ESP Cov
User Guide

Version Z-2006.12, June 2010

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HAPS, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes; known problems and limitations; and resolved Synopsys Technical Action Requests (STARs) is available in the *ESP-CV Release Notes* in SolvNet.

To see the *ESP-CV Release Notes,*

1. Go to https://solvnet.synopsys.com/ReleaseNotes. (If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)
2. Click ESP-CV, then click the release you want in the list that appears at the bottom.

About This Guide

This guide explains the symbolic coverage that you can use with ESP-CV. It enables you to evaluate and use symbolic coverage and run reports.

Audience

The symbolic coverage utility of ESP-CV is intended for design engineers who use the ESP-CV product and need to determine how much of their design is being covered by symbols.

Related Publications

For additional information about ESP-CV, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys electronic software transfer (EST) system
- Documentation on the Web, which is available through SolvNet at http://solvnet.synopsys.com/DocsOnWeb
- The Synopsys MediaDocs Shop, from which you can order printed copies of some Synopsys documents, at http://mediadocs.synopsys.com

You might also want to refer to the documentation for the following related Synopsys product:

- Formality

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as `pin1 [pin2 ... pinN]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at https://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages.

1

Overview

When doing symbolic simulation, you might have questions about coverage and symbolic simulation, such as

- How do I ensure that a symbolic simulation is doing anything and that reasonable checks are being made? How to avoid false positives?
- How do I ensure that I'm running enough cycles and conversely, that I'm not waiting for more cycles to complete than are required?
- How do I know that I'm not overconstraining the testbench and what are the effects of the constraints that are being provided?

This document shows how you can use this coverage tool and how you can interpret the reports to determine if you have answered these questions. But before answering the preceding questions, the sections that follow introduce some concepts and coverage metrics pertaining to symbolic simulation:

Concepts

A symbolic simulator like ESP-CV is different from standard Verilog simulations in that every net can have equations associated with it besides the fixed values of 0, 1, X or Z you would normally expect.

There are four coverage metrics that the ESP coverage tool reports on. These are

- Propagated or dropped symbol coverage

  This measures the percentage of symbols that go into a scope but do not come out. The idea is that no symbols should be dropped without some investigation. In a proper set of tests, this metric should be near 100 percent. The measurement is a ratio of the number of symbols propagated divided by the total of the propagated symbols and dropped symbols.

- Symbolic net coverage

  This is the percent of all nets that received symbols at any time during the simulation.

- Toggle coverage (or stuck-at coverage)

  This is the percent of all nets that achieved both a 0 and a 1 value at any point during the simulation. Any nets that never achieve either 0 or 1, or both 0 and 1 values during the simulation are assumed nontoggled or stuck-at and reported under this coverage metric.

- Line coverage

  This is the percent of all lines of design code that were touched and executed by the symbolic simulation. This metric is more meaningful for RTL-level code than gate-level code.

Collectively these metrics provide information about the effectiveness of the symbolic testbenches.

While toggle and line coverage are used extensively in binary simulation coverage, the propagated and symbolic net coverage have been introduced specifically to address the coverage problem in the domain of symbolic simulation.

Terminology

In practice, the mnemonics used represent the negation of the ESP metrics to report the coverage results (except line coverage).

Specifically, DS (Dropped Symbols) represents the propagated and dropped symbol coverage, NS (Non Symbolic) represents symbolic net coverage, SA (Stuck-At) represents toggle coverage, and LC (Line Coverage) represents line coverage information. In the coverage output, there are a number of symbolic statistics that use the following terms:

- Nonsymbolic nets

  This is a list of nets that never had symbols. These nets always had binary content (0,1,X,z) and possibly were stuck at one value).

- Propagated symbols

  A list of symbols that were driven out of a particular scope. The scope might have bidirectional nets and using the word driven, rather than output, shows that the metric accounts for directionality.

- Dropped symbols

  Similar to propagated symbols, this is the list of the symbols that entered but did not come out of the specified scope.

The use of symbolic coverage does not guarantee that everything is 100 percent covered. It can tell you what has not been covered. These are not the same assertion or even an inverse function of the other. Coverage is a summation of simulation results over the time of simulation. Here are a few examples of its shortcomings:

1. If at any point a net has a symbol, it is considered as symbolic. It does not matter whether the symbolic value propagated to the output to be observed.
2. A net could have a symbolic value for only one of the three symbolic cycles. However, the net is still reported as a symbolic net.

Thus, symbolic coverage (and coverage in general) must be used as a negative check to ensure that the testbenches in question are not lacking significantly in their ability to verify the design. In other words, high coverage numbers do not imply absence of bugs in a design but only that the testbench in question is good. On the other hand, if coverage metrics report low numbers, it implies that the testbench needs improvement.

How Symbolic Coverage Helps

Having defined the coverage metrics, the questions raised in the introduction can now be answered. You need to answer these questions to use a coverage tool effectively.

How do the reports show that symbolic simulation is doing anything?

1. Certainly having symbols going in but not coming out is a bad sign. Dropped symbols at the top level of a design reports such situations.

2. If you use ESP-CV, the $display functions that are already part of an ESP TBGen generated testbench should identify when nothing but X's are coming out. This indicates a gross timing error in the testbench or that the testbench or design have a serious bug in it.
3. If the symbolic simulation is slightly active for the design under test, there might be large parts of the design that do not take on symbolic values. In this case, symbolic net coverage numbers help to show that the simulation is deficient (or the design is incorrect).

How do the reports show if enough cycles have been run?

Dropped symbols should help here as well. This question comes up most frequently on pipelined designs and datapath designs where there is uncertainty on how many cycles should be run. By looking at the dropped symbols, it should be evident if some of the cycles haven't come out yet, and need more flush cycles to propagate them out. (A flush cycle is a stimulus cycle in which the input stimulus is kept binary to allow the design an extra cycle to propagate interesting values to the output.)

Symbolic nets might help here as well, but to a much lesser degree. A net is considered symbolic even if it gets a symbolic value in one (out of many) symbolic cycles. Thus if no symbols have ever reached an area of a design or a stage of the pipeline, then this metric can be useful. But in practice, this metric is less useful than propagated or dropped symbols.

How do the reports show if the design is over-constrained?

Dropped symbols might help here, but the bulk of this issue is addressed by the report on nonsymbolic nets. Dropped symbols can help because an over-constrained design swallow symbols and they are not propagated through the design. However, they might not enter the design in the first place. Therefore, they are not dropped. This can happen if the testbench constrains the input stimulus and kills certain symbols before they enter the design.

Symbolic nets help the most here in that an over-constrained design have large parts of the design that do not get symbolic values. Toggle and line coverage can be secondary indicators to help identify regions of low activity due to extra constraints.

