which the difference between the positive limit and the negative limit is less than the value specified in the *`tolerance_value`* argument.

  For example, suppose that the timescale is 1ns, and that you have the following `$setuphold` timing check:

  ```
  $setuphold (posedge clk, data, 2.5, -2.3, ntfy_reg);
  ```

  In this example, the violation region extends from 2.5ns to 2.3ns before `posedge clk`. The window is only .2ns. If you are not interested in timing windows this small, you can filter out the timing checks with the `-ntc_tolerance` option.

  In this example, specifying a *`tolerance_value`* of .3ns will deactivate the timing check.

  ```
  % ncelab -ntc_tolerance .3ns top_level_module
  ```

  Negative limits specified in the timing check are preserved and used in the calculation of delayed signals.

- Suppress the warning messages that are generated for `$setuphold` and `$recrem` timing checks in which the negative value is greater than the positive value, and in which the difference between the two is less than the value specified in the *`tolerance_value`* argument.

  In `$setuphold` or `$recrem` timing checks, the sum of the two limits must be 0 or greater. If the sum of the two arguments is less than 0, the negative limit is set to 0 by default. For example, in the following timing check, the negative value for the hold limit will be set to 0.

  ```
  $setuphold (posedge clk, data, 2.5, -2.7, ntfy_reg);
  ```

  In these cases, a warning message similar to the following is issued:

  ```
  ncelab: *W,SBNGL2 (./test.v,25|13): The sum of both limits in $setuphold or
  $recrem is less than the tolerance value: the negative limit will be set to
  zero.
  ```

  In some cases, you may want to suppress this warning message for timing checks in which the difference between the negative limit and the positive limit is smaller than some value. Use the `-ntc_tolerance` option to specify this tolerance value.

  For example, in the timing check shown above, the negative limit is greater than the positive limit by .2ns. The following option will suppress the warning message generated for this timing check:

  ```
  -ntc_tolerance .3ns
  ```

  When the value of the negative limit is greater than the value of the positive limit, the negative value is set to 0 by default. You can use the `-ntc_poslim` or `-ntc_neglim` to override this behavior.

1.2.177 -NTC_Verbose

(Verilog only)

Display the limits that have been changed by the negative timing check (NTC) algorithm in order to make a circuit converge. This option also displays all of the non-zero delays calculated for the different nets in an NTC topology.

See “Negative Timing Check Limits in $setuphold and $recrem” for details on negative timing checks.

1.2.178 -NTC_Warn

Print convergence warnings for negative timing checks for both Verilog and VITAL if delays cannot be calculated given the current limit values. By default, warnings are not printed.

See "Negative Timing Check Limits in $setuphold and $recrem" for details on how delays are calculated when negative limits are used.

1.2.179 -OLddeposit

Disable interconnect sharing to keep the `deposit` command from behaving like `force`.

By default, the `deposit` command places a value into a net. The value exists until any driver of that net has to update its output value. If the net has interconnect delays on it, the `deposit` command will behave similar to the `force` command and apply the value to all sides of all interconnects on the net.

Use the `-olddeposit` option at elaboration time to apply the alternate behavior when you want to set the value of a net. This option disables interconnect sharing and keeps the deposited value from jumping to the surrounding interconnects. For example, if you deposit a value on the destination side of an interconnect, the source side will not see this value.

Note: Disabling interconnect sharing may impact simulation performance to as much as 30-percent in some cases.

1.2.180 -OMicheckinglevel checking_level

Specify OMI checking level. The *`checking_level`* argument can be:

- `max`–Maximum checking level. Use this level for early integration testing and to debug problems.
- `std`–Standard checking level. This is the default.
- `min`–Minimum checking level. Select this level to achieve higher performance after problems have been debugged.

See "The Open Model Interface (OMI)" for details on OMI.

1.2.181 -OVERRIDE_Precision

(Verilog only)

Use the timescale precision specified with the `-timescale` option for all modules in the design.

The simulator uses the smallest simulation granularity specified for any module as the simulation granularity for all modules in the design. For example, if one module in the design

specifies the following `timescale directive, the simulation granularity will be one femtosecond.

```
`timescale 1ns / 1fs
```

If you want to improve simulation performance by simulating with a larger simulation granularity without modifying the time unit specified with `timescale directives in all modules, specify the global timescale with the -timescale option and include the -override_precision option. For example:

```
% ncelab -timescale '1ns/1ns' -override_precision top_level_module
```

The timescale unit must be greater than or equal to the timescale precision.If the override precision is greater than the timescale for a module, a warning is generated, and the timescale is increased to match the override precision. For example:

```
// a.v
`timescale 1 ps / 1 fs
module A;
  initial
  $printtimescale;
endmodule

// b.v
`timescale 1 ns / 1 ps
module B;
  initial
  $printtimescale;
endmodule
% irun -timescale 1ns/1ns -override_precision a.v b.v
...
        Elaborating the design hierarchy:
ncelab: *W,CUMPTU: Timescale precision larger than timescale unit for module 'A'.
...
ncsim> run
Time scale of (A) is  1ns / 1ns
Time scale of (B) is  1ns / 1ns
...
```

You cannot use this option with the -override_timescale option, which overrides both the time unit and time precision values.

1.2.182 -OVERRIDE_Timescale

(Verilog only)

Assign the timescale specified with the `-timescale` command-line option to all modules in the design.

This option overrides the timescale and time precision values specified with `` `timescale `` compiler directives for all modules with a single, global timescale.

Using this option can increase the performance of RTL simulations in which no timing checks are being performed, without modifying library cells. For example, suppose that you have library elements from a vendor, and that all library elements contain a `` `timescale `` of 1ps/1ps. If you want to run an RTL simulation with no timing checks, you can set a global timescale of 1ns/1ns, to improve simulation performance, decrease the size of the waveform database, and yet retain the library cells as they are for future gate-level simulations. To do this, specify the global timescale with the `-timescale` option and include the `-override_timescale` option, as shown in the following example.

```
% ncelab -timescale '1ns/1ns' -override_timescale top_level_module
```

1.2.183 -PARtialdesign

Elaborate the design and generate a simulation snapshot even if definitions for some instances in the design are not available.

By default, elaboration fails if instances in the design cannot be bound to compiled design units in the libraries. However, in many cases, especially early in the design cycle, some design components may not be available because they are still under development. While a snapshot of a partial design cannot be simulated, you may want to explore the partial design using SimVision tools such as the Schematic Tracer or Design Browser, or use Incisive Formal Verifier (IFV) to formally verify design units that contain instances of units that are not yet defined.

The `-partialdesign` option enables elaboration of a partially specified design. This option instructs the elaborator to ignore certain missing definitions and continue elaborating the design.

- For Verilog, missing definitions for modules and UDPs are ignored.

  Note: For SystemVerilog, missing definitions for modules and program blocks are supported. Missing definitions of other SystemVerilog constructs, such as interfaces or classes, are not supported.

- For VHDL, missing definitions for entities or configurations are ignored.

If an instance declaration does not have a corresponding definition, the elaborator issues a message telling you that an instance of a design unit could not be resolved.

A snapshot of a partially specified design cannot be simulated. If you try to simulate a snapshot of a partial design, the simulator generates an error message `SSNOTS` telling you that the snapshot is not simulatable.

You can invoke SimVision in post-processing mode with the `-ppe` option to explore the partially defined design.

```
% ncsim -ppe snapshot_name
```

You can also load the snapshot into SimVision with the following command:

```
% simvision -snapshot snapshot_name
```

You cannot decompile a snapshot of a partial design with the NC decompiler (*ncdc*).

1.2.184 -PATHPulse

(Verilog only)

Enable `PATHPULSE$` specparams, which are used to set module path pulse control on a specific module or on specific paths within modules.

See "Setting Pulse Controls" for more information.

1.2.185 -PATHTran

(Verilog only)

Remove multiple path delays driven by a single `tran` gate.

By default, a bidirectional primitive (one of the types of `tran` gate) can drive only one `specify` path output. An elaborator error is generated if multiple `specify` path delays are driven by a single bidirectional element, or if multiple bidirectional nets drive one path delay. For example:

```
module pad_ae(pad, ipp_ina, pad_res);
  inout pad;
  inout ipp_ina;
  inout pad_res;

  tran (pad, ipp_ina);
  tran (pad_res, pad);
```

```
  specify
    (pad => ipp_ina) = (0.2, 0.4);
    (pad_res => pad) = (0.2, 0.3);
  endspecify

endmodule

% irun -nocopyright test.v
file: test.v
        module worklib.pad_ae:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
...
   inout pad;
           |
ncelab: *E,TRNTOA (./test.v,3|11): A bidirectional net driving a specify path delay
is also an alias of another bidirectional net driving a specify path delay:
pad_ae.pad.
   inout pad_res;
               |
ncelab: *E,MTRNAL (./test.v,5|15): Multiple specify path delays are driven by a
single bidirectional element: pad_ae.pad_res.
        Design hierarchy summary:
...
irun: *E,ELBERR: Error during elaboration (status 1), exiting.
```

Use the `-pathtran` option to remove the path delays that cannot work with these `tran` type gates. The elaborator issues a warning for each path delay that was deleted, and the design is simulated without these path delays.

```
% irun -nocopyright -pathtran test.v
...
        Elaborating the design hierarchy:
...
ncelab: *W,TRPTKL: The path delay on net "pad" for instance pad_ae(pad_ae) was
removed because of improper interactions with tran type gates.
ncelab: *W,TRPTKL: The path delay on net "ipp_ina" for instance pad_ae(pad_ae) was
removed because of improper interactions with tran type gates.
        Design hierarchy summary:
...
        Writing initial simulation snapshot: worklib.pad_ae:v
Loading snapshot worklib.pad_ae:v ..................... Done
```

1.2.186 -PErfstat

Dump the high level profile statistics in `ncperfstat.out`.

You can use the `-perflog` option to dump these statistics to a specified file. By default, if you do not specify the `-perflog` option with `-perfstat`, then the statistics will be written to a file in the working directory named `ncperfstat.out`. However, if you specify `-perflog <filename>` along with the `-perfstat` option, then the statistics will be written to the file specified with `<filename>`.

Note: If you do not specify the `-perfstat` option with `-perflog`, then the option will be ignored and a warning will be generated. No logs will be written to the specified file.

You can also know the exact location of perfstat logfile using the `logfile -show -perfstat` command. When `-perfstat` is used with `logfile -show` command, then, the full directory path where the perfstat logfile is stored, is displayed.

Refer to Using the Light Weight Profiler in *Maximizing Simulation Performance* for more details.

1.2.187 -PLI_Export

(Verilog only)

Enable the export of symbols from dynamic libraries that are loaded with the `-loadpli1` or `-loadvpi` command-line options.

In the IUS 5.4 and earlier releases, symbols in dynamic libraries were exported by default. Because this sometimes resulted in symbol collisions if the same symbols were defined in multiple applications, this default export of symbols has been turned off.

Use the `-pli_export` option to enable the export of symbols if one dynamic library has a dependency on another dynamic library.

Alternatively, you can add the `:export` qualifier to the `-loadpli1` or `-loadvpi` option. For example, suppose that a dynamic library called `libddr.so` has a dependency on a dynamic library called `libdigeo.so`. Use the following command-line option:

```
% ncelab -loadpli1 libdigeo:digeo_boot:export -loadpli1 libddr:ddr_boot
top_level_design_unit
```

A third alternative is to link the library that has the dependency against the library it is dependent upon when building the dependent library.

1.2.188 -PLINOOptwarn

(Verilog only)

Display only one warning message the first time that a PLI read, write, or connectivity access violation is detected.

By default, the elaborator displays all of the warning and error messages that are generated when an error is detected due to a PLI access violation. Use this option to suppress the display of these access violation messages. If you use this option, a warning message is displayed once, when the first read or write access violation is detected. The message is displayed again if an access violation is detected after a reset or a restart has been executed.

Example:

```
% ncelab -plinooptwarn worklib.top
```

1.2.189 -PLINOWarn

(Verilog only)

Suppress the display of PLI warning and error messages. These messages are displayed by default.

Example:

```
% ncelab -plinowarn alu_16
```

1.2.190 -PLIVerbose

(Verilog only)

Display information about PLI1.0 and VPI task and function registration. This option provides more detailed messages to help you debug your PLI applications.

The `-pliverbose` option must be used when you invoke the elaborator (`ncelab -pliverbose`) and when you invoke the simulator (`ncsim -pliverbose`).

This option displays:

- Information on system environment variables that were used to load dynamic libraries, along with their contents
- The full path to dynamic libraries that were loaded
- All registered system tasks and functions

1.2.191 -PREserve

(VHDL only)

Preserve resolution functions on signals that have only one driver.

This option allows reflexive signal calls to the resolution function; otherwise, these calls are removed for simulation performance improvement.

A resolved signal is called *reflexive* when it has only one source and the value of the signal is defined to be the same as that source. This case is common. Type conversions are not required to resolve reflexive signals because the output is the same as the input.

The elaborator identifies reflexive signals and removes the call to the resolution function in the simulator. Removing this function improves the performance of the signal evaluation process.

To always call resolution functions, use `-preserve`.

1.2.192 -PRIMBind

Include primary snapshots when searching for library units to bind to instances during final elaboration.

For example:

```
ncelab tb primbind
```

During final elaboration, this option (`ncelab -primbind` or `irun -primbind`) includes primary snapshots when building the simulation snapshot.

With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.193 -PRIMHrefupdate

Enable automatic re-elaboration of primary snapshots when objects need additional hierarchical reference permissions.