2

Usage

This chapter describes the basic usage flow for running coverage. It includes the following sections:

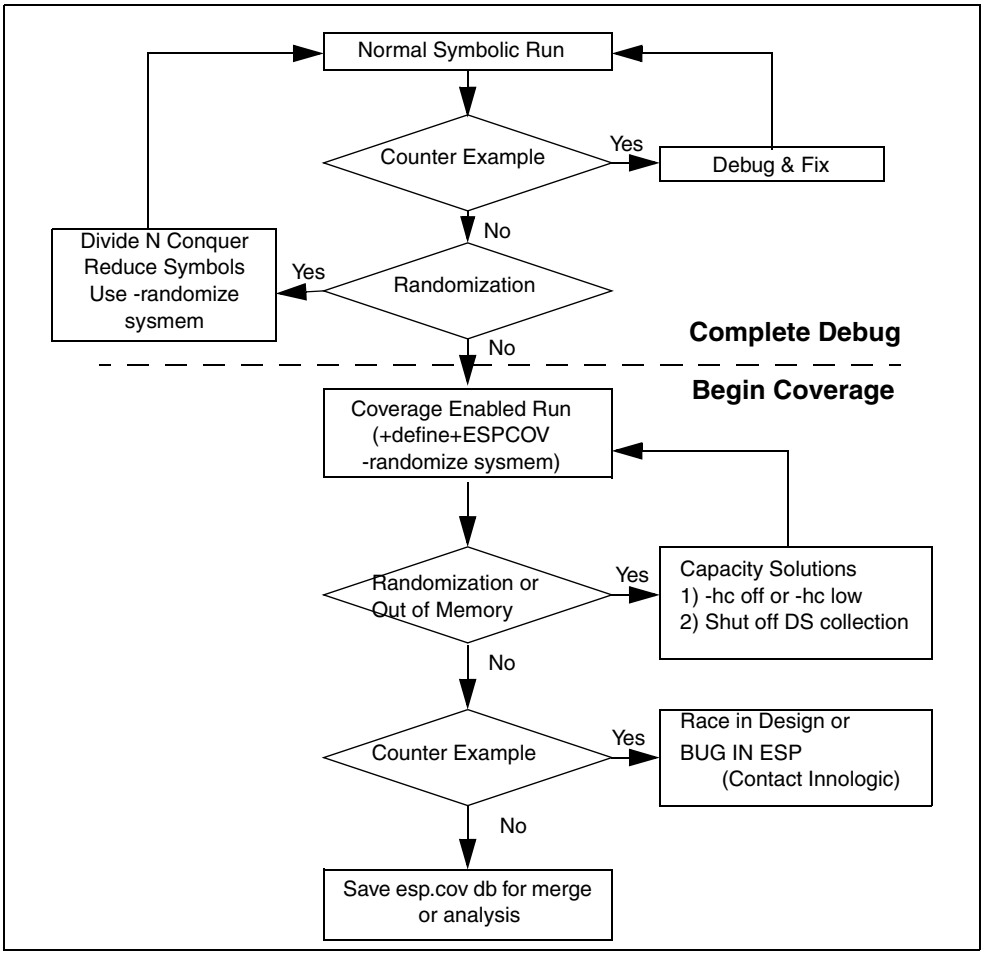
Coverage Flow

Symbolic coverage analysis is done in two steps. The first step generates raw coverage data in a database file during simulation using ESP-CV or ESP-LV. The information collected can be controlled by the $espcovscope PLI function. Typically, you would dump data for only the behavioral (ref) and transistor (xtor) models.

The second step generates a summary report on the data collected in the first step using the ESP Cov utility. Multiple simulation results can be merged during this step to generate the summary report. The amount of information reported is by default only the top level information, but a specification (spec) file can be used to request more detailed information to be dumped.

It is expected that prior to running coverage, you have simulated a testbench without detecting any errors and without having randomization in the simulation results. Either of these causes the coverage results to be invalid or severely handicapped.

Figure 2-1 ESP-COV Symbolic Coverage Flow



The usage model for running symbolic coverage is as follows:

1. Make a noncoverage recording run to ensure that no errors are detected and randomization has not occurred. If randomization occurred and you still want to continue, then the randomize information in the esp.Random file is required. For more information about randomization related issues, see the *ESP-CV User Guide*.
2. Make a coverage run with `-randomize sysmem` so that new randomization does not occur and if randomization occurs in step 1, use `-r` to force all of the symbols randomized in step 1 to a binary value.
3. Merge any coverage databases of multiple simulation runs to create the final coverage database using ESP CovMerge.
4. Use `espcov` to generate a coverage report from the final coverage database.

This usage model achieves three things:

1. Randomized variables are treated pessimistically for coverage because the -r option fixes them to binary values at time 0. Therefore, these variables do not contribute to symbolic nets and propagated symbols. They are neither reported as dropped nor propagated.
2. It reduces the chances of capacity inconsistency between coverage and normal runs by allowing the coverage run extra slack (`-randomize sysmem`).
3. If multiple testbenches are required, the coverage results are merged so that one consolidated coverage report can be generated.

Enabling Coverage in Testbenches

Coverage is turned off by default. To enable coverage use the +define+ESPCOV command-line argument while running espcv. This enables espcv to execute the Verilog code inside the `` `ifdef ...`endif `` section. See the example section that follows on how this would work.

First add the code in the following example to your Verilog testbench at the top level.

This example assumes that you used the esptbgen generated testbench you created and have a reference model instantiated as “ref” and an implementation switch-level model instantiated as “xtor”. Use the appropriate instance names for your design if you are using your own testbench. For more information about how to use this statement, see “$espcovscope” on page 3-3.

If you are adding this code to the generated testbench, you could place this code just after the code generated for creating VCD files.

```
`ifdef ESPCOV
initial begin
      $espcovfile(“esp.cov”);
      $espcovscope(0,xtor);//Collect coverage info for xtor
instance
      $espcovscope(0,ref); //Collect coverage info for ref
instance
end
`endif
```

The `$espcovscope(0,xtor)` statement gathers information about the transistor switch-level implementation model and `$espcovscope(0,ref)` gathers information about the behavioral reference model.

Running Coverage Simulation

Now run the normal symbolic simulation and enable your coverage commands by adding the `+define` on the command line.

```
espcv ..... +define+ESPCOV -randomize sysmem
```

This generates coverage information in the file (`esp.cov`) specified in the `$espcovfile` statement. The database in this file can then be viewed with the `espcov` program.

It is advised that `-randomize sysmem` be added to the `espcv` command to allow for the extra capacity constraints imposed by collecting coverage information. Otherwise, randomization can happen, which invalidates the coverage results.

Merging Coverage Data From Multiple Simulations

Merging of multiple coverage simulation results is done by using the `espcovmerge` command to create a new database file. You then use `espcov` to report coverage information about this new consolidated database.