This option (`ncelab -primhrefupdate` or `irun -primhrefupdate`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.194 -PRIMLibdir directory

Specify the directory where the first `irun` command was run, if the invocation directories are different; or, specify the directory assigned by the `-nclibdirname` option to the first `irun` command.

Note: This option has been deprecated in favor of the `-primname <name>@<directory>` syntax.

This option is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.195 -PRIMName name[@directory]

Specify which previous `irun` command created the primary snapshot.

The *`name`* argument can be:

- The argument to the `-snapshot` option specified on the `irun` command line used to create the primary snapshot.
- The argument to the `-name` option specified on the `irun` command line used to specify the top-level design unit when creating the primary snapshot.
- The argument to the `-ncuid` option given to the first `irun` command.

  If `-ncuid` is used in addition to the `-snapshot` option, specify the argument used for `-ncuid`.
- `irun`

  If the `-snapshot`, `-name`, or `-ncuid` options are not used, specify `irun` as the argument to the `-primname` option.

If the primary snapshot was built in another directory or with the `-nclibdirname` option, the directory must also be specified. This directory can also be specified with the `-primlibdir` option if only one `-primname` option is used.

This option is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.196 -PRIMParamsok

Suppress error messages that are generated when code in one partition tries to change the value of a parameter or generic in another primary partition.

This option (`ncelab -primparamsok` or `irun -primparamsok`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.197 -PRIMSnap snapshot_name

Specify a primary snapshot to be included in a simulation snapshot.

Note: Use the `-primsnap` option when running in multi-step invocation mode. If you are using *irun*, use the `-primname` option to indicate which previous `irun` command created the primary snapshot.

This option is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.198 -PRIMTop module_name

Specify the top level of a primary partition when using the `-genhref` option to generate a hierarchical permission file.

For example:

```
% irun -f dutsrc.f tb_top.v test1.v -genhref href.txt -primtop dut
```

This option (`ncelab -primtop` or `irun -primtop`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.199 -PRIMVhdlcompat

Generate a primary snapshot that is compatible with VHDL.

This option (`ncelab -primvhdlcompat` or `irun -primvhdlcompat`) is used in Multi-Snapshot Incremental Elaboration (MSIE). With MSIE, you partition the design into the following sections:

- One or more primary sections, which contain the stable part of the design.
- An incremental section, which contains the part of the design that is changing.

Each section is elaborated separately using multiple snapshots: one or more primary snapshots and the final, simulation snapshot. At simulation time, all the specified snapshots

are combined. If a change is made in the incremental partition, it is not necessary to re-elaborate the primary snapshot(s).

See *Multi-Snapshot Incremental Elaboration* for details on MSIE.

1.2.200 PRINt_hdl_precision

Print VHDL time precision information in the elaborator log file.

By default, the elaborator prints the simulation precision for any design that has a Verilog top level if there is a `timescale compiler directive or a $timescale task, or if the -timescale command-line option is used. However, if the top level is VHDL, the precision information is not printed.

Use the -print_hdl_precision option to print the simulation precision for a VHDL design during elaboration. For example:

```
% ncvhdl -nocopyright fa1.vhd test_adder.vhd test_adder_conf.vhd
% ncelab -nocopyright -messages -print_hdl_precision -access +rwc \
-vhdl_time_precision 1ns WORKLIB.TEST_ADDER:BEHAVIORAL
        Elaborating the design hierarchy:
...
...
        Design hierarchy summary:
                                Instances  Unique
                Components:         3       2
                Default bindings:   1       -
                ...
                Simulation timescale:  1ns
        Writing initial simulation snapshot: WORKLIB.TEST_ADDER:BEHAVIORAL
```

For a mixed-language design, the elaborator output includes information for both the VHDL and Verilog hierarchy. Use the -print_hdl_precision option to print the precision for the VHDL hierarchy. In the following example, the precision for the Verilog hierarchy is printed because the -timescale option is used on the command line.

```
% ncelab -nocopyright -print_hdl_precision -messages -timescale '1ps/1ps' \
-vhdl_time_precision 10ps worklib.top:module
        Elaborating the design hierarchy:
...
...
        Design hierarchy summary:
                                    Instances  Unique
        VHDL Design hierarchy:
```

```
        Components:              2       1
        Default bindings:        4       1
        ...
        Simulation timescale:   10ps
Verilog Design hierarchy:
        Modules:                 3       2
        Registers:               4       3
        ...
        Simulation timescale:  1ps
Writing initial simulation snapshot: worklib.top:module
```

1.2.201 -PROpspath property_file

(AMS)

Use the specified property file (`prop.cfg`) in a non-5x config flow.

The `prop.cfg` file is an ASCII text file listing the source file locations of analog block netlists. Each entry gives the location of the file in which that cell is defined.

See “-propspath Option” in the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for information on the `-propspath` option.

See “The Property File” in the chapter “Setting Up Your Environment” in the *Virtuoso AMS Designer Simulator User Guide* for information on the property file.

1.2.202 -PULSE_E error_percent

(Verilog only)

Set the percentage of delay for the pulse error limit for both module paths and interconnect. If the `-pulse_int_e` option is also used, this option applies only to module paths.

See “Setting Pulse Controls” for more information.

1.2.203 -PULSE_INT_E error_percent

(Verilog only)

Set the percentage of delay for the pulse error limit for interconnect only.

See “Setting Pulse Controls” for more information.

1.2.204 -PULSE_INT_R reject_percent

(Verilog only)

Set the percentage of delay for the pulse reject limit for interconnect only.

See "Setting Pulse Controls" for more information.

1.2.205 -PULSE_R reject_percent

(Verilog only)

Set the percentage of delay for the pulse reject limit for both module paths and interconnect. If the `-pulse_int_r` option is also used, this option applies only to module paths.

See "Setting Pulse Controls" for more information.

1.2.206 -Quiet

In the log file, print the *ncelab* tool banner and the command-line arguments used when the tool was invoked, but suppress the display of the summary messages from the elaborator.

Using this option can enhance the readability of the log file when there are a large number of source files because it suppresses the output of verbose informational messages. It is also useful because the tool banner and the command-line arguments that were used are stored in the log file.

This option suppresses the "summary" output from the elaborator. It does not suppress tool warning or error messages.

Note: If you also include the `-messages` option on the command line, or if you have created an `hdl.var` file that contains a definition of the `NCELABOPTS` variable that includes the `-messages` option (`DEFINE NCELABOPTS -messages`), the `-messages` option overrides the `-quiet` option, and the summary messages are printed to the log file.

1.2.207 -Relax

(VHDL only)

Enable a looser interpretation of some VHDL rules specified in the LRM.

This option relaxes the interpretation of the following rules:

- Allow design units to be visible for default binding when those design units exist in a library that has not been made visible with a `LIBRARY` declaration in the VHDL source and when the design units do not exist in the library that has been defined as the work library.

  By default, the simulator adheres to a strict interpretation of the VHDL LRM, which states that you must use `LIBRARY` statements with corresponding `USE` clauses in the source code to provide visibility to the declarative region that an unbound instance resides in. To bind component instances to compiled design units in the libraries, the elaborator:

  a. Uses explicit binding indications.

  b. If there is no explicit binding indication, the elaborator tries to bind the component to (in order):

  1. A design unit made visible with a `USE` clause given to the architecture instantiating the component.

  2. A design unit made visible with a `USE` clause given to the entity of the architecture instantiating the component.

  3. A design unit available in the library into which the component was compiled. For example, if you have the following instantiation statement:

  ```
  inst1 : DUT port map (......)
  ```

  and the component `DUT` was compiled into library `LIB_COMP`, the elaborator will search for entity `DUT` in the library `LIB_COMP`.

  4. A design unit in the work library.

  If a binding cannot be found, the elaborator generates an error.

  The `-relax` option extends the set of binding rules followed when a component is being instantiated using default binding. The search order used with the `-relax` option is as follows:

  1. A design unit made visible with a `USE` clause given to the architecture instantiating the component.

  2. A design unit made visible with a `USE` clause given to the entity of the architecture instantiating the component.

  3. A design unit available in the library into which the component was compiled.

  4. A design unit in the work library.

  5. A design unit made visible with a `LIBRARY` clause given to the architecture instantiating the component (no corresponding `USE` clause).

6. A design unit made visible with a `LIBRARY` clause given to the entity of the architecture instantiating the component (no corresponding `USE` clause).

7. A design unit in a library defined in the `cds.lib` file. If a binding has not been found, the elaborator opens the `cds.lib` file and searches all of the libraries that are defined in the file that have not already been searched. The search stops when the elaborator finds a component that has the same name or after all libraries have been searched and binding has failed.

When using the `cds.lib` file for visibility, the elaborator searches the libraries in the order in which they appear, and `cds.lib` files are searched sequentially in the order that they appear in the main `cds.lib` file. For example, given the following `cds.lib` file, the search order would be: `foo3`, `foo1`, `foo2`, `foo4`.

```
# File: cds.lib
INCLUDE my_cds.lib
DEFINE foo1 ./foo
DEFINE foo2 ./foo2
INCLUDE my_other_cds.lib

# File: my_cds.lib
DEFINE foo3 ./foo3

# File: my_other_cds.lib
DEFINE foo4 ./foo4
```

If a library is redefined, the new definition supersedes the old definition. For example, given the following `cds.lib` file, the order of libraries to be searched is `lib2`, `lib1`, `lib3`.

```
# File: cds.lib
DEFINE lib1 lib1
DEFINE lib2 lib2
DEFINE lib1 lib1
DEFINE lib3 lib3
```

For a mixed-language design in which the top-level is VHDL, the elaborator will select a VHDL unit over a Verilog unit that has the same name, even if the VHDL unit is in a library that is listed after the library that contains the Verilog unit. If the top-level is Verilog, the elaborator will select a Verilog unit over a VHDL unit that has the same name.

Note: The `ncelab` `-lib binding` option can also be used to relax the strict default binding search order. The search order used with `-lib_binding` is the same as that used with `-relax`, except that `-lib_binding` does not include searching for a design unit in a library defined in the `cds.lib` file (number 7 in the order shown above).

- Array shape mismatch check.

  The simulator reports an array shape mismatch error if the port width declared in a component instantiation or component declaration differs from the port width in the entity declaration when using generics in the width definition. If you use the `-relax` option when you compile the source (`ncvhdl -relax`) and when you elaborate the design (`ncelab -relax`), the simulator relaxes the array shape checking rules so that the error is not generated.

  See the description of the `ncvhdl` `-relax` option for details and for an example.

1.2.208 -SCCreateviewables

(NC-SC only)

Create `ncsc_viewable` objects inserted by *ncsc_wizard*.

See the section “Using the Transformation Wizard” in the chapter “Code Transformation Wizard” in the *SystemC Simulation Reference* for more information.

1.2.209 -SCOnly

(NC-SC only)

Specifies that the *`cell`* argument to the `ncelab` command is the name of the top-level SystemC module in a SystemC-only design.

The `-sconly` option cannot be used with the `-sctop` option.

See “Elaborating SystemC Designs” in the chapter called “Simulating SystemC Models” in the *SystemC Simulation User Guide* for details on elaborating designs containing SystemC models.

1.2.210 -SCParameter param_name=value

(NC-SC only)

Associate a value with a top-level SystemC parameter.

See the section “Elaborating SystemC Designs” in the chapter “Simulating SystemC Models” in the *SystemC Simulation User Guide* for details on this option.

1.2.211 -SCTop name

(NC-SC only)

Specify the SystemC module name to be used as the top-level of a mixed SystemC/HDL design.

You cannot use the `-sctop` option with the `-sconly` option.

See “Elaborating SystemC Designs” in the chapter called “Simulating SystemC Models” in the *SystemC Simulation User Guide* for details on elaborating designs containing SystemC models.

1.2.212 -SCUpdate

(NC-SC only)

Update SystemC design units used in the design.

1.2.213 -SDF_Cmd_file sdf_command_file

Use the specified SDF command file to control SDF annotation.

For VITAL SDF annotation, you must write an SDF command file and then include the command file with the `-sdf_cmd_file` option. For Verilog, you can annotate by using `$sdf_annotate` or by using an SDF command file.

See “SDF Timing Annotation” for details on SDF annotation.

1.2.214 -SDF_File sdf_filename

(Verilog only)

Use the specified SDF file instead of the SDF file specified in the `$sdf_annotate` system task. This option lets you override the file specified as an argument to the `$sdf_annotate` system task. For example:

```
% ncelab -sdf_file sdf2.sdf worklib.top:module
```

The specified SDF file can be a file that has been encrypted with *ncprotect*. For example:

```
% ncelab -sdf_file sdf2.sdfp  worklib.top:module
```

If multiple `-sdf_file` options are detected on the command line, a warning is generated telling you that the first option on the command line will be used.

If there are multiple `$sdf_annotate` tasks in the design, a warning is generated telling you that all of the SDF annotations will use the file specified by the `-sdf_file` option.

1.2.215 -SDF_NOCheck_celltype

(Verilog only)

Disable celltype validation between the SDF annotator and the Verilog description. By default, the annotator checks the type that is specified in the `CELLTYPE` construct against the module name in the description. If there is a mismatch, a warning is generated and no annotation to that module instance is performed.

See "SDF Timing Annotation" for details on SDF annotation.

1.2.216 -SDF_NOPAthedge

(Verilog only)

Ignore edge specifiers in SDF IOPATH specifications.