The rules that you must follow are

1. The models referenced by the $espcovscope instance path must be identical for all simulations generating coverage data. This includes code optionally included using command-line +define options. The same `+define` should be used on all simulations.
2. For the simple merge used here, the top-level testbench module name must be identical and all the symbol names used for each test must be identical. For example, `A_sym1`, `A_sym2`, `A_sym3` are used in all testbenches for the same A input pin.
3. If used, the `-hc` command-line option must be consistent across all simulations being merged. For example use `-hc high` (the default) across all simulations to be merged.

A simple merge command is

```
espcovmerge db1.cov db2.cov -db
      esp.cov -log espmerge.log
```

This example shows how a simple merge of results can be done. In the previous simulation runs, the coverage database files `db1.cov` and `db2.cov` were created. They could have been given those names in the `$espcovfile` command or even more simply been renamed after the simulation completed. In this case, the merged output file is called `esp.cov` and you have directed the log file output to the file `espmerge.log`.

An optional merge specification file can be used to identify equivalent instance and module names in multiple simulation runs. However, if your module names remain the same in all the testbenches used, you do not have to use the specification file. The specification file commands for merging are documented in section “Merge Specification File Commands.”

Use the `-h` command-line option for each coverage utility to obtain help on any of the available command-line options. Subsequent sections in this document also explain the options available for the ESP Cov utilities.

Reporting Coverage Results

After simulation (or database merge) is finished, you can examine coverage results in the file `esp.cov` by using the `espcov esp.cov` command.

You should now have a report file, called `espcov.log` by default, created in the local directory showing each type of coverage information that was gathered. By default, the report shows only top-level summary information for each `$espcovscope` specified in the coverage simulation (`xtor` and `ref` in the preceding example). To obtain more information about each of the coverage types in the output report file, use the `-printdetail` command-line option.

```
espcov -printdetail “NS DS SA LC” esp.cov -log espcov.log

           all        Use in lieu of specifying all coverage
types
                      on the command line (i.e. printdetail-all
           NS         Nonsymbolic
           DS         Dropped Symbols and propagated symbols
           SA         Stuck At binary values and toggle coverage
           LC         Line Coverage
```

The detailed report file reports the name and number of symbols that entered and exited each scope, thereby determining the amount of coverage that was achieved. It also reports symbols that entered a scope but did not exit as dropped.

If you use `-hc off`, it also reports which nets had symbols coming in, but did not get propagated to the output (dropped symbols). When you use hierarchical compression, this information cannot be generated.

You can specify a specification file using `espcov -spec` to control the information that is to be reported. You can specify different modules to report on and how many levels to report.

3

Verilog PLI Statements

You can control ESP coverage results in a manner similar to the VCD dump PLI interface. There are four ESP coverage PLI statements that control the generation of the coverage database. These statements are placed within an initial statement and can dynamically control the generation of the coverage data. For more information about methods to control coverage in your testbench, see “Enabling Coverage in Testbenches” on page 2-4.

This chapter includes the following sections:

$espcovfile

The `$espcovfile` statement is used to specify the storage file for the coverage information and to specify the type of coverage information desired. If the coverage database file is not specified, a default value of `esp.cov` is used. If a list of coverage types is not specified, all types are generated. The reason for allowing specification of coverage types is to reduce the amount of information dumped and increase the simulation capacity while capturing coverage information.

Only one file specification is allowed. If more than one is encountered during simulation, a runtime warning message is issued and the remaining coverage file specifications are ignored.

Usage: `$espcovfile( "<espcovfile>" [, "<listofcovtypes>" ] ) ;`
`<listofcovtypes> ::= <covtype>+`
`<covtype ::= 'NS' | 'DS' | 'SA' | 'LC'`

Examples of different formats available are

```
$espcovfile(); // Default file esp.cov
$espcovfile; // Default file esp.cov
$espcovfile("abc.cov"); // Specify file abc.cov
$espcovfile("abc.cov", "NS DS SA LC");
$espcovfile("abc,cov", "DS" );
```

Coverage types are

- NS - Nonsymbolic

  Signals that have never been assigned a symbolic value during the simulation.

- DS - Dropped symbols and propagated symbols

  Symbols that have entered the specified scope, but that have not been observed on the outputs of that scope.

- SA - Stuck-at-binary values and toggle coverage

  Signals that never go to a 0 or 1 value during the simulation.

- LC - Line coverage

  List of Verilog lines that have not been executed.

$espcovscope

You can use another PLI statement, `$espcovscope`, to limit the scope for which coverage information is gathered. If no scope is specified, coverage is gathered for all of the design. The specification of scope follows the same rules as the `$dumpvars` statement. The first number determines the number of levels that coverage data is gathered for. The second parameter is the hierarchical path to a module that the coverage data is gathered from.

Usage: `$espcovscope [ ( [<level> , <instancepath>] ) ] ;`
`<instancepath> ::= [ <modulename> '.' ]`
`<instanceid> [ '.' <instanceid> ]`

Specifying Scope Level

An integer value of 0 means that all levels starting at the specified hierarchical path are gathered. If no hierarchical path is specified, the scope starts at the top level of the design.

A level value of 2 gathers data for the specified scope and its children. Information about the lower levels is not gathered. The typical value for levels is 0, which gathers all information below the specified scope. A nonzero level is used only if the information being gathered is too large.

Defining Instance Path

In the context of `$espcovscope`, the instance path can be absolute or relative. For a relative instance path name, the first instance identifier is the name of an instance in the module where the `$espcovscope` is located. An absolute instance path name is the complete hierarchical name of the instance starting from the top level. For a complete definition of instance paths applicable in general context, see Appendix A, "Coverage Syntax Summary."

Typical Usage in a Testbench

Some examples of `$espcovscope` statements are

```
$espcovscope; // Like dumpvars,dumps all levels
// of design
$espcovscope(); // Again, all levels are dumped.
$espcovscope(0, a.b.c); // Dumps all levels starting at
                        // instance a.b.c
$espcovscope(2,ref); // Dumps only the instance ref and
// its children and nothing below that.
$espcovscope(0,xtor);// Dumps every level below the
                     // instance xtor
```

The typical usage of `$espcovscope` in a testbench is that the specified scope is the `ref` and `xtor` model instances, but not the testbench. This is shown by the following example:

```
$espcovscope(0,xtor);
$espcovscope(0,ref);
```

If overlapping scopes are present in two different testbenches, you can have problems merging the coverage data. Overlapping scope is defined to be the use of an instance path in one `$espcovscope` that causes coverage data to be gathered on nets that are being covered by an instance path used in another `$espcovscope` statement. Overlapping scope is not intended to happen in the normal operation and you are strongly encouraged not to use it.

In the following example, only the top three levels in one simulation are dumped, along with all the levels below one particular instance at the third level down in another simulation. The coverage results cannot be merged and the following error is issued: “Scopes in all files should not have ancestor-offspring relationships.”

```
$espcovscope(3,xtor);            // In testbench 1
$espcovscope(0,xtor.a.b); // In testbench 2
```

You must use identical scopes in simulation runs where you want to merge coverage data. In the aforementioned case, you could look at coverage data from separate runs, but you still cannot merge the results.

$espcovoff, $espcovon

These two statements allow you to turn coverage recording on and off dynamically in your Verilog testbench.

Usage:`$espcovon ;`
`$espcovoff ;`

Using these statements can affect the accuracy of the reported coverage.

These statements do not affect line coverage results. These statements are used infrequently mainly to mitigate the capacity constraints. If the design is large or it runs a large number of cycles, important cycles can be selected for coverage data collection by using the above PLIs.

4

Merging Coverage Data

Merging of coverage databases is required when you run multiple simulations to verify a design and want to know the overall coverage of the design. For example, you might have a protocol verification and a data integrity test, and you want to know how much of the design is covered with both of these tests.

After you have merged all the results together into one coverage database, use `espcov` to view the results.

This chapter includes the following sections:

- Command-Line Options
- Merge Specification File Commands

Note:
The coverage internal database format for the ESP Cov version X-2005.06 release has been upgraded and is not compatible with previous versions of the tool. Therefore, ESP CovMerge does not merge databases created with versions released prior to version X-2005.06. ESP Cov also does not report coverage metrics for databases created with versions prior to that release.

You can use a specification file to identify equivalent instance and module names in multiple simulation runs. The command-line option `-spec` identifies the specification file name.

If you use the same testbench names and hierarchy for all the tests and you use the same symbol-naming convention, you can use *simple merging*. No specification file commands are required to merge the coverage results.

For merge to work, the design that you specified in the instance path of `$espcovscope` must remain identical. If you have two runs with different `` `define `` that instantiate different designs, or modify the design in any way, the merge fails with an error message indicating that the merge cannot be done. When using merge, make sure the same `` `define `` is used in all simulation coverage results that are required to be merged.

To merge hierarchical compression (HC) results, the HC value has to be identical between all simulations that are to be merged.

The usage syntax of `espcovmerge` is as follows:

Usage: `espcovmerge [<options>] [-spec <file>] <coveragedbinputfile>+”`

This command merges all the coverage database files specified on the command line and generates a single file. Merging of different coverage results can be controlled by the specification file when testbench module names are different between the simulation runs.

You can specify the name of the merged coverage database file created. If none is supplied, the default name is `esp.cov`. The merged database file has to be different from any input database files.

Command-Line Options

The command-line options available for `espcovmerge` are

```
-h              Display help information
-spec <file>    Optional coverage specification file. Allows you to
                specify how to merge data under certain conditions
                where testbench names may be different.
-db <file>      Optionally specify a new merged coverage database
                name. If not specified, esp.cov is used.
-l <log>        Specify a log file name different from the default
                value of espcov.log.
```

Sample command lines:

```
espcovmerge cov1.db cov2.db
espcovmerge -spec cov.spec cov1.db cov2.db -db newcov.db
```

Merge Specification File Commands

The following sections describe the commands for setting the merge specifications.

Comments

Comments start with the character `#` in column 1. For example,

```
#Comment line
```

Map

If you have used different top-level module names in the testbenches that are used to verify the design, use this command to merge the results. The design under test must be the same in both tests. The dumpscope used should be the same fully qualified instance path specified in each of the `$espcovscope` statements in the testbenches whose coverage results are being merged. This command identifies and renames the scope dumped in each of the simulations to a new scope name to be reported on.

If the top-level module name and the instance name used to instantiate the reference and transistor models are the same in all simulations, you do not have to use this command. Do not try to map the top-level testbench name because a testbench is never the same when you use multiple testbenches to verify a design.

Usage: `map <dumpscope> <newscope>`

For example,

```
map seq_tb_top.ref mergedref
map dat_tb_top.ref mergedref
map seq_tb_top.xtor mergedxtor
map dat_tb_top.xtor mergedxtor
```

Another example,

```
map inno_tb_top.xtor xtor
map inno_tb_top.ref ref
map protocol.xtor xtor
map protocol.ref ref
map data.xtor xtor
map data.ref ref
```

If you use a relative instance path in the simulation, for example in the top-level module (`inno_tb_top`), you place the following statement:

```
$espcovscope(0,xtor);
```

The map statement requires the fully qualified instance path that is used as follows:

```
map inno_tb_top.xtor newxtor
```

Rename

This statement creates alias names for symbolic variables used within the simulation. If you have used different symbol names in the simulations for the same input pins, you need this statement to identify which fully qualified symbol names are equivalent.

Usage: `rename <oldname> <newname>`

For example,

```
rename inno_tb_top.A_sym1 A_sym1
rename protocol.addresssym1 A_sym1
rename data.adrsym1 A_sym1
```

Another example,

```
rename tb1.* *
rename tb2.* *
```

Example 2 shows a shorthand notation to alias all the symbols used in module `tb1` to just the symbol name. Module `tb2` also has all its symbol names aliased to just the symbol name. This assumes that the symbol names used are consistent and just the top-level module name is different. An easier way to accomplish the same thing would be to just use the same top-level module name in each of the testbenches.

5

Coverage Reporting

The `espcov` utility provides a summary report as the default. The summary report prints just the scope, module name, and the coverage percentage for each of the metrics reported. Summary information is also sent to `stdout` regardless of the amount of detailed reporting selected.

The coverage report is scope-based. It might be impractical to expect all scopes to be printed out, so you can select the scopes to be printed. For example, a coverage report for a typical hierarchical SRAM would list all the core cells if everything were printed.

The mechanism to select the scopes that are reported is the specification file, which lists the Verilog scopes to report. It needs to be specified from the top (using full paths). There can be multiple scopes in the specification file.

If no scope is specified, the top-level scopes where data was gathered in the testbench is reported. In the case of a testbench generated by `esptbgen`, these would be `ref` and `xtor`.

To obtain more information, use the optional `-spec` command-line option to choose the information to display, to request more information about other scopes, or to filter the coverage results.

Using either command-line options or specification file commands, you can turn on or off the detailed reporting for each of the individual sections in the report file. Each is controlled independently from the other.

Usage: `espcov [<options>] [-spec <file>] [<coverage_db_file> ]`

This command writes out a single report file (by default `espcov.log`) summarizing the coverage information for each of the coverage types.

You can specify the name of the coverage database file created by a previous simulation. If no name is supplied, the file is called `esp.cov`.

This chapter includes the following sections:

Command-Line Options

The command-line option available for `espcov` are

```
-h              Display help information
-printdetail "<listofcovtypes>"
                A command line option to enable more detailed
                information to be placed in the output file without
                using a spec file. To specify all options use
                'all', otherwise use one or more of DS, NS, LC, and
                SA in a space delimited string. Cannot be used when
                -spec option also present.
-spec <file>    Specify the name of an optional file which controls
                the level of detail of coverage information to be
                reported. This option cannot be specified if
                -printdetail used.
-filterdetail   Specify that detailed information for signals/nets
                that are filtered out by commands in the spec file
                are printed in the report output.
-unusedfilters  Print out detailed information about unused filters
                in the coverage report file.
-l <log>        Optionally specify a report/log file name. If not
                specified, the file name espcov.log is used.
```

Examples of command lines:

```
espcov -printdetail all
espcov -printdetail "NS DS" esp.cov
espcov -spec cov.spec -l espcov.log esp.cov
```

Report Specification File Statements

The following sections provide details about the file statements for controlling the information put in your summary report.