According to the IEEE SDF specification, a path with an edge specifier in the SDF file must have the same edge in the specify block. For example, by default, the following path specified in the specify block:

```
(clk *> out) = 5;
```

Will not be annotated by the following SDF construct:

```
(IOPATH (posedge clk) out (1))
```

Use the `-sdf_nopathedge` option to ignore the edge specifiers in the SDF file and apply the annotation to the pathdelay(s) that do not have the edge specifier in the HDL.

1.2.217 -SDF_NOPUlse

(Verilog only)

Ignore pulse reject and error limit specifications in an SDF file.

Path pulse reject and error limits can be specified in an SDF file in `IOPATH`, `PATHPULSE`, or `PATHPULSEPERCENT` statements. Use the `-sdf_nopulse` option if you want to ignore the path pulse information in an SDF file.

Path pulse information specified in `PATHPULSE$` specparams in `specify` blocks will be used. You must include the `-pathpulse` option to enable `PATHPULSE$` specparams. If path pulse limits are not specified in a `specify` block, they are calculated based on the global reject/error limits, which can be specified on the command line with the `-pulse_r`, `-pulse_e`, `-pulse_int_r`, and `-pulse_int_e` options.

Example:

```
% ncelab -sdf_nopulse -pathpulse worklib.top:module
% irun -sdf_nopulse -pathpulse source_files
```

See "Setting Pulse Controls" for more information.

1.2.218 -SDF_NO_Warnings

Do not report warning messages from the SDF annotator.

See "SDF Timing Annotation" for details on SDF annotation.

1.2.219 -SDF_Orig_dir

(Verilog only)

Specify that the automatically compiled SDF file will written to, and read from, the directory where the SDF file is stored.

By default, a `$sdf_annotate` task looks for a compiled SDF file and writes the compiled SDF file in the directory in which the tool was invoked. Use the `-sdf_orig_dir` option if you want to write the compiled SDF file in the directory where the SDF file is located. For example:

```
ncelab -access rwc -sdf_verbose -sdf_orig_dir top_level_unit
```

Or:

```
irun -access rwc -sdf_verbose -sdf_orig_dir source_files
```

1.2.220 -SDF_Precision precision

Round the precision of timing values in the compiled SDF file.

The SDF compiler (*ncsdfc*) compiles the SDF file with a precision of 1 fs. Use the `-sdf_precision` option if you want to specify a coarser precision. Specifying a coarser precision can improve simulation performance.

The *`precision`* argument consists of an integer and a time unit. The integer can be 1, 10, or 100. The time unit can be `fs`, `ps`, `ns`, `us`, or `s`. No space is allowed between the integer and the time unit.

In the following command, the `-sdf_precision` option specifies a precision of 100 picoseconds for timing values.

```
% ncelab -sdf_precision 100ps
```

Timing values in the compiled SDF file are rounded to the nearest 100 ps. For example, the timing value in the following `IOPATH` statement is rounded to 6.1.

```
(IOPATH in out (6.127))
```

The timing values in the following `IOPATH` statement are rounded to 6.1 : 9.6 : 15.0.

```
(IOPATH in out (6.127:9.554:15.031))
```

1.2.221 -SDF_SImtime

(Verilog only)

Enable simulation-time SDF annotation.

By default, SDF annotation is performed during elaboration. The elaborator recognizes `$sdf_annotate` system tasks in your design source files, and if the `$sdf_annotate` system tasks are scheduled to run at time 0, annotation is performed automatically. The `$sdf_annotate` system tasks must be inside an `initial` block, cannot be preceded by delay or event controls, and cannot be within or follow `for`, `while`, `case`, `repeat`, or `wait` constructs. If a `$sdf_annotate` task violates these requirements, the elaborator generates a warning message telling you that it is ignoring the system task.

Use the `-sdf_simtime` option to enable annotation during simulation.

This option lets you specify a delay or event control. For example:

```
#1000 $sdf_annotate("my.sdf", testand.insta);

#1000000 toggle = 0;
$sdf_annotate("my.sdf", testand.insta);

@(posedge toggle)
$sdf_annotate("my.sdf", testand.insta);
```

It also lets you backannotate different SDF files during the same run. For example, the following code changes the SDF file during simulation based on a signal in the design.

```
initial
  begin
    forever
    begin
      wait (toggle == 1'b0)
      $sdf_annotate("my1.sdf", testand.insta);
      wait  (toggle == 1'b1)
      $sdf_annotate("my2.sdf", testand.insta);
    end
  end
```

Although annotation takes places at simulation time, all SDF files are processed at elaboration time. Compiled SDF files cannot be updated after elaboration. If the file is modified after elaboration, the design must be re-elaborated.

All warnings and errors that are detected are written to the appropriate log file during elaboration. Warnings detected during elaboration are not repeated during simulation.

Annotation to primitives is not supported. If a request to annotate to a primitive is detected, the elaborator generates an error and no annotation is performed.

Because some performance optimizations must be turned off in order to allow annotation during simulation time, simulation-time SDF annotation affects simulation performance. If you use the `-sdf_simtime` option, any annotation request that can be determined to happen at simulation time 0 is annotated during elaboration to allow for better simulation performance.

The `-sdf_simtime` option does not affect elaboration-time SDF annotation to VITAL. If an SDF file scheduled for simulation time has an annotation to a VITAL construct, an elaboration error is generated.

See "SDF Timing Annotation" for details on SDF annotation.

1.2.222 -SDF_SPecpp

(Verilog only)

Use `PATHPULSE$` specparams in `specify` blocks for calculating path pulse reject and error limits.

When a path delay is SDF annotated, the original path pulse limits of the path are discarded, and new path pulse limits are calculated based on the SDF path delay. If the SDF delay does

not contain pulse information, or the path pulse information is not specified in the SDF file in PATHPULSE or PATHPULSEPERCENT statements, global reject and error limits are used. If you include the -sdf_specpp option, the path pulse reject and error limits specified in PATHPULSE$ specparams in specify blocks are used.

You must include the -pathpulse option on the command line to enable the use of PATHPULSE$ specparams.

Note: Use the -sdf_nopulse option if you want to ignore any path pulse information in an SDF file and use the limits specified in PATHPULSE$ specparams. The -sdf_nopulse option automatically enables -sdf_specpp.

1.2.223 -SDF_SPLIT_Two_timing_check -SDF_SPLITVLOG_Setuphold -SDF_SPLITVLOG_Recrem

(Verilog only)

When a SETUPHOLD or RECREM is present in an SDF file, by the IEEE 1497 standard, the annotation must occur to a corresponding $setuphold or $recrem in the specify block of the Verilog source. If there is no corresponding match then no annotation occurs, even if separate $setup, $hold, $recovery, or $removal timing checks are present in the Verilog source.

- The -sdf_split_two_timing_check option allows both the SETUPHOLD and RECREM present in the SDF source to be split and annotated to corresponding $setup, $hold, $recovery, and $removal timing checks in the Verilog source.

  If only a $setup or $hold is present for a single SETUPHOLD, then only that timing check will be annotated from the SETUPHOLD. The same is true for RECREM if only a single $recovery or $removal timing check is present.

- The -sdf_splitvlog_setuphold option allows only the SETUPHOLD present in the SDF source to be split and annotated to corresponding $setup and $hold timing checks in the Verilog source. If only a $setup or $hold is present for a single SETUPHOLD, only that timing check will be annotated. A RECREM timing present in the source will not be split to corresponding $recovery or $removal timing checks.

- The -sdf_splitvlog_recrem option allows only the RECREM present in the SDF source to be split and annotated to corresponding $recover and $removal timing checks in the Verilog source. If only a $recovery or $removal is present for a single RECREM, then only that timing check will be annotated from the RECREM. A SETUPHOLD timing present in the source will not be split to corresponding $setup or $hold timing checks.

Example:

SDF file:

```
(SETUPHOLD data clk () (0.2))
(RECREM clk  clk (100) ())
```

The following timing checks are in the specify block of the corresponding Verilog source:

```
specify
$hold(data, clk, ...
$recovery(clr, clk, …
```

Using the `-sdf_split_two_timing_check` option would allow the timing from the `SETUPHOLD` to be annotated to the single `$hold` timing check, and the timing from the `RECREM` to be annotated to the single `$recovery` timing check.

1.2.224 -SDF_Verbose

Include detailed information in the SDF log file.

You specify the SDF log file with the *`log_file`* argument of the `$sdf_annotate` system task or with the `LOG_FILE` statement in an SDF command file.

See "$sdf_annotate System Task" for information on the arguments of the `$sdf_annotate` task. See "Writing an SDF Command File" for details on the SDF command file.

Note: SDF files can be encrypted with the *ncprotect* utility. Data that corresponds to protected regions in the file will not be included in the log file.

1.2.225 -SDF_Worstcase_rounding

(Verilog only)

For timing values in the SDF file, truncate the min value, round the typ value, and round up the max value. For example, using this option changes the annotated timing values in the following `IOPATH` statement to 0 : .1 : .1 (assuming a precision of .1).

```
(IOPATH in out (.05:.05:.03))
```

How a single timing value is treated depends on the command-line option that you use. For example, the timing value in the following `IOPATH` statement is annotated as 0 for `-mindelays`, and as .1 for `-typdelays` and `-maxdelays`.

```
(IOPATH in out (.05))
```

1.2.226 -SDFDir directory

(Verilog only)

Specify that the automatically compiled SDF file will write to and read from the specified directory.

By default, a `$sdf_annotate` task looks for a compiled SDF file and writes the compiled SDF file in the directory in which the tool was invoked. Use the `-sdfdir` option to specify a directory of your choice. For example:

```
ncelab -access rwc -sdf_verbose -sdfdir ./sdf top_level_unit
```

Or:

```
irun -access rwc -sdf_verbose -sdfdir ./sdf source_files
```

1.2.227 -SDFSTats filename

(Verilog only)

Generate a file that contains detailed information on SDF annotation.

The output file contains a listing of annotated scopes and non-annotated paths and/or timing checks. For scopes with non-annotated path delays or timing checks, the names of the elements that were not annotated are listed.

Note: The output file contains information only for non-annotated paths or timing checks. If all paths and timing checks are annotated, no information is printed to the file.

The following is an example SDF statistics output file called `sdfstats.txt`:

```
% ncvlog test.v
% ncelab -sdfstats sdfstats.txt worklib.top
```

Or:

```
% irun -sdfstats sdfstats.txt test.v

% cat sdfstats.txt
<SDF annotation statistics>

Instance fullname= top.b -- lib="worklib" cell="MYBUF" view="module"

    List of Unannotated Timing Checks:

        ($setuphold (in1 (edge= posedge) , (in2 (edge= negedge)))
```

```
Instance fullname= top.a -- lib="worklib" cell="MYAND" view="module"

    List of Unannotated Paths:

        (in1-->out)
        (in2-->out)

    List of Unannotated Timing Checks:

        ($setuphold (in2 (edge= posedge) , (in1 (edge= posedge)))
        ($setuphold (in2 (edge= posedge) , (in1 (edge= negedge)))
```

Note: SDF files can be encrypted with the *ncprotect* utility. Data that corresponds to protected regions in the file will not be included in the output file.

1.2.228 -SEM2009

(Verilog only)

Enable the IEEE Standard 1800-2009 semantics for simulation.

The simulation semantics of the Incisive Simulator are based on the Accelera Standard 3.1a, which has a different simulation cycle as compared to IEEE Standard. You can use the default Accelera Standard 3.1a simulation semantics, or you can specify this option on the command line to use the IEEE Standard 1800-2009 simulation semantics.

For example:

```
ncelab -sem2009 [other_options] worklib.top:module
```

Or:

```
irun -sem2009 [other_options] design_files
```

For more information on IEEE simulation semantics, see Simulation Semantics for SystemVerilog in the *SystemVerilog Reference*.

1.2.229 -SET_eto_pulse

Enable output pulse modeling with set Boolean values for all modules using the Enhanced Timing Output (ETO) delay algorithm.

You can define ETO modules in one of two ways:

- automatically, using the `-accu_path_delay` option
- manually, using the `pathdelay_enhanced` qualifier in the specify block

The `-set_eto_pulse` option assigns a value of either 1 or 0 (whatever is appropriate) to the output for the duration of the pulse. Use this option when X-propagation is not needed.

Alternatively, to set a value of X you can use the `-enable_eto_pulse` option.

See Enhancing Path Delay Accuracy for details on the ETO delay algorithm.

1.2.230 -SETDiscipline argument

(AMS)

Set discipline for a specified scope.

See the section "-setdiscipline Option" in the chapter "Elaborating" in the *Virtuoso AMS Designer Simulator User Guide* for details on this option.

1.2.231 -SEQ_udp_delay delay_specification

Apply the specified delay value to the input/output paths of all sequential UDPs in the design.

The *`delay_specification`* argument can be a real number, or a real number followed by a time unit. The time unit can be `fs`, `ps`, `ns`, or `us`. If no time unit is specified, `ns` is the default.

The specified delay value overrides any delay specified for sequential UDPs in the design, in specify blocks, through SDF annotation, and so on. The option also removes any timing checks associated with sequential UDPs.

Examples

The following options assign a `10 ns` delay to all sequential UDP paths.

```
-seq_udp_delay 10
-seq_udp_delay 10ns
```

The following option assigns a `0.7 ns` delay to all sequential UDP paths.

```
-seq_udp_delay 0.7ns
```

The following option assigns a `5 ps` delay to all sequential UDP paths.

```
-seq_udp_delay 5ps
```

See "Timing Delays and Race Conditions in Gate-Level Netlists" on page 173 for more information on using the option to avoid race conditions. Also see the `-add_seq_delay`, `-delta_sequdp_delay`, and `-sequdp_nba_delay` for information on specifying delays.