Comments

Comments start with the character `#` in column 1. For example:

```
#Comment line
```

Select

Selects a list of modules, other than top-level modules, to report on in the output file. The hierarchical name used in the select command must be the full hierarchical name from the root and not a relative hierarchical name. This is different than `$espcovscope` usage where relative hierarchical names are allowed. This is because the `select` command is not in any local scope and hence absence of a full hierarchical name from the root can lead to ambiguity unless you have mapped the root of your design hierarchy. For example, if you have a merged database with two new top-level modules, `xtor` and `ref` you can use

```
select (0,ref)
select (0,xtor)
```

Usage:`select <scope> [ <scope>+]`
`<scope> ::= '(' <level> ',' <instancepath> ')'`

Example 5-1 Selecting a Scope of Lower-level Modules

```
select (0,inno_tb_top.ref)
select (0,inno_tb_top.xtor.I1)
select (0,inno_tb_top.xtor.I2)
select (1,top.ref) (0,top.xtor)
```

Suppose you created a merged database by running `espcovmerge` with a specification file that contained the following `map` commands:

```
map inno_tb_top1.ref refmerged
map inno_tb_top2.ref refmerged
map inno_tb_top1.xtor xtormerged
map inno_tb_top2.xtor xtormerged
```

The `map` commands, which define mergable scopes `ref` and `xtor`, end up creating two new top levels in the merged database: `refmerged` and `xtormerged`. The `select` commands shown in Example 5-2 apply to the merged database.

Example 5-2 Selecting a Scope from a Merged Database

```
select (0,xtormerged.I1) (0,xtormerged.I2)
select (1,refmerged)
```

In this example, the `select` command applies to the merged database of `inno_tb_top1` and `inno_tb_top2`. Thus in this context, the full hierarchical paths for the `select` command start from `xtormerged` or `refmerged`.

printdetail

Causes detailed coverage information to be reported for all scopes. If you do not specify the `printdetail` option, the default is to report only summary information for all the coverage types. With this option, you also indicate which coverage type (or types) should have detailed information listed. You can use `printdetail all` to specify all the keywords available in the specification file. Otherwise, use SA, DS, NS, or LC specific selections.

Usage: `printdetail <covtypelist>`

For example,

```
printdetail SA DS NS
```

Another example,

```
printdetail NS
printdetail DS
```

filter and filtertree Commands

A filter removes information from the coverage report that is not needed. For example, when analyzing the results of real-world test cases, some signals are always nonsymbolic, such as clocks. In other cases, you can apply symbols to inputs just to ensure that they are not used (for example, data input symbols on a read operation). These symbols show up in the output report as dropped symbols. To mask these symbols out of the report, two filter operations are provided: `filter` and `filtertree`.

The `filter` command takes a list of signal names, which can include wild cards, and a coverage metric type (NS, DS, SA, or LC). It prevents the selected signals from being reported for that coverage type. Since the LC coverage type uses a completely different format, it is covered separately in the section “Filtering for LC Coverage Type” on page 5-8. Notice that symbols are implemented as signals in the testbench.

The `filtertree` command, which is applicable only for SA and NS coverage metrics, helps in filtering all buffer- and inverter-connected trees associated with a signal. This convenience can sometimes drastically reduce the number of `filter` commands or the length of the list of a `filter` command in the spec file.

Usage:

```
filter <covtype> <fully_scoped_identifier> [<missingtoggle>]

filtertree <covtype> <fully_scoped_identifier> [<missingtoggle>]
```

where

<covtype> is NS, DS, SA, or a combination of these, separated by spaces.

<*fully_scoped_identifier*> ::=
<*instancepath*>'.' <*net_name*> [<*bit_enum*>]

<*bit_enum*> ::= '[' <*enum_range*> [',' <*enum_range*>] ']'

<*enum_range*> ::= <*num*> | <*num*> '-' <*num*>

<missingtoggle> ::= norise | nofall, and only applies to stuck-at (SA) coverage.

If you do not specify a value for <missingtoggle>, both missing rise and missing fall transitions are ignored. If norise is specified, missing rise transitions are ignored in the report. Similarly, nofall ignores missing fall transitions.

Defining a Fully-Scoped Identifier

A *fully_scoped_identifier* has the form <*instancepath*>'.' <*net_name*> [<*bit_enum*>].

Rules for defining the *instancepath* are explained in the next section. The *net_name* supports bit number enumeration the same as those used in the coverage report (for example, net[20-5,3-1]).

Note:

A *fully_scoped_identifier* can be created by a merge remap operation. If you run espcovmerge with a merge spec file that contains one or more map commands, the output is a single merged file. This <*merge_renamed_scope_identifier*> can be used as a <*fully_scoped_identifier*> in a filter or filtertree command.

Defining an Instance Path

The *instancepath* must begin with a module name, which can be the top module name, but is not necessary (inno_tb_top in the case of an esptbgen generated testbench). This usage of instancepath is more restrictive than the one allowed in $espcovscope.

Each element in the *instancepath* represents a level of hierarchy. It is separated from the other elements by the "." character. Each element can also contain wildcard characters. See the next section for more information.

The *instancepath* can never be empty in the context of a filter command. If the net name does not exist in the coverage database, an error message is displayed.

Wildcard Characters in Filter Instance Paths

This feature is exclusively available for `filter` and `filtertree` commands to allow for concise representation for filtering a large number of signals. The name of a filtered signal and all the identifiers in the instance path including the starting module name can use wildcard characters.

The SA, NS, and DS filters have the form `hier.net` where part1 = `hier` is the hierarchical path to the net and part2 = `net` is the name of the net being filtered. This hierarchical path is restricted to be a `module_name` followed by zero or more `instance_name`s.

You can apply the following wildcards to part1 as well as part2:

```
*               Match any string of zero or more characters.
?               Match any single character.
(ab|cd|ef)      Match the alternative selections between
parentheses.
```

The following wildcard can be applied only to part1 of the filter:

```
[xyA-Z0-9]      Match any one of the enclosed characters; a
hyphen can be used to specify a range (e.g., a-z, A-Z, 0-9)
[^xyz]          Match any character not among the enclosed
characters.
```

The following wildcard can be applied only to part2 of the filter:

```
23-25,12-9      Match any number between the specified range.
In this example 23, 24, 25, 12, 11, 10 and 9 are matches.
```

Example 5-3 Filtering All Bits of a Net

```
filter DS inno_tb_top.net[*]
```

In this example, the module name is `inno_tb_top`, the `net_name` is `net` and entire bus bits are specified with `'[*]'`.

Example 5-4 Filtering Specific Bits in a Group of Modules

```
filter DS *ABC*.rst.X*B.hn*[30-8,5-0]
```

In this example, the module name is all modules that have the substring `ABC` in their names. The instance path starts with the instance `rst` under each of the selected modules. The `rst` instance must have a child whose name begins with `X` and ends with `B` (`X*B`). The net name is all the bus signals that begin with `hn` and have the specified bits.

Example 5-5 More Wildcard Examples

```
filter NS A(pq|rs)*.*bc*.a??[a-zA-Z]*b?[^0-9]*.net[15-11,9-2]
filter SA module.net[*]
filtertree NS module.net[*]
```

Follow these rules when you use wildcards:

- The * matches 0 or more characters.
- `module_name` cannot be * (an exhaustive search is too expensive). For example, `filter NS *.clk` is not allowed, but `filter NS M*.clk` is allowed).
- If the first character of *`instancepath`* is a backslash (`\`), the entire filter does not pass to the regular expression engine for evaluation. You can protect any wildcard by using the backslash (\ ) as a regular character.
- The top-level instance name (for example, `inno_tb_top`) is also treated as a module name for filters and wild cards.
- Wildcard characters cannot include the hierarchical separator ".". They cannot cross hierarchical boundaries. You must explicitly specify each of the levels of hierarchy in the path. For example, `a.b.net` can be represented by `a.*.n*` but not by `a.*`.

Filtering for LC Coverage Type

Line coverage filtering identifies and removes source code that you know does not require coverage reporting. The LC filter has the form *`file`*`:`*`line`*`|`*`range_of_lines`* where a range is separated by a hyphen.

Usage: `filter LC <`*`filename`*`>:<`*`range_of_lines`*`>`
where
`filename` is the source file you want to report on. *`range_of_lines`* gives the lines of source code that are filtered from the report.

Example 5-6 Filtering Lines of Source Code

```
filter LC abc.gv:5
filter LC ../ab*.v:5-8
filter LC abc.gv:8-10
```

Filtering Examples

This section illustrates many of the forms used in `filter` and `filtertree` commands.

Example 5-7 Filter Statements for Various Coverage Types

```
filter NS inno_tb_top.ref.clk
filter NS inno_tb_top.ref.MX0.clk
filter SA inno_tb_top.ref.MX0.clk norise
filter SA inno_tb_top.ref.MX1.clk nofall
filter SA inno_tb_top.ref.MX2.clk
filter DS inno_tb_top.a_sym1[0-2]
```

Example 5-8 Using Filtering on a Merged Database

```
map inno_tb_top1.ref refmerged
map inno_tb_top2.ref refmerged
```

```
map inno_tb_top1.xtor xtormerged
map inno_tb_top2.xtor xtormerged

filter NS refmerged.clk xtormerged.I1.net1
```

In this example, the `filter` command applies to the merged database of `inno_tb_top1` and `inno_tb_top2`. The `map` commands that define mergable scopes `ref` and `xtor` end up creating two new top levels in the merged database: `refmerged` and `xtormerged`. Thus, in this context, the full hierarchical paths for the `filter` command start from `xtormerged` or `refmerged`.

Example 5-9 Clock Filtering

```
1a)        filter NS inner_module_name.clk
1b)        filter SA inner_module_name.stuck_net_name
```

In this example, the instance path starts with a module name that is not the top-level module name. This helps in identifying `clk` as a nonsymbolic signal for all instances of `inner_module_name`. You have to make sure that this is always a clock signal in all instances where the module is used. Otherwise, you could filter out signals that you should examine. Alternatively, you could enumerate all instantiations of `inner_module_name` as follows:

```
2a)        filter NS inno_tb_top.xtor.I1.inst1.clk
2b)        filter NS inno_tb_top.xtor.I2.inst1.clk
```

This could be a laborious task, and lead to a long list. Yet another way is to use wildcards, but this is expensive in terms of the runtime of the coverage tool.

```
3)         filter NS inno_tb_top.xtor.I*.inst*.clk
```

This assumes that all the module instantiations are covered by the preceding regular expression.

Finally, another way to filter the specified `clk` signal is by using the `filtertree` approach:

```
4)         filtertree NS inno_tb_top.xtor.clk
```

This works only if the `clk` signal of all the instantiations of `inner_module_name` are connected through a buffer or inverter tree to `inno_tb_top.xtor.clk`.

In terms of efficiency and ease of use, (1a,1b) are the best, followed by (4), (3) and (2).

Filtering Internal Non-Driving Nets

Using the `espcv -cov_filter_intnodes` command ESP-CV filters out internal nets in a design that are not driving any gates (off by default), and places them in the omitnet.spec file. These nets are not reported in the nonsymbolic (NS) and toggle coverage (SA) section of the espcov.log file. Run coverage using `+define+ESPCOV` for this command to take effect. For example, you would run

```
espcv <design files> +define+ESPCOV -cov_filter_intnodes
```

ESP-CV writes out all filtered nets to the omitnet.spec file with a filter command for both NS and SA. For example, an omitnet.spec file enter is

```
filter SA <internal node>
filter NS <internal node>
```

The detection mechanism for filtering these internal nets detects and filters nets that are not driving a transistor or Verilog gate. For example, internal nodes of channel connected regions (CCR), flip-flop inputs and outputs, and bit lines in core memory cells are typical topologies in a memory design that get filtered out.

Detecting and Levelizing bufinv Trees

ESP Cov automatically detects bufinv trees and prints them in a levelized manner for nonsymbolic (NS) and toggle (SA) coverage. Use the `-printdetail <NS|SA>` option to the `espcov` command to output this information to the espcov.log file. You can turn this default operation off using the `-nobuffertree` option to the `espcov` command.

In the nonsymbolic coverage report section, those regular nodes not getting any symbols during simulation are printed first, followed by bufinv nodes. In the toggle coverage section of the report, the regular nodes that did not have a rise or fall transition or both, are printed first followed by the bufinv nodes. The source node of the bufinv tree is printed first, marked with >>. Rise, fall, and type information is provided to enable you to identify why the bufinv tree did not toggle. The driven nodes (net name only) follows the source node and are indented.

ESP Cov creates a spec file for each of the bufinv nodes at the end of the ESP Cov run listing all the bufinv nodes with a 'filtertree' command for the source node and the 'filter' command for all driven nodes. The filtertree lines by default are commented out with the '#' sign and the filter commands are left active (uncommented). You can pass this file in a subsequent run of `espcov -spec` and filter out all the driven nodes except the master or the source node to reduce debugging the number of nodes. If you don’t care about the bufinv nodes, you can edit the bufinv.spec file to uncomment the filtertree lines and comment out or delete the filter command lines.