1.2.232 -SEQUDP_nba_delay

Add a nonblocking delta delay to sequential UDPs at the end of the simulation cycle, after all values have settled and when nonblocking assignments are evaluated.

This option (`ncelab -sequdp_nba_delay` or `irun -sequdp_nba_delay`) improves on `-delta_sequdp_delay` when running a zero delay simulation with the `-nospecify` option, which disables SDF annotation and the timing features of specify blocks.

See "Timing Delays and Race Conditions in Gate-Level Netlists" on page 173 for more information on using the option to avoid race conditions. Also see the `-seq_udp_delay`, `-add_seq_delay`, and `-delta_sequdp_delay` for information on specifying delays.

1.2.233 -SHow_forces

(Verilog only)

Enable the display of objects that have been forced to a value with a Verilog procedural `force` continuous assignment.

The `-show_forces` option enables the display of Verilog code forces by the Tcl `force -show` command or in the SimVision GUI *Show Forces* display. See the Tcl `force -show` command for more details.

Forces that do not come from Verilog `force` statements do not require this option in order to be displayed.

Note: The Tcl `show_force` variable must be set to `1` at the time the forces are applied to enable the display of forces, whether from the Verilog code or from other sources. This variable is automatically set to `1` when the `-show_forces` option is used or when the simulator is invoked with the `-gui` option.

You can also enable the display of Verilog code forces by compiling the Verilog source files with the `-linedebug` option (`ncvlog -linedebug`). However, compiling with `-linedebug` does *not* automatically set the Tcl `show_forces` variable.

Note: VHDL signals and Verilog wires must have read access. Verilog regs must have read and connectivity access.

1.2.234 -SNapshot snapshot_name

Use the specified name for the simulation snapshot. Use this option to give different elaborations of your design unique snapshot names.

The *snapshot_name* argument is a Library.Cell:View specification. The default, if the full specification is not given, is the cell name.

```
[Library.]Cell[:View]
```

If you do not specify the `-snapshot` option, the snapshot name is the name of the top-level design unit that you specified on the command line. If you specify more than one Verilog top-level module on the command line, the snapshot name is the name of the first top-level module.

Example:

```
% ncelab alu_16 -snapshot alu16_vcd
```

Note: You cannot specify a snapshot name that includes a slash character (/) with the `ncelab -snapshot` option. For example, the following command generates an error message:

```
% ncelab -messages -snapshot worklib.alu_16:behave/SIM alu_16:behave
```

1.2.235 -SPArsearray number_of_array_elements

(Verilog only)

Treat all arrays with the specified number of elements, or greater, as sparse arrays.

If your design contains a very large array, but the simulation writes to only a small number of elements in the array, the amount of memory required to simulate the large array can be reduced by declaring the array as sparse. Instead of allocating space for all elements of the array, the simulator allocates space for only the elements that have non-default values.

Declaring large arrays as sparse tells the simulator that you do not intend to use the entire array. This allows the simulator to perform memory optimizations. It does not affect the behavior of arrays. The behavior of a sparse array is identical to a “normal” non-sparse array.

Note: This feature applies only to one-dimensional arrays of bit vectors, integers, times, and packed structs.

Use the `-sparsearray` option to specify that all arrays over a specified size are to be treated as sparse arrays. The argument is a positive number that indicates the number of array elements. For example:

```
% ncelab -sparsearray 1000 worklib.top
```

This feature is intended to be used when modeling large memories that are not used to their full extent by any one simulation run.

The number of elements in the array that your design writes to also affects the amount of memory that is allocated. In general, the fewer elements in the array that are accessed, the more memory you will save by using sparse arrays. The amount of memory that is saved decreases as the percentage of elements that are written increases.

There is a small run-time performance impact when using sparse arrays.

You can also declare an array as sparse by inserting the `/*sparse*/` pragma anywhere within the declaration of the array or after the semicolon that ends the array declaration. For example:

```
reg [31:0] /*sparse*/ mem [0:3000000];
reg [31:0] mem /*sparse*/ [0:3000000];
reg [31:0] mem [0:3000000]; /*sparse*/
```

1.2.236 -SPECTRE_Argfile_spp arg_file

(AMS)

Run the Spectre parser with the `-spp` option (spp on) when parsing files specified by the `-modelpath` option, and configure spp using options defined in the specified file.

See the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for information on the `-spectre_argfile_spp` option.

1.2.237 -SPECTRE_E

(AMS)

Run the Spectre parser with the `-e` option (cpp on) when parsing files specified by the `-modelpath` option.

See the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for information on the `-spectre_e` option.

1.2.238 -SPECTRE_Spp

(AMS)

Run the Spectre parser with the `-spp` option (spp on) when parsing files specified by the `-modelpath` option.

See the chapter “Elaborating” in the *Virtuoso AMS Designer Simulator User Guide* for information on the `-spectre_spp` option.

1.2.239 -STatus

Print statistics on memory and CPU usage after elaboration.

The following example shows the output of the `-status` option:

```
ncelab: Memory Usage Peak  - 28.5M program + 16.4M data = 44.8M total
ncelab: Memory Usage Final - 28.5M program + 13.6M data = 42.0M total
ncelab: CPU Usage - 0.0s system + 0.0s user = 0.0s total (0.1s, 35.4% cpu)
```

1.2.240 -SVPerf {+ | -} checking_specification

(Verilog only)

SystemVerilog adds the keywords `unique` and `priority`, which can be used before `if` and `case`/`casex`/`casez` statements.

The SystemVerilog LRM specifies that when a `case` or `if` statement is specified as `unique`, the software tools will verify that all of the decision conditions are mutually exclusive, and that they must generate a warning if more than one condition is true, or can be true. Tools are also required to generate a warning if the `case` or `if` statement is evaluated and no branch is executed.

The SystemVerilog LRM also specifies that when a `case` or `if` statement is specified as `priority`, there must be at least one true condition. Tools must generate a warning if the case or if statement is evaluated and no branch is executed.

By default, the simulator performs the checks that SystemVerilog requires for `unique` and `priority` constructs. This semantic checking can have an impact on simulation performance.

Use the `ncelab -svperf` command-line option to disable checking for `unique` and/or `priority` violations.

The *checking_specification* argument begins with a plus sign (disable), or with a minus sign (enable). This is followed by:

- u–Indicates `unique` constructs.
- p–Indicates `priority` constructs.

Examples:

The following option is the default. Checking is performed for both `unique` and `priority` constructs, and warning messages are generated for all violations.

```
ncelab -svperf -up              // Same as -svperf -u-p
irun -svperf -up
```

The following option disables checking of both `unique` and `priority` constructs.

```
ncelab -svperf +up              // Same as -svperf +u+p
irun -svperf +up
```

The following option disables checking of `unique` constructs, but enables checking of `priority` constructs.

```
ncelab -svperf +u               // Same as +svperf+u-p
irun -svperf +u
```

1.2.241 -TFile timing_file [-tfverbose]

(Verilog only)

Use the specified timing file.

A timing file is a text file that lets you turn off timing for particular instances or portions of a design. You can use wildcard characters in the path specification of the timing file to match full or partial instance names at any level. The two supported wildcard characters are:

- An asterisk (*) that matches any instance at the current scope
- Three dots (. . .) used as a suffix that match any instance in the hierarchy below the current scope

With the optional `-tfverbose` argument, you can enable verbose mode which will print out a list of instances matched to each wildcard rule during elaboration. For example, given the following timing rule:

```
PATH tb.*.d* -tcheck
```

And the following *irun* command:

```
% irun -tfile check.tfile -tfverbose top.v dff.v -clean
```

The elaborator will match any instance of `tb` in the design that has a third level name starting with the letter `d`. The results will post to the `irun.log` file, as follows:

```
TFVerbose wildcard match log:
      instance tb.t1.d1 matched mode: *.d*
      instance tb.t1.d2 matched mode: *.d*
      instance tb.t1.d3 matched mode: *.d*
      instance tb.t1.d4 matched mode: *.d*
```

See “Disabling Timing in Selected Portions of a Design” for details on writing and using a timing file.

1.2.242 -TImescale 'time_unit / time_precision'

(Verilog only)

Set the default timescale for Verilog modules that do not have a timescale set.

If any module in a source file specified on the command line has been compiled with a `` `timescale `` compiler directive, all other modules must have a timescale in effect when those modules are compiled. For example, suppose that you have three modules defined in the following three source files:

```
source1.v (includes `timescale 1ns/1ps)
source2.v (no `timescale directive)
source3.v (no `timescale directive)
```

If you compile the source files in the order shown in the following command, the timescale specified with the directive in `source1.v` remains in effect when the modules in `source2.v` and `source3.v` are compiled. All modules have a timescale set, and the design will elaborate successfully.

```
% ncvlog source1.v source2.v source3.v
% ncelab top_level_unit
```

or:

```
% irun source1.v source2.v source3.v
```

However, if you compile the source files in the order shown in the following command, the module defined in `source2.v` will not have a timescale set. The module in `source3.v` will have the timescale specified in `source1.v`. The elaborator generates an error message because not all modules have a timescale set.

```
% ncvlog source2.v source1.v source3.v
% ncelab top_level_unit
```

or:

```
% irun source2.v source1.v source3.v
```

You can avoid this error by:

- Adding a `` `timescale `` directive to each unit (perhaps after a `` `resetall `` directive).
- Including a `` `timescale `` directive in the first unit that you list on the compile command, and then making sure that other units do not override its effect. For example, a subsequent unit that has a `` `resetall `` directive must also have a `` `timescale `` directive.
- Using the `-timescale` option to specify a default timescale for modules that do not otherwise have one.

For example, the following command specifies a timescale of `1ps/1ps` for modules that do not have a timescale set when the design is elaborated. In this example, the module in `source2.v` will have a timescale of `1ps/1ps`. The module in `source1.v` will have the timescale set by the directive (`1ns/1ps`), and this timescale remains in effect for the module in `source3.v`.

```
% ncvlog source2.v source1.v source3.v
% ncelab -timescale '1ps/1ps' top_level_unit
```

or:

```
% irun source2.v source1.v source3.v -timescale '1ps/1ps'
```

Note: The `-timescale` option is ignored if all modules have a timescale set after compilation. In the example shown above, if you compile the source files in the order `source1.v source2.v source3.v`, all modules will have a timescale set. Including the `-timescale` option on the command line has no effect. No warning message is issued.

Using the `-timescale` option is useful in situations where you do not want to, or cannot, edit files that are generated by other tools or that are provided by library vendors. For example, a synthesis tool may generate a structural netlist that models the device as Verilog gates connected by wires. No timing information is needed at this level, and each subcomponent may have a `` `timescale `` directive. You can use the `-timescale` option to specify a timescale for the top-level module.

The format of the argument to the `-timescale` option is the same as that for the `` `timescale `` directive. Enclose the argument in single quotation marks.

Example:

```
% ncelab -timescale '1 ns / 1 ps'
```

1.2.243 -TRanmin

(Verilog only)

Apply the minimum delay if multiple delays drive a single `tran` gate from the same source.

By default, if multiple module path delays drive a bidirectional `tran` gate from the same source, then the elaborator will issue a warning and apply the maximum delay. Use the `-tranmin` option (`ncelab -tranmin` or `irun -tranmin`) to specify the alternative, minimum delay. See Tran Gates with Interconnect and Module Path Delays for more details.

1.2.244 -TYpdelays

Apply the typical delay value from a timing triplet in the form min:typ:max that appears in a specify block in the Verilog description.

This option also selects the typical delay value if the min:typ:max value appears in the SDF file while annotating to Verilog or to VITAL unless an SDF-specific construct is used to override it. For example, if you use `-typdelays` on the command line, but specify MAXIMUM in an SDF command file (`MTM_CONTROL = "MAXIMUM"`), in an SDF configuration file (`MTM = MAXIMUM;`), or in the `$sdf_annotate` task, the typical values in the specify block will be used, but the maximum values in the SDF file will be used.

Example:

```
% ncelab -typdelays top_mod
```

1.2.245 -UPDate

Automatically recompile any out-of-date design units and then re-elaborate the design.

For example:

```
% ncelab top -primsnap dut -update
```

When re-elaborating a snapshot using this option (`ncelab -update`), the elaborator will generate a new snapshot if one of the following conditions are true:

- The design units changed in an available snapshot. For instance, a primary snapshot is out of date when re-elaborating the simulation snapshot using *Multi-Snapshot Incremental Elaboration*.
- The *ncelab* command-line options are different from the previous elaboration.

- A different installation of the software is used for re-elaboration. For instance, the software was updated with a hotfix release.

Note: Because the information contained in some files used as input to the elaborator is not part of the simulation snapshot, a new snapshot is always generated if the command-line options used to specify these files are included on the command line. For example, Verilog configurations (specified with the `-libmap` option) are not compiled configurations, and a new snapshot is always generated if the `-libmap` option is included on the command line. Including the `-modelpath` option on the command line will also cause a new snapshot to be generated because the modelpath related information is not dumped in the snapshot. The only exceptions to this are the `-cdslib` and `-hdlvar` options. For these options, the elaborator will generate a new snapshot only if the argument (that is, the filename and the path to the file) has changed. The elaborator does not generate a new snapshot if the argument is the same, even if the content of the `cds.lib` or `hdl.var` file has changed.

Note: If the location of a source file has changed (for example, if a source file has been moved to a new directory), you can include the `-cmdfile` option to perform incremental compilation. This option specifies a compilation command file in which the `SEARCH_PATH` variable has been defined. The parser uses the search paths specified with this variable to locate the file whose location has changed. See "Writing a Compilation Command File" for details on the compilation command file.

```
% ncelab -update -cmdfile compilation_command_file top_level_unit
```

When using the `-update` option, the compiler, by default, displays information only for the design units that are actually recompiled. Use the `-uptodate_messages` option if you want to display information about all design units, including those that are not being recompiled because they are up-to-date.