The bufinv.spec file contains the filtertree and filter lines for each of the coverage metric, NS, and (SA) coverage depending on how you used the `-printdetail` option in the initial run. If you run `espcov -printdetail 'SA'`, the bufinv.spec file contains filter tree and filter commands with the 'SA' specification. If you specify both NS and SA using the `-printdetail` option, the filter tree and filter commands are listed for both metrics.

With each execution of `espcov`, the tool creates the bufinv.spec file based on how the filtertree and filter commands are applied to those nodes. When running ESP Cov, if all the bufinv nodes are filtered by passing in the bufinv.spec file with the filtertree lines uncommented out, the bufinv.spec at the end of this run is empty.

In the examples in this section, the following file, top.v is used. This file shows Verilog definitions.

```
module myand(z,a,b) ;
input a,b;
output z;
and a10(z,a,b);
endmodule

module myinv(o,i,j);
input i;
input j;
output o;
wire w1,w2,w3;
wire za0, ma1;
supply1 innovcc;
supply0 innovss;

pmos MP1(o,innovcc,i);
nmos MN1(o,innovss,i);
not n11(w2,w1);
not n21(w3,w2);
and an1(za0,j,ma1);
endmodule

module top;
reg ss;
wire a1,a2,a3;
wire b1,b2,b3;
wire c1;
wire a0;
wire d1,d2,d3;
wire f1,f2,f3;
wire e1,e2;
not n1(a0,a1);
not n2(a3,a0);
myinv i3(b2,b1,f1);
myinv i4(b3,b2,f2);
myinv i5(e2,e1,f3);
myand i9(d3,d2,e2);
```

```
initial begin
$esp_var(ss);
$espcovfile("top.cov");
$espcovscope(0, top);
end
endmodule
```

After running the commands,

```
espcv top.v -hc high
espcv top.cov -printdetail SA
```

the espcov.log file includes the following information. Notice all the bufinv nodes start with >> so you can identify them.

```
espcov <header info>
...

***********************************
and2_a (and2_a)
Symbolic Net Coverage: 0%
 All nets nonsymbolic
Propagated Symbol Coverage: not applicable
 Propagated Symbols: 0 propagated / 0 symbols
 Dropped Symbols: 0 dropped / 0 symbols
Toggle Coverage:     0%
 Untoggled Nets: 31 / Total Nets 31
        Missing toggles rise    fall    type    name
        X       X       B       a2
        X       X       B       c1
        X       X       B       d1
        X       X       B       d2
        X       X       B       d3
        X       X       B       f1
        X       X       B       f2
        X       X       B       f3
        X       X       B       i3.ma1
        X       X       B       i3.za0
        X       X       B       i4.ma1
        X       X       B       i4.za0
        X       X       B       i5.ma1
        X       X       B       i5.za0
>>      X       X       B       a1
                                        a0
                                        a3
>>      X       X       B       b1
                                        b2
                                        b3
>>      X       X       B       e1
                                        e2
>>      X       X       B       i3.w1
                                        i3.w2
```

```
                                                  i3.w3
>>      X       X       B       i4.w1
                                                  i4.w2
                                                  i4.w3
>>      X       X       B       i5.w1
                                                  i5.w2
                                                  i5.w3
```

The bufinv.spec file that ESP CV outputs after the espcov.log file is as follows. Notice, the master and source nodes have the filtertree command and the driven nodes are listed with the filter command

```
#****************************************
#filter information for bufinv structures  (and2_a)

printdetail SA
#filtertree SA and2_a.a1
filter SA and2_a.a0
filter SA and2_a.a3
#filtertree SA and2_a.b1
filter SA and2_a.b2
filter SA and2_a.b3
#filtertree SA and2_a.e1
filter SA and2_a.e2
#filtertree SA and2_a.i3.w1
filter SA and2_a.i3.w2
filter SA and2_a.i3.w3
#filtertree SA and2_a.i4.w1
filter SA and2_a.i4.w2
filter SA and2_a.i4.w3
#filtertree SA and2_a.i5.w1
filter SA and2_a.i5.w2
filter SA and2_a.i5.w3
```

This feature does have a few limitations. The first being ESP Cov does not include true source node information for a bufinv tree during toggle coverage reporting in HC mode. In one manifestation, if the bufinv tree's source node is supply1 or supply0, ESP Cov treats the source node as a constant signal and there is no rise/fall/type information for that node. ESP Cov reports the first driven node as the master node, indicated with >> and noted with an NM indicating it is not a true master (source) node. The second manifestation arises when you run ESP Cov with the `select` command set to a lower-level instance in the design hierarchy than the top module instance. If the source node of the bufinv tree inside this lower-level module is determined to be at the top-most level, the rise/fall/type information for this source node is unavailable, and ESP Cov makes the first driven node the master (source) node. ESP Cov marks these nodes in the toggle section with NM. You can remedy this limitation by removing the `select` command from the spec file.

Another limitation deals with non bufinv tree nodes appearing in the bufinv tree levelization section in HC mode. This is caused by the determining master (source) node of nets under HC mode. For a driven node in this instance, the master or source node is the starting node

of the tree and the driven node is different from the source node. For non bufinv tree nodes, the driven node and the master or source nodes are identical. Although distinctions are made, situations can arise where the master node for a particular non bufinv tree node is derived to be different than the actual node. For example, this could occur if the input to the non bufinv type gate is also an input to a bufinv tree section. Then this input is passed from a higher level in the design hierarchy and you placed a `select` command on an instance lower in the hierarchy where the non bufinv type gate and the bufinv tree are present. In this situation, you can eliminate the problem by removing the `select` command.

Interpreting the Report Format

The significant sections of the coverage report are the propagated symbol coverage followed by the symbolic net coverage sections. You want these numbers to be as high as possible. In particular, if the propagated symbol coverage is not 100%, all dropped symbols must be logically reasoned and subsequently filtered with the help of the spec file (for example, data in symbols of a read cycle).You might also need to use filtering to remove known nonsymbolic signals (such as clock signals and their fanouts). `filtertree` can help filter clock trees and other uninteresting buffer or inverter connected signal trees in a single spec file command.

The format of the coverage report output is shown below. There might be one or more `<scope>` sections depending on the number of `$espcovscope` statements in your Verilog code or the number of scopes selected in your `espcov` spec input file.

If you have not specified `-printdetail`, only the summary information appears in the report. The lists under each summary header is not printed. Adding `-printdetail all` adds the detailed information for each coverage type.

The meaning of fields in the report are as follows:

```
 <scope>              Full hierarchical path to the scope in
question.
 <modulename>         Name of module
 <TotalNets>          Total number of nets as seen by the
simulator
```

Figure 5-1 illustrates the format of the coverage report file.

Figure 5-1 Coverage Report File Format

```
Welcome, ESP COV Version <version>, <compiledate>
Copyright (C) 1998-2002 by Synopsys, Inc. All rights reserved.

command line arguments:
 <commandlineused>

  ESP Coverage Database Version 2.0 read on <currentdate>

**********************************
<scope> (<modulename>)
Symbolic Net Coverage: XXX%
 Nonsymbolic Nets: <NumberOfNets> / Total Nets <TotalNets>
    <list of nets which have never had symbols applied>
 <number> Nonsymbolic Nets Ignored by Filter
    <list of nets which have been removed by the filter>

Propagated Symbol Coverage: XXX%
 Propagated Symbols: <NumberOfSymbols> propagated / <TotalSymbolsUsed>
    <list of symbols which entered and exited this scope>
 Dropped Symbols: <NumberOfSymbols> dropped / <TotalSymbolsUsed>
    <list of symbols which entered but did not exit this scope>
 <number> Dropped Symbols Ignored by Filter
    <list of nets which have been removed by the filter>
 <NumberOfNets> Binary nets with symbolic fan in
    <list of nets which always have binary values but have symbolic inputs>

Toggle Coverage: XXX%
 Untoggled Nets: <NumberOfNets> / Total Nets <TotalNets>
    <list of nets which never go to 1 or 0>
 <number> Untoggled Nets Ignored by Filter
    <list of nets which have been removed by the filter>

**********************************
Line Coverage: XXX%
 Total lines: <TotalLines>, Uncovered lines: <NumberOfUncoveredLines>

    <file name> has the following uncovered lines
  <line> <list of instances which do not have this line covered>
 <number> Uncovered Lines Ignored by Filter
    <list of nets which have been removed by the filter>

Unused Filters in <specfile>: <numberunused>/<totalfilters>
    <list of filters which were never activated>
```

An example of a summary output for the coverage generated by the standard testbench is as follows:

Figure 5-2 Summary Output of Coverage Report

```
Welcome, ESP COV Version 5.0.0, Mon Sep 2 21:38:37 PDT 2002
Copyright (C) 1998-2001 by Synopsys, Inc. All rights reserved.

command line arguments:
 espcov esp.cov
ESP Coverage Database Version 1.1 read on January 23, 2002 10:22:06

***********************************
inno_tb_top.ref (Reference)
Symbolic Net Coverage: 90.32%
 Nonsymbolic Nets: 3 / Total Nets 31
Propagated Symbol Coverage: 90.47%
 Propagated Symbols: 57 propagated / 63 symbols
 Dropped Symbols: 6 dropped / 63 symbols
 0 Binary nets with symbolic fan in
Toggle Coverage: 100%
 Untoggled Nets: 0 / Total Nets 31

***********************************
inno_tb_top.xtor (implement)
Symbolic Net Coverage: 90.55%
 Nonsymbolic Nets: 12 / Total Nets 127
Propagated Symbol Coverage: 90.47%
 Propagated Symbols: 57 propagated / 63 symbols
 Dropped Symbols: 6 dropped / 63 symbols
 0 Binary nets with symbolic fan in
Toggle Coverage: 100%
 Untoggled Nets: 0 / Total Nets 127

***********************************
Line Coverage: 100%
Total lines: 1479, Uncovered lines: 0
```

No filters were applied in this case, so the filter information is not applicable. This explains the difference from the template in Figure 5-2. For information about how the detailed output looks for each section or how to interpret the data, see the appropriate sections that follow.

Symbolic Net Coverage Results

This section reports the nets in every module that do not have symbolic values assigned to them at any time during the simulation. To add details for this section, specify `printdetail NS` in the spec file, or add the command-line option `-printdetail 'NS'`.

The default is to dump information for all modules below the specified scope. You can limit the information dumped per scope by specifying how many levels below that you want dumped using the `select` spec file command.

The following example shows a report of symbolic net coverage.

```
**********************************
test.a.x (blk5)
Symbolic Net Coverage: 85%
 Nonsymbolic Nets: 6 / 40
                bResult
                clk
                I1.cb
                I2.dat[4,2-0]
```

This report shows that under the instance `test.a.x`, where `blk5` is the module name, `bResult` and `clk` are always binary. There is also an instance `I1` within `blk5` that has the signal `cb` that is always binary. In instance `I2`, only bits 4 and bits 2 to 0 of net `dat` are nonsymbolic. All other bits of the bus are symbolic. The format of the bit range is little endian, not necessarily the original bus order.

Note:

If no `$esp_var` exists in the testbench, ESP runs as if `NS` and `DS` are not present in `$espcovfile`. The sections *Symbolic Net Coverage,* and *Propagated and Dropped Symbols* are not shown in the report.

Similarly, if `$esp_var` has no arguments (so it looks like `$esp_var()`), it does not report these sections either.

If `$esp_var` has arguments but the arguments are not used, it generates a report for symbolic net coverage and propagated symbol coverage. In this case, the symbolic net coverage is expected to be 0 percent, and propagated symbol coverage 100 percent(0 propagated out of 0 symbols).

Propagated Symbol and Dropped Symbol Results

This section reports symbols in modules that are propagated through the module and reports symbols that come in, but do not go out of the module as dropped symbols.

Add the command-line option `-printdetail 'DS'` or the spec file command `printdetail DS` to enable the detailed output for this section.

An example of a detailed report file for DS is shown here.

```
************************************
inno_tb_top.xtor (rand)
    :
    :
Propagated Symbol Coverage: 80%
 Propagated Symbols: 4 propagated / 5 symbols in
    inno_tb_top.a_sym2
    inno_tb_top.a_sym3
    inno_tb_top.b_sym2
    inno_tb_top.b_sym3
 Dropped Symbols: 1 dropped / 5 symbols in
    inno_tb_top.b_sym1
 0 Binary nets with symbolic fan in
```

In this example, the symbols `a_sym2`, `a_sym3`, `b_sym2` and `b_sym3` were applied to the inputs of the module and were propagated by the simulation to the outputs of the module.

The symbol `b_sym1` was applied as an input but it did not reach an output pin at any point in the simulation.

Propagated Symbol Coverage information is recorded for all the sub-instances subject to the `$espcovscope` directive. For example, if you specify `$espcovscope(0,xtor)`, all the sub-instances under module `xtor`, propagated and dropped symbol information is recorded. The reporting at any specific level includes all the nets in that scope as well as all the nets of all its children and their children down to the lowest leaf cells. Notice that `$espcovscope` can limit the number of generations of children that the data is gathered for, which can exclude the leaf cells.

Setting the Scope for Dropped Symbol Reporting

Collecting dropped symbol data during a symbolic simulation is costly in runtime. As you gather data for lower and lower levels of modules, this operation becomes more expensive. Furthermore, if you run with `-hc`, it becomes so expensive that it can cause the symbolic simulation run to randomize or run out of memory. However, usually you do not need to collect the DS data for lower levels. By default, DS coverage is done only for the top-most level. More detailed data is required only for debugging and locating the logic that dropped a symbol. If the propagated symbol coverage is 100 percent or the dropped symbols are easily explainable