1.2.246 -UPTodate_messages

Display compilation information for all design units, including those that are not being recompiled because they are up-to-date, when recompiling source files and re-elaborating the design with the `-update` option.

By default, when recompiling source files, the compiler does not display information about design units that are up-to-date. Information is displayed only for design units that are recompiled. Use the `-uptodate_messages` option if you want to display information about all design units.

The `-uptodate_messages` option can be used only in conjunction with the `-update` option.

1.2.247 -USE5X4VHdl

(AMS)

Use 5.X configurations for elaborating VHDL hierarchies.

A 5.X configuration is an ASCII text file, usually written with a tool such as the Hierarchy Editor, that specifies the rules that the elaborator is to use for selecting design units out of design libraries for binding to instances in a design hierarchy. 5.x configurations are used by Cadence® AMS simulator users.

The `-use5x4vhdl` option affects the set of rules that is used to resolve a VHDL instance during elaboration.

- If you do not include the `-use5x4vhdl` option, the elaborator first uses VHDL language rules (configuration declarations, configuration specifications, entity aspect specifications) and then the default VHDL binding rules to determine a binding.
- If you include the option, the elaborator first uses VHDL language rules to determine a binding. It then uses the binding rules specified in a 5.x configuration specification before using the default binding rules.

1.2.248 -USE5X4VLog

(AMS)

Use 5.X configurations for elaborating Verilog hierarchies.

A 5.X configuration is an ASCII text file, usually written with a tool such as the Hierarchy Editor, that specifies the rules that the elaborator is to use for selecting design units out of design libraries for binding to instances in a design hierarchy. 5.x configurations are used by Cadence® AMS simulator users.

The `-use5x4vlog` option allows 5.X configurations to be used along with library map files. You must use this option to override default binding in the following two situations:

- If a configuration has been specified in the library map file, and you include the `-use5x4vlog` option, 5.X configurations will be used for binding before the default binding through the `default liblist` clause.
- If a configuration has not been specified in the library map file, and you include the `-use5x4vlog` option, 5.X configurations will be used for binding before the default binding through the default configuration in the library map file (that is, by searching the libraries in the order in which they are declared in library declarations).

1.2.249 -V93

(VHDL only)

Enable VHDL-93 features.

See "Features Included from the IEEE 1076-1993 Standard" in the *VHDL Simulation User Guide* for a list of supported VHDL-93 features.

1.2.250 -VErsion

Print the version of the elaborator and exit.

This option is ignored if you include it with the `NCELABOPTS` variable in an `hdl.var` file.

1.2.251 -VHDLSParsearray value

(VHDL only)

Treat all arrays with the specified number of elements, or greater, as sparse arrays in 64-bit mode.

If your design contains a very large array, but the simulation writes to only a small number of elements in the array, the amount of memory required to simulate the large array can be reduced by declaring the array as sparse. Instead of allocating space for all elements of the array, the simulator allocates space for only the elements that have non-default values.

Declaring large arrays as sparse tells the simulator that you do not intend to use the entire array. This allows the simulator to perform memory optimizations. It does not affect the behavior of arrays. The behavior of a sparse array is identical to a "normal" non-sparse array.

This feature is intended to be used when modeling large memories that are not used to their full extent by any one simulation run.

The number of elements in the array that your design writes to also affects the amount of memory that is allocated. In general, the fewer elements in the array that are accessed, the more memory you will save by using sparse arrays.

Notes:

- This feature applies only to one-dimensional arrays of VHDL signals and variables, not ports.
- This feature applies only to the 64-bit version of the compiler and elaborator.

- Use the `-vhdlsparsearray` option to specify that all arrays over a specified size are to be treated as sparse arrays. The argument is any positive number greater than 36384(2 **15).
- The `-vhdlsparsearray` option must be used for both compilation and elaboration.

For example:

```
% ncvhdl -vhdlsparsearray 40000 top.vhd
% ncelab -vhdlsparsearray 40000 worklib.top
```

Limitations:

The following are not supported for a signal implemented as a sparse array:

- Setting probe/SHM dumping
- deposit/value/driver command on the complete signal/variable
- Signal should not be on the sensitivity list of any process
- Guarded signals
- Alias on slice or indexed expression
- Waveform expression
- Assert on the signal
- Multiple drivers of the signal across processes will not be honoured
- Expanded signals like a(0)(0), a(0)(1 downto 0), a(1 to 2)(1)

Example:

```
entity top is
end;

architecture a of top is
type large_arr is array(0 to 2**28-1) of std_logic_vector(8 downto0);
 signal my_arr : large_arr;

begin
end;
```

Run commands:

```
$ ncvhdl top.vhd -64bit -vhdlsparsearray 40000
$ ncelab worklib.top -64bit -vhdlsparsearray 40000
```

1.2.252 -VHDLSYnc

Modify the simulation cycle semantics for a mixed-language design so that Specman generates consistent results regardless of whether the clock is in a Verilog or VHDL portion of the design.

By default, in a simulation cycle for a mixed-HDL design, all signal updates, except for Verilog non-blocking assignments (NBAs) are performed, followed by the execution of VHDL processes and Verilog `always` blocks. The execution of VHDL processes or Verilog `always` blocks can cause more signal updates. After all signal updates and process are executed, Verilog NBAs are executed.

Specman synchronizes (samples values) just before the execution of NBAs. In other words, by default, it synchronizes *after* all VHDL signals are updated, but before Verilog NBAs are executed. This can lead to sampling inconsistencies, depending on whether the clock is in Verilog or VHDL. When VHDL drives Verilog registers and nets, Specman might sense premature value changes on the Verilog signals. When Verilog drives VHDL signals, Specman might miss value changes on those signals.

If you use the `-vhdlsync` option, the simulation cycle semantics are modified so that VHDL signal assignments are treated as NBAs. Because Specman synchronizes before the execution of NBAs, VHDL values are sampled just *before* they are updated. This provides consistent results regardless of whether the sampling signal is in Verilog or VHDL.

When `-vhdlsync` is used, there can be a slight degradation in simulation performance, and the order in which behaviors are executed in a given simulation time may be altered. Both versions simulate correctly, but designs with race conditions can exhibit differences in behavior.

1.2.253 -VHDL_Time_precision time_precision

(VHDL only)

Specifies the timescale precision for VHDL portions of a design.

Note: This option affects VHDL portions of a design only. The time precision for Verilog modules can be set with the `ncelab -timescale` option or with the `` `timescale `` compiler directive in the Verilog source code. This option does not affect analog portions of a design.

According to the IEEE 1076-1993 VHDL Language Reference Manual (Section 3.1.3.1), the primary unit of type `TIME` (1 femtosecond) is, by default, the resolution limit for type `TIME`. All simulations run in femtoseconds by default. Use the `-vhdl_time_precision` option to specify a secondary unit of type `TIME` as the resolution limit.

The *`time_precision`* argument is specified as a value (1, 10, or 100) followed by a time unit. The time unit can be: `s`, `ms`, `us`, `ns`, `ps`, or `fs`.

You must enclose the argument in single or double quotes if you have a space between the value and the time unit. For example:

```
-vhdl_time_precision 1ns
-vhdl_time_precision '1 ps'
-vhdl_time_precision "1 us"
```

If you use this option to specify a time precision, all delays specified in the VHDL design are rounded off to the nearest precision specified with the option. For example, if you compile the following source file and then elaborate the design with `-vhdl_time_precision 1ps`, the delays are rounded off as shown in the table following the source file.

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
end test;

architecture vhdl of test is
  signal sig : std_logic := '0';
  begin
    sig <= '1' after 0.4 ps, '0' after 0.5 ps, '1' after 1.5 ps, '0' after 2.9 ps;
end vhdl;
```

Delay in Design	Delay After Rounding Off
0.4 ps	0 ps
0.5 ps	1 ps
1.5 ps	2 ps
2.9 ps	3 ps

Setting the timing resolution to a coarser value may increase simulation performance, as the simulator will not default to femtoseconds. However, in some cases, the rounding off may result in more time bins, which increases the event density and slows performance. For example, consider the following statement:

```
clk <= not clk after 1.2 ns;
```

If you simulate for 6 ns, without using the `-vhdl_time_precision` option, `clk` changes at 1.2, 2.4, 3.6, 4.8, 6.0 ns (total five time bins).

If you set the time precision to 1ns (`-vhdl_time_precision 1ns`), the 1.2 ns delay is rounded off to 1 ns, and `clk` changes at 1, 2, 3, 4, 5, 6 ns (total six time bins).

While delays specified in the VHDL are rounded off, the `-vhdl_time_precision` option does not change the values of time signals. For example, suppose that you have the following code:

```
constant DEL : time := 6 ns;
clk <= not clk after DEL;
```

If you elaborate with a precision of 10 ns, `clk` cycles at 10 ns intervals, but the value of `DEL` does not change.

```
10 NS + 0 (stop 1: :clk = '1')
20 NS + 0 (stop 1: :clk = '0')
30 NS + 0 (stop 1: :clk = '1')
40 NS + 0 (stop 1: :clk = '0')
ncsim> value :DEL
6000000 fs
```

The time precision specified with `-vhdl_time_precision` affects the Tcl `deposit` command. Simulation times specified in a `deposit` command, such as the following, are rounded off.

```
ncsim> deposit object = value -after time_spec value -after time_spec
```

Delays are also rounded off by the `nc_deposit()` procedure. See "nc_deposit" for details on the `nc_deposit()` procedure.

The Tcl `time` and `run` commands are not affected by this option. The `time` command shows the current simulation time, and the `run` command advances the simulation by the specified time with no precision being applied.

Using the `-vhdl_time_precision` option also affects the display of time units in the SimVision analysis environment. Tools such as the SimVision Waveform viewer will show the precision time unit specified with the option instead of femtoseconds.

In a mixed-language design, the `-vhdl_time_precision` option does not control input signals coming from across the language boundary from Verilog. For example, consider the following mixed-language design:

```
// File: top.v
module top;
  reg sig1, sig2;
  wire out1;

  and_gate inst1 (.in1(sig1), .in2(sig2), .out1(out1));

  initial
    begin
      sig2 = 1'b1;
    end

  always
  #3 assign sig1 = 1'b1;

endmodule


-- File and_gate.vhd
library IEEE;
use IEEE.std_logic_1164.all;

entity and_gate is
  port ( in1, in2 : std_logic;
         out1 : out std_logic);

end and_gate;

architecture beh of and_gate is
  begin
    out1 <= in1 and in2 after 6 ps;
```

```
end beh;
```

Now, suppose that you elaborate this design with the following command, in which the VHDL time precision is set to 10 ps and the Verilog timescale is set to 1 ps:

```
% ncelab -vhdl_time_precision 10ps -timescale '1ps/1ps' work.top
```

In this example, the `-vhdl_time_precision` option changes the 6 ps delay in the VHDL to 10 ps. However, the Verilog signal update at 3 ps causes the VHDL process to wake up at 3 ps, and the assignment to `out1` happens at 13 ps. The outputs are generated at the following times:

0 ps: top.sig2 = 1

3 ps: top.sig1 = 1

13 ps: top.inst1.out1 = 1

1.2.254 -VIPDMAx

(VHDL only)

During VITAL SDF annotation, select the maximum delay value if more than one interconnect specification maps to the same interconnect path delay generic.

By default, the SDF annotator maps every interconnect construct that has the same destination to one `tipd` generic that is associated with the destination port. When more than one construct maps to a given generic, the annotator sets the value of the generic to the last interconnect delay that it encounters. Use the `-vipdmax` option to select the maximum delay value.

Use the `-intermod_path` option if you want to specify unique delays for each source-load path during VITAL SDF annotation. See "VITAL SDF Annotation" for details on VITAL SDF annotation.

You cannot use the `-vipdmax` option with the `-intermod_path` option.

1.2.255 -VIPDMIn

(VHDL only)

During VITAL SDF annotation, select the minimum delay value if more than one interconnect specification maps to the same interconnect path delay generic.

By default, the SDF annotator maps every interconnect construct that has the same destination to one `tipd` generic that is associated with the destination port. When more than one construct maps to a given generic, the annotator sets the value of the generic to the last interconnect delay that it encounters. Use the `-vipdmin` option to select the minimum delay value.

Use the `-intermod_path` option if you want to specify unique delays for each source-load path during VITAL SDF annotation. See "VITAL SDF Annotation" for details on VITAL SDF annotation.

You cannot use the `-vipdmin` option with the `-intermod_path` option.

1.2.256 -VPicompat {1364v1995 | 1364v2001 | 1364v2005 | 1800v2005 | 1800v2008}

(Verilog only)

Specify the default IEEE VPI compatibility mode.

There are incompatibility issues in VPI between the 1364 standards (1364-1995, 1364-2001, and 1364-2005) and between the 1364 standards and the IEEE 1800 SystemVerilog standards (1800-2005 and 1800-2008). By default, the Incisive simulators are compatible with the VPI specified in the IEEE 1800-2005 standard. Because of this, existing VPI applications that are not compliant with the 1800-2005 VPI may not run.

VPI users and application developers are strongly encouraged to update their applications to the 1800-2005 VPI version as soon as possible. Until these upgrades are completed, you can use the `-vpicompat` command-line option to specify a default VPI compatibility mode so that you can run an existing application. The default is `-vpicompat 1800v2005`.

If you are running the simulator in multi-step invocation mode, include this option on the `ncelab` command line if the `-loadvpi` option is also required. The `-vpicompat` option is required on the `ncsim` command line. For example:

```
% ncelab top -snapshot worklib.top
% ncsim -vpicompat 1364v2005 -loadvpi ./vpilib:myvpiapp worklib.top
```

or:

```
% ncelab -vpicompat 1364v2005 -loadvpi ./vpilib:myvpiapp top -snapshot worklib.top
% ncsim -vpicompat 1364v2005 -loadvpi ./vpilib:myvpiapp worklib.top
```

Using the `-vpicompat` option lets you run your VPI applications without modification or recompilation. However, this option sets the compatibility mode for all VPI applications. You

can select only one mode for a given simulation run. If different applications require different modes in the same simulation, you can:

- Define a compiler symbol in your own code in such a way that it is compiled before `vpi_user.h`. You can define the compiler symbol in each of your VPI source code files or in your own header file. For example:

  ```
  #define VPI_COMPATIBILITY_VERSION_1364v2001 1
  #include "vpi_user.h"
  ```

- Specify the compatibility mode using an option on the C compiler command line. For example:

  ```
  -DVPI_COMPATIBILITY_VERSION_1364v2001
  ```

See “VPI Compatibility with IEEE Standards” in the chapter “Introduction to VPI” in the *VPI User Guide and Reference* for information on VPI incompatibilities between the different VPI standards and for details on how to set the compatibility mode.

1.2.257 -WANdwor_compat

Change the way in which nets are resolved when Verilog wand or wor nets are at the mixed-language boundary.

By default, the logic for Verilog wand nets is applied on all of the Verilog and VHDL drivers to compute the resolved value of the net. For example, consider the net `inp` in the following mixed-language example:

```
-- File: top.vhd
library ieee;
use ieee.std_logic_1164.all;

entity top is
  port (top_net : inout std_logic;);
end;

architecture a of top is

  component bottom
    port(inp : inout std_logic);
  end component;

  begin
    top_net <= '0';
    b1: bottom port map (top_net);
```

```
  end;


// File: bottom.v
module bottom (inp);
  inout inp;
  wand inp;
  reg a, b;

  assign inp = a;
  assign inp = b;

  initial
    begin
    #0;
    a = 1;
    b = 1;
    #5 $finish;
    end

  initial $monitor ("Displaying wand resolved value of inp = %b ", inp);

endmodule
```

In this design, the net `inp` has three drivers:

- `a` (= 1) and `b` (= 1) on the Verilog side
- `0` on the VHDL side

By default, the logic for Verilog wand nets is applied on all of the drivers and the resolved value of `inp` is `0`.

Truth table for wand nets

wire/ triand	0	1	x	z
0	0	0	0	0
1	0	1	x	1
x	0	x	x	x
z	0	1	x	z

Use the `-wandwor_compat` option (`ncelab -wandwor_compat` or `irun -wandwor_compat`) if you want the nets resolved in each language independently and then resolved as a normal mixed net. This option changes the default behavior so that there are two levels of resolution, as follows:

1. Apply wand logic on all the Verilog drivers.
2. Using the resolved value of the Verilog drivers, apply wire resolution with the VHDL driver(s).

Truth table for wire nets

wire/ tri	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z

For the example shown above, if you specify `-wandwor_compat`, the resolved value of `inp` is `x`.

The resolution rules used by `-wandwor_compat` apply to more complicated mixed-language scenarios. For example, for a design consisting of a Verilog `top`, which instantiates a VHDL `mid`, which instantiates a Verilog `bot`, the resolved value is computed as follows:

1. Apply wand logic on all the Verilog drivers present in `top` and `bot`.
2. Using the resolved value of the Verilog drivers, apply wire resolution with the VHDL driver(s).

The `-wandwor_compat` option also applies to wor nets. In the example design shown above, if `inp` is a wor net, the resolution logic would be:

1. Apply wor logic on all the Verilog drivers of `inp`.
2. Using the resolved value of the Verilog drivers, apply wire resolution with the VHDL driver(s).

1.2.258 -WARnmax integer

Specify the maximum number of times that a particular warning message can be generated.

The *integer* argument must be greater than zero. If a warning is generated more than the specified number of times, that warning is subsequently ignored.

By default, a warning message is ignored if it is generated more than 1000 times.

1.2.259 -WOrk work_library

Use the specified location as the work library in default binding.

The `-work` option (`ncelab -work` or `irun -work`) overrides the setting for the `WORK` variable in the `hdl.var` file. The elaborator saves the snapshot to the location specified by `-work` at elaboration time rather than to the location of the compiled code.

Note: The `-work` and `-useworklib` command-line options are interchangeable when used with the elaborator. The `-useworklib` option has been deprecated in favor of `-work`.

1.2.260 -XFile filename

(Verilog only)

Use the specified configuration file to enable X-propagation selectively during elaboration.

The `-xfile` option takes a single file as an argument to enable X-propagation on select hierarchical modules or instances. A configuration file should contain one or more lines with the following syntax:

```
SCOPE    <hierarchical_name>    {F | C | D}
```

The configuration file supports the following three modes:

Forward only X (FOX) Mode	F	Always drives X at the output when there is an X in the control signal, regardless of the actual input value. FOX mode uses semantics that are close to the behavior of gate-level simulation. Use FOX mode to propagate X in a pessimistic way.
Compute as Ternary (CAT) Mode	C	Drives the resolved, or merged, value of all possible paths at the output when there is an X in the control signal. CAT mode works similar to the ternary operator in Verilog. Use CAT mode to propagate X in a non-deterministic way and to compare the simulation behavior with that of the real hardware behavior.

Default (LRM) Mode	D	Treats X as a false value, following the Verilog LRM.

Consider the following example, `my.xfile`:

```
SCOPE testbench.inst_42... F
```

This file enables FOX mode for all instances below `testbench.inst_42`. To enable this rule for the design `test.v`, specify the file using the following command:

```
% irun -xfile my.xfile test.v
```

Or:

```
% ncvlog test.v
% ncelab -xfile my.xfile testbench
```

Alternatively, you can use the `-xprop` option to enable X-propagation on a complete design. The `-xfile` option and the `-xprop` option are mutually exclusive. In other words, if both options are used on the same command line, then the `-xfile` option will take priority and the `-xprop` option will be ignored.

To generate a log of X-propagation messages, use the `-xverbose` option.

See Using X-propagation to Solve X-optimism for more details.

1.2.261 -XLifnone

(Verilog only)

Emulate Verilog-XL's `ifnone` SDF annotation implementation.

In the Incisive simulators, an unconditional SDF pathdelay annotates all matching unconditional and conditional pathdelays in the `specify` block, including `ifnone` pathdelays. This behavior is compliant with the IEEE standard.

In Verilog-XL, an unconditional SDF pathdelay annotates all matching unconditional and conditional pathdelays in the `specify` block, except if there is an `ifnone` pathdelay. If the `specify` block contains an `ifnone` pathdelay, only the `ifnone` delay is annotated.

Use the `-xlifnone` option to emulate the behavior of Verilog-XL.

1.2.262 -XProp {F | C}

(Verilog only)

Enable X-propagation globally when elaborating your design.

Use this option to specify one of the following supported modes:

Forward only X (FOX) Mode	F	Always drives X at the output when there is an X in the control signal, regardless of the actual input value. FOX mode uses semantics that are close to the behavior of gate-level simulation. Use FOX mode to propagate X in a pessimistic way.
Compute as Ternary (CAT) Mode	C	Drives the resolved, or merged, value of all possible paths at the output when there is an X in the control signal. CAT mode works similar to the ternary operator in Verilog. Use CAT mode to propagate X in a non-deterministic way and to compare the simulation behavior with that of the real hardware behavior.

Note: When simulating a snapshot that uses X-propagation, the software requires the performance option license feature (Performance_Option_To_Incisive) for successful simulation.

For example, when evaluating X on the design `test.v`, use the following command to trigger CAT mode:

```
% irun -xprop C test.v
```

Or:

```
% ncvlog test.v
% ncelab -xprop C top
```

Alternatively, you can use the `-xfile` option to enable X-propagation on select modules or instances. The `-xfile` option and the `-xprop` option are mutually exclusive. In other words, both options will not work on the same command line. If you place both options on the same command line, then the `-xfile` option will take priority and the `-xprop` option will be ignored.

To generate a log of X-propagation messages, use the `-xverbose` option.

See Using X-propagation to Solve X-optimism for more details.

1.2.263 -XVerbose

(Verilog only)

Enable reporting of X-propagation information during elaboration.

This option saves messages about X-propagation to the `xp_elab.log` file. At the command-line, use the `-xverbose` option to receive notification as concerns those blocks which have enabled X-propagation.

1.2.264 -Zlib compression_level

Compress the `.pak` file.

When you compile, elaborate, and simulate a design, the tools create or modify intermediate objects. All intermediate objects that are required by the NC tools are stored in a single database file in a library directory. This library database file is called `inca.`*`architecture.lib_version`*`.pak`. For example, the name of the library database file is similar to the following:

```
inca.sun4v.132.pak
```

For a large design, the `.pak` file can consume a significant amount of disk space. Use the `-zlib` option to compress the `.pak` file before it is written to disk.

The `-zlib` option is supported for the following tools:

- Verilog and VHDL parsers (*ncvlog* and *ncvhdl*)
- The SystemC *ncsc* utility
- The elaborator (*ncelab*)
- The simulator (*ncsim*)

If you are simulating in single-step mode with *irun*, the `-zlib` option is automatically passed to all appropriate tools.

The level of compression can be set from 1 to 9. For example:

```
% ncelab -zlib 1 ....
% irun -zlib 7 ....
```

A higher number results in a more highly compressed file, but performance can decrease because the tools must uncompress the file before reading it.

If no compression level is specified, a warning is issued and level 1 is used.

1.3 Example ncelab Command Lines

The following command includes the `-messages` option, which prints elaborator messages.

```
% ncelab -messages top
```

The following example includes the `-logfile` option, which renames the log file from `ncelab.log` to `top_elab.log`.

```
% ncelab -messages -logfile top_elab.log top
```

In the following example, `-errormax 10` tells the elaborator to abort after 10 errors.

```
% ncelab -messages -errormax 10 top
```

The following example uses the `-file` option to include a file called `ncelab.args`, which contains a set of elaborator command-line options.

```
% ncelab -file ncelab.args top
```

The following example uses the `-snapshot` option to name the snapshot `topsnap`. When the simulator is invoked, this name should be used with the `ncsim` command.

```
% ncelab top -snapshot topsnap
```

In the following example, `-nowarn` is used to suppress the printing of a specific error message. The argument to the option is the mnemonic for the message.

```
% ncelab top
b_2bit_adder under_test (sum, c_out, bus_a, bus_b, c_in);
                         |
ncelab: *E,CUVWLP (2bit_adder_test.v,7|22): Too many module port connections.
% ncelab -nowarn CUVWLP top
```

The following example uses the `-sdf_cmd_file` option to specify an SDF command file called `dcache_sdf.cmd`. Using this option overrides the automatic SDF annotation to Verilog portions of the design. The command file contains commands that control SDF annotation. See "SDF Timing Annotation" for details on SDF annotation.

```
% ncelab -messages -sdf_cmd_file dcache_sdf.cmd top
```

1.4 Timing Delays and Race Conditions in Gate-Level Netlists

The simulation of a gate-level netlist with no timing delays is prone to race conditions. This typically occurs when simulating with no SDF delay annotation or when using a zero delay option (`-delay_mode_zero`). Designs with test scan chain circuitry are particularly susceptible to this. With no proper clock delay balancing, the clock and data signals can propagate instantaneously, making the data from one stage available at downstream stages before the appropriate clock edge is available.

1.4.1 Using Delay Options to Prevent Race Conditions

The following delay options can help you avoid race conditions caused by zero timing delays:

- The seq_udp_delay Option—Applies a delay value to the input/output paths of sequential UDPs.
- The add_seq_delay Option—Applies a delay value to the input/output paths of undelayed sequential UDPs.
- The -delta_sequdp_delay Option—Adds a delta delay to the input/output paths of sequential UDPs.
- The -sequdp_nba_delay Option—Adds a nonblocking delta delay to the input/output paths of sequential UDPs.

1.4.1.1 The seq_udp_delay Option

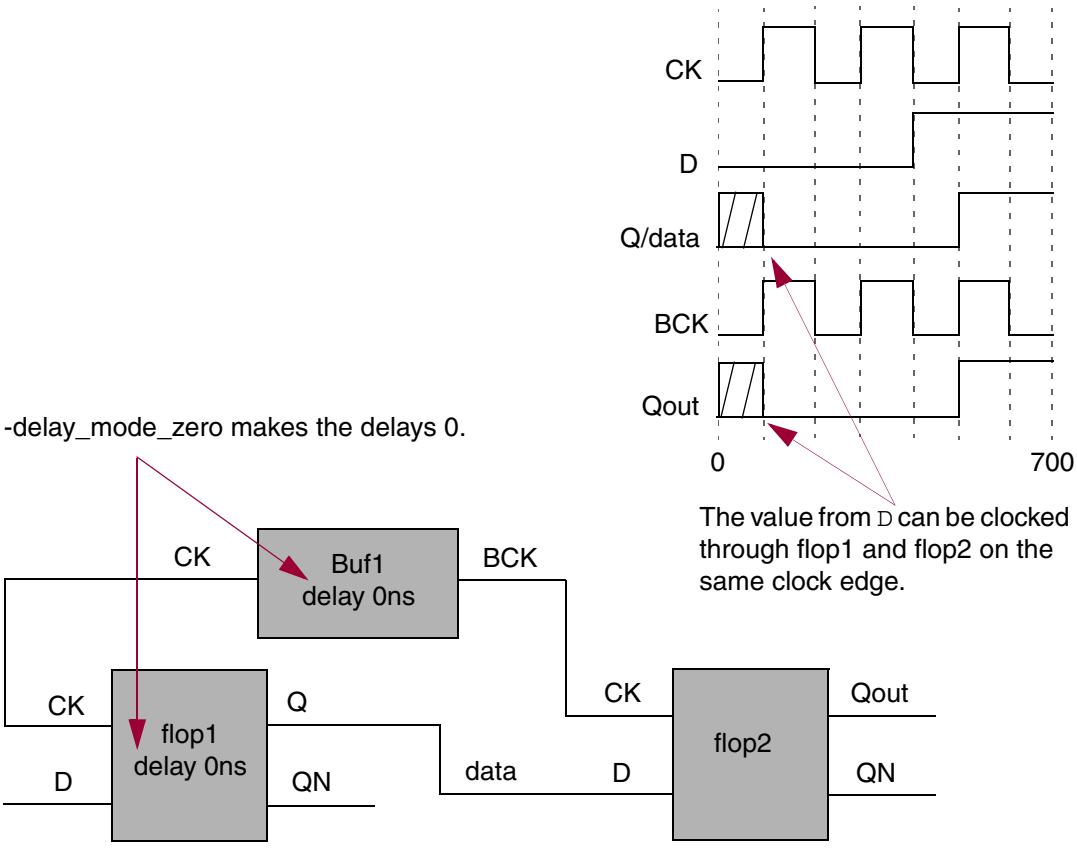
Use the `-seq_udp_delay` option with a delay specification value to set all delays to zero (as if you are using the `-delay_mode_zero` option) except for sequential UDPs.

The delay specification value overrides any delay specified for the sequential UDPs in the design in `specify` blocks, through SDF annotation, and so on. The option also removes any timing checks associated with the sequential UDPs.

The shift register example in Figure 1-1 on page 174 illustrates the problem of race conditions occurring in a gate-level netlist with a zero delay. Figure 1-2 on page 175 shows how the `-seq_udp_delay` option avoids the race.

Figure 1-1 No Timing Delay Creates a Race Condition

When `-delay_mode_zero` is applied, the delays from `CK` to `BCK` and from `CK` to `Q` are zero. This causes `BCK` and `data` to transition at the same time, and potentially allows the changed value of `data` to also be seen by flop2 on the same clock edge.

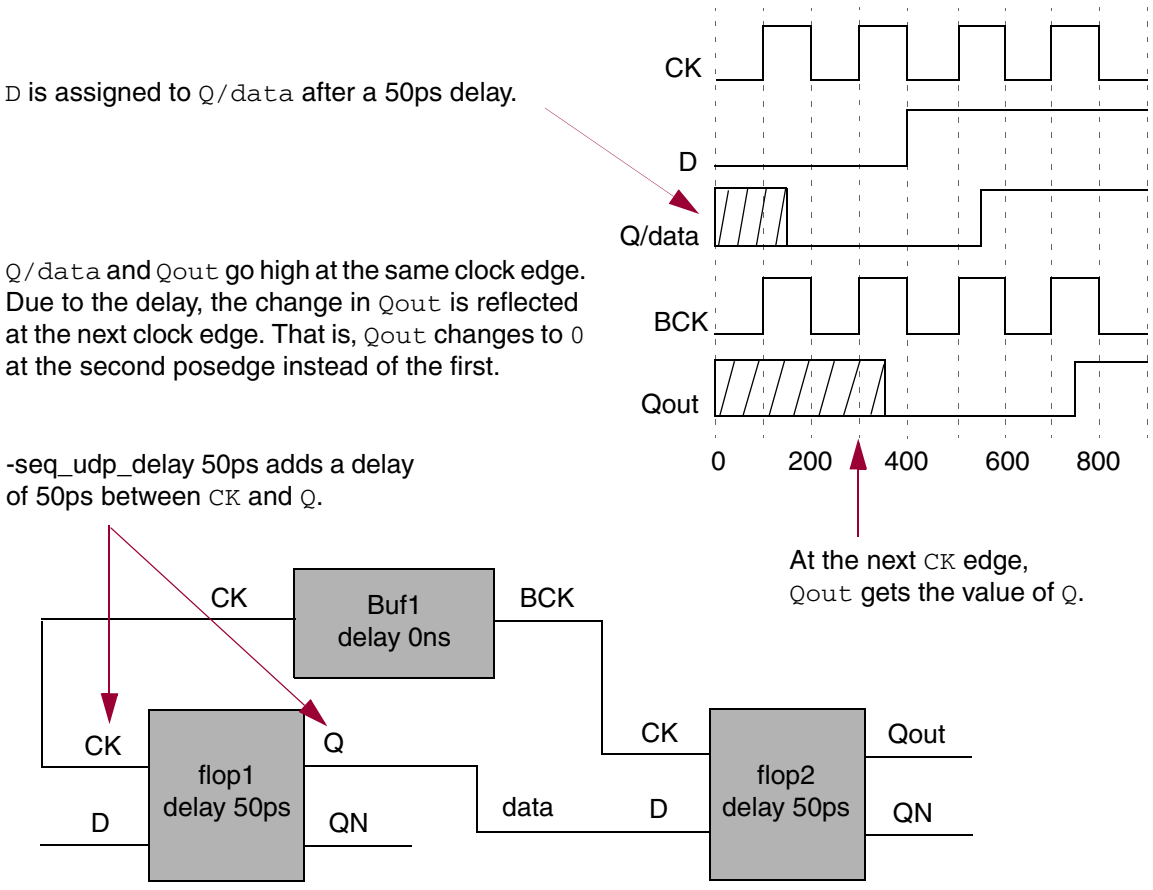


The following figure illustrates the effect of using `-seq_udp_delay` with a delay specification of `50ps`, for the shift register example in Figure 1-1 on page 174.

Figure 1-2 Setting a Timing Delay on Sequential UDPs Avoids a Race Condition

-seq_udp_delay 50ps

When `-seq_udp_delay` is applied, it adds a delay from `CK` to `Q` (50ps in this example). This causes `data` to transition 50ps after `CK` and `BCK`, and causes the change in `data` to be seen by flop2 on the next clock edge.



1.4.1.2 The add_seq_delay Option

The `-add seq delay` option updates undelayed sequential UDPs with a specific delay value. The option applies the delay to only those sequential UDPs that don't have a path delay already defined in the instantiation.

Unlike `-seq udp delay`, this option preserves all other delay values in the design, such as module path delays, specify blocks, SDF annotation, and so on. This option is useful for avoiding race conditions, but comes at the expense of performance improvements that can be gained with `-seq_udp_delay`, which removes all other delays and timing checks.

Combining -add_seq_delay With –seq_udp_delay

You can combine the `-add_seq_delay` option with the `-seq_udp_delay` option. In this case, the instance-specific `-add_seq_delay` value overrides the delay value applied with `-seq_udp_delay` (it does not add to the prior delay).

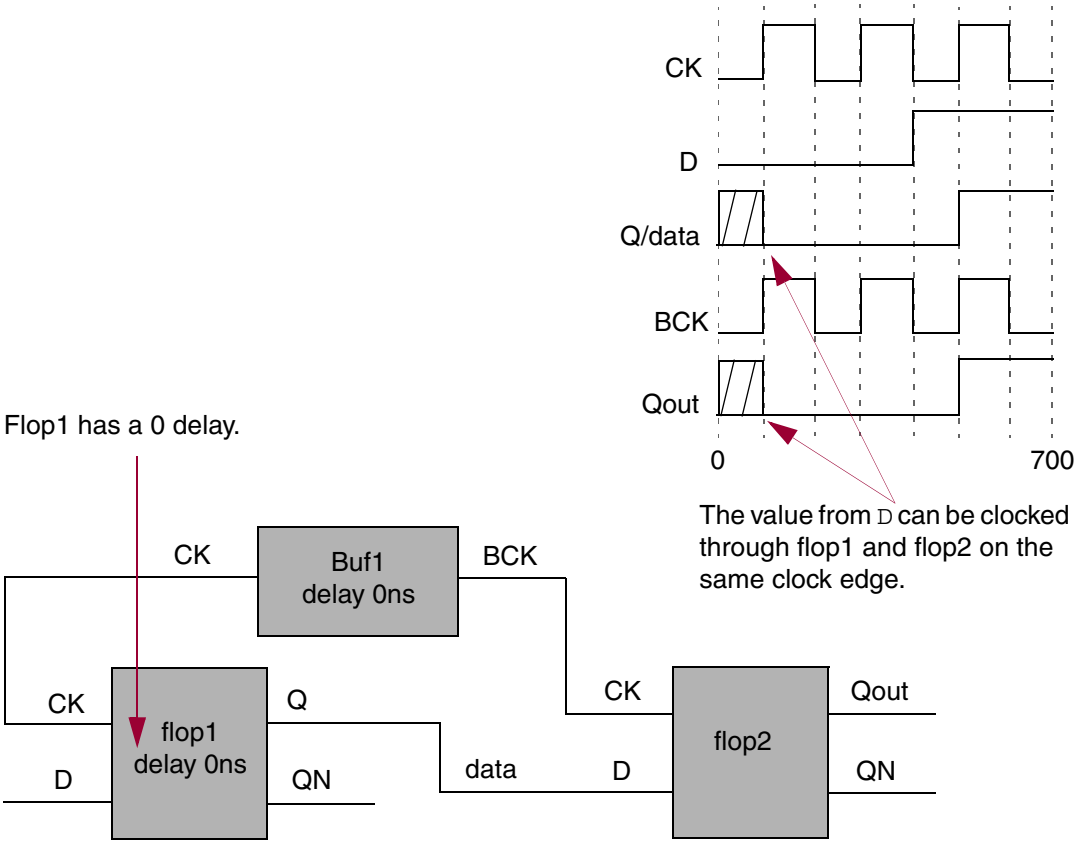
For example, in the following line, `-add_seq_delay` applies a `40ps` delay to the instance `top.dut_top.u3`, while `-seq_udp_delay` applies a `20ns` delay to all other UDPs:

```
-seq_udp_delay 20ns -add_seq_delay top.dut_top.u3=40ps
```

The shift register example in Figure 1-3 on page 176 illustrates a race condition occurring in a flip-flop with a zero delay. Figure 1-4 on page 177 shows how applying `-seq_udp_delay` with a specific delay value avoids the race.

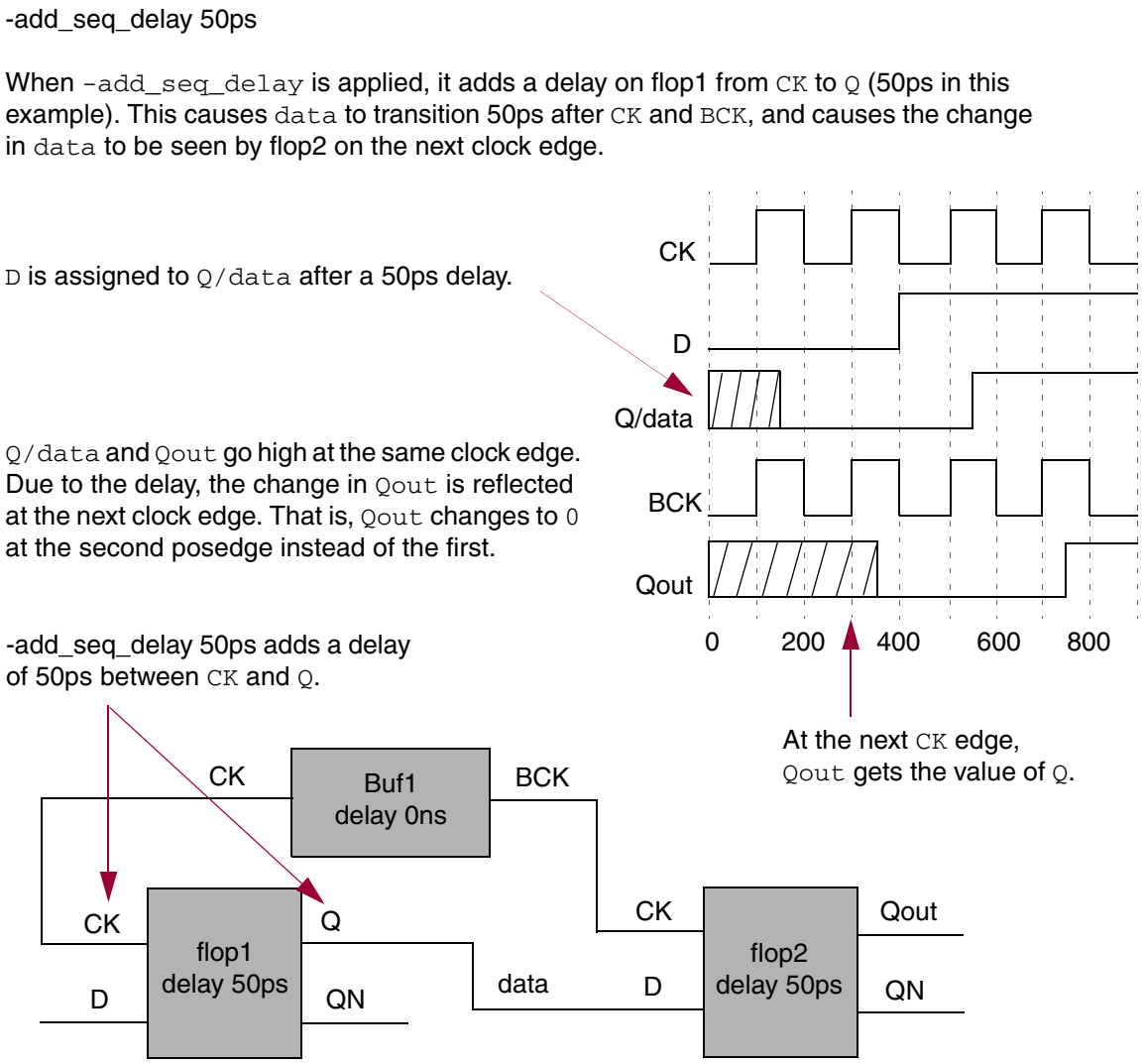
Figure 1-3 No Delay On a Sequential UDP Creates a Race Condition

The delays from `CK` to `Q` and from `CK` to `BCK` are zero. This causes `BCK` and `data` to transition at the same time, and potentially allows the changed value of `data` to also be seen by flop2 on the same clock edge.



The following figure illustrates the effect of using `-add_seq_delay` with a delay specification of `50ps`, for the shift register example in Figure 1-3 on page 176.

Figure 1-4 Adding a Sequential Delay Avoids a Race Condition



1.4.1.3 The -delta_sequdp_delay Option

The `-delta_sequdp_delay` option adds a delta delay to sequential UDPs as a way to settle the flow of logic value transitions being transferred through each logic stage (such as scan chains). A delta delay provides a minimum delay when sequential UDPs do not include explicitly specified delays.

1.4.1.4 The -sequdp_nba_delay Option

The -sequdp_nba_delay option adds a nonblocking delta delay to sequential UDPs at the end of the simulation cycle, after all values have settled and when nonblocking assignments are evaluated. As with the -delta_sequdp_delay option, a nonblocking delta delay provides a minimum delay when sequential UDPs do not include explicitly specified delays.

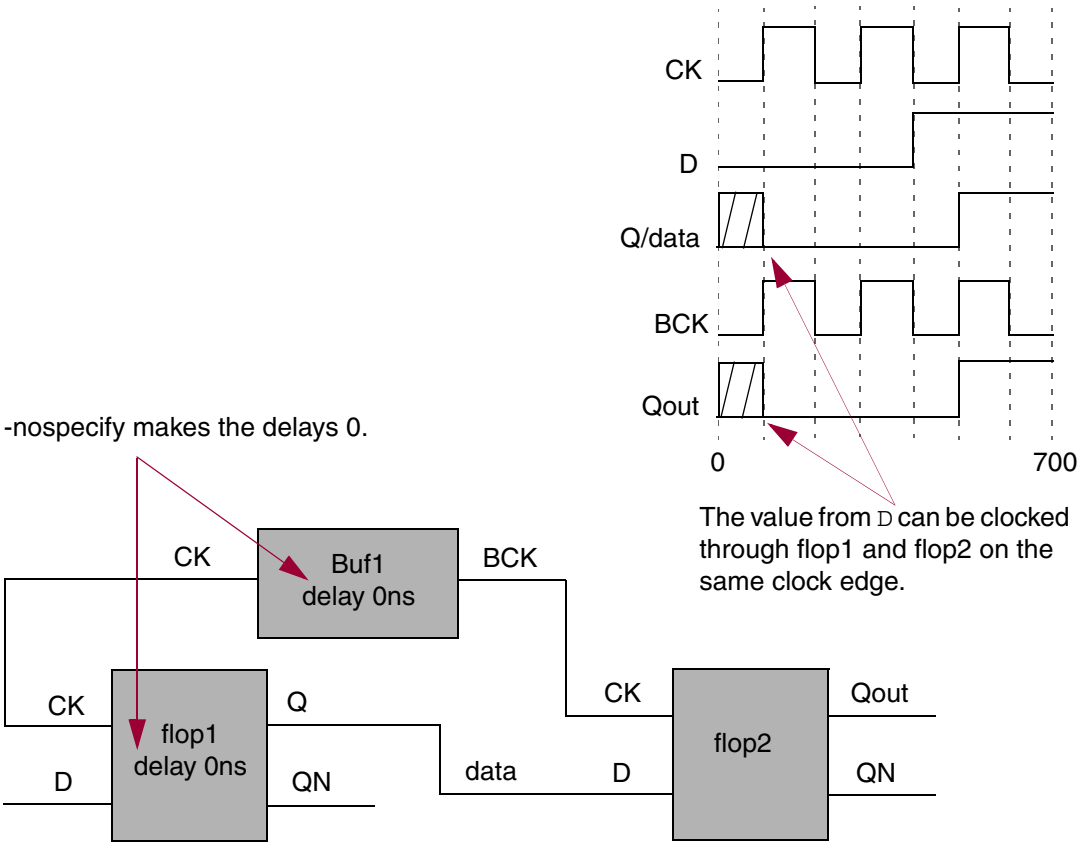
Using Delta Delays to Avoid a Race Condition

Both -delta_sequdp_delay and -sequdp_nba_delay are useful when running zero delay simulations with the -nospecify option (which disables SDF annotation and the timing features of specify blocks). Adding a delta delay to sequential UDPs allows the combinational logic to settle before the simulator updates the output of the sequential logic.

The shift register example in Figure 1-5 on page 179 illustrates the problem of a race condition occurring in a gate-level netlist simulating with the -nospecify option. Figure 1-6 on page 180 shows how the -delta_sequdp_delay option avoids the race. Figure 1-7 on page 181 shows how the -sequdp_nba_delay option avoids the race.

Figure 1-5 Simulating With -nospecify Creates a Race Condition

When `-nospecify` is used, the delays from `CK` to `BCK` and from `CK` to `Q` are zero. This causes `BCK` and `data` to transition at the same time, and potentially allows the changed value of `data` to also be seen by flop2 on the same clock edge.



The following figure illustrates the effect of using `-delta_sequdp_delay` for the shift register example in Figure 1-5 on page 179.

Figure 1-6 Specifying a Delta Delay Avoids a Race Condition

-delta_sequdp_delay

When `-delta_sequdp_delay` is applied, it adds a a minimum delay from `CK` to `Q`. The delay updates the output of flop1 and causes data to transition after `CK` and `BCK`, which lets the change in data to be seen by flop2 on the next clock edge.

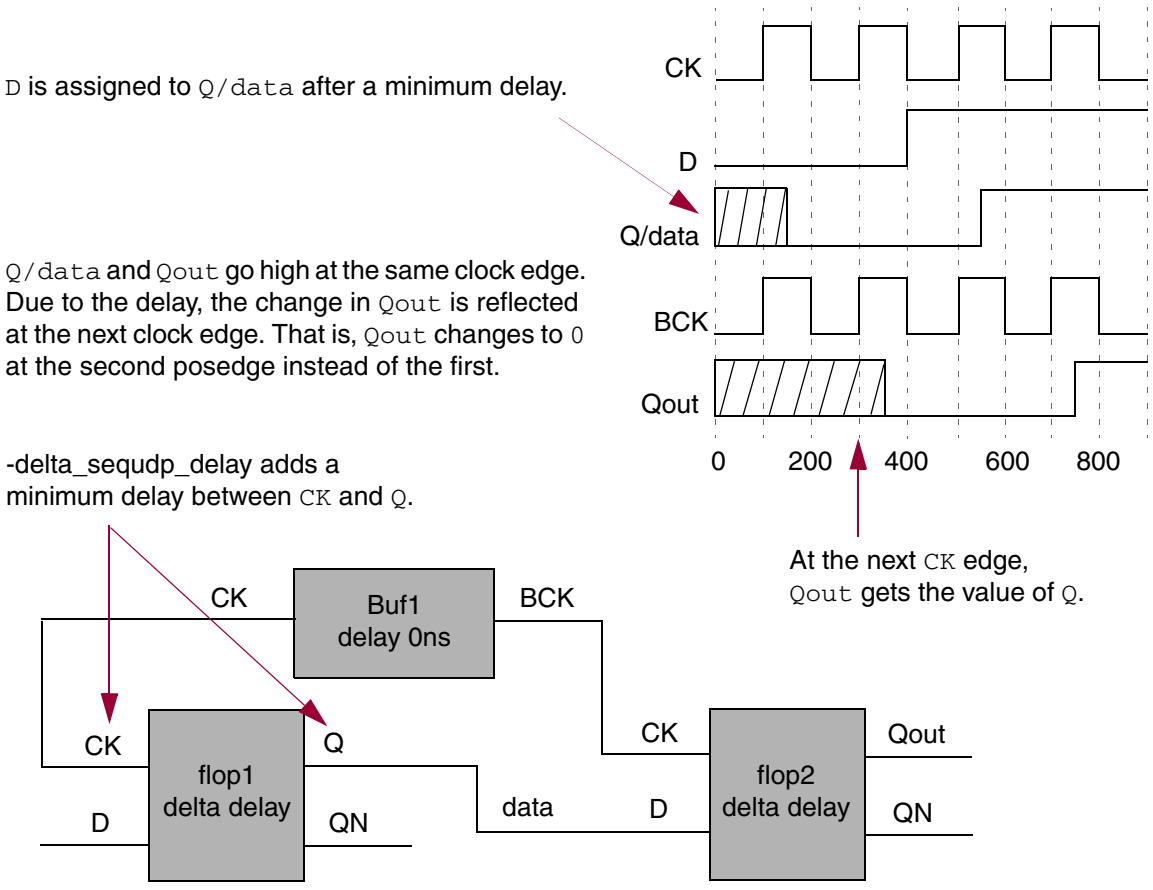


Figure 1-7 on page 181 illustrates the effect of using `-sequdp_nba_delay` for the same shift register example.

Figure 1-7 Specifying a Nonblocking Delta Delay Avoids a Race Condition

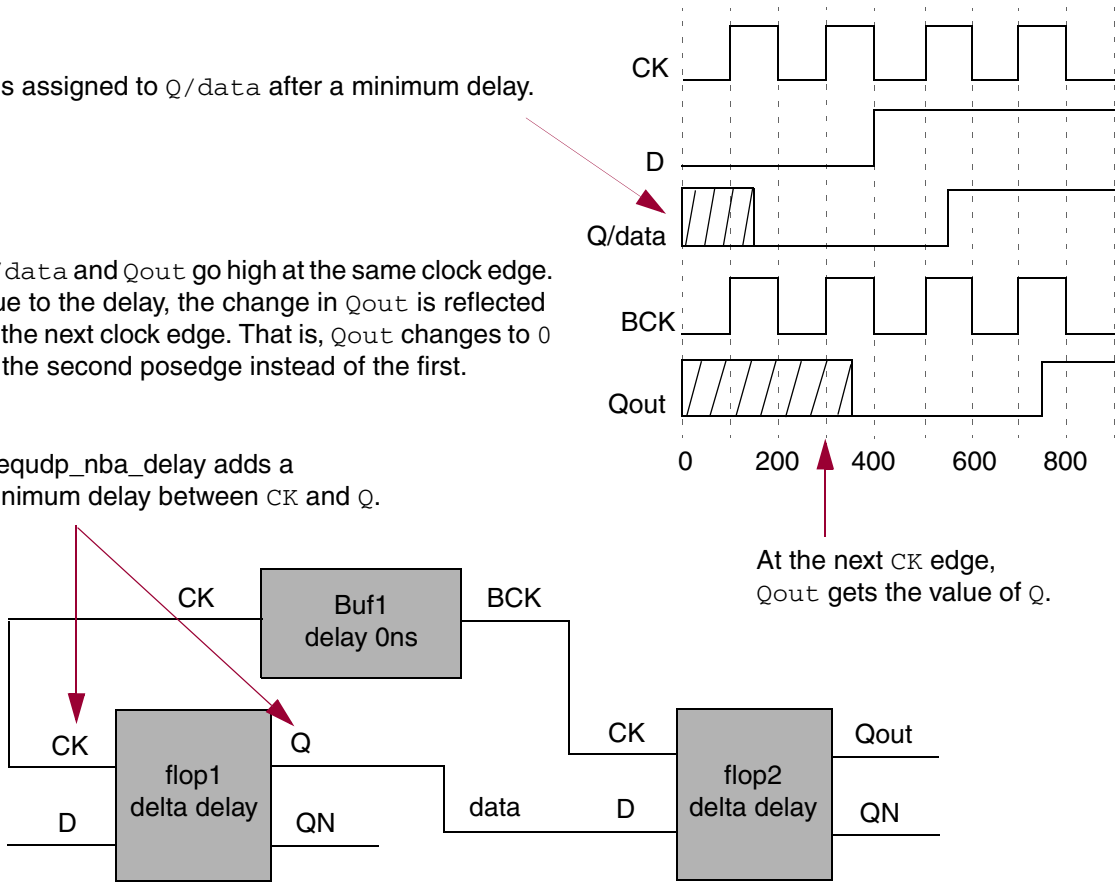
-sequdp_nba_delay

When `-sequdp_nba_delay` is applied, it adds a a minimum delay from `CK` to `Q`. The delay updates the output of flop1 and causes data to transition after `CK` and `BCK`, which lets the change in data to be seen by flop2 on the next clock edge.

`D` is assigned to `Q/data` after a minimum delay.

`Q/data` and `Qout` go high at the same clock edge. Due to the delay, the change in `Qout` is reflected at the next clock edge. That is, `Qout` changes to `0` at the second posedge instead of the first.

-sequdp_nba_delay adds a minimum delay between `CK` and `Q`.



Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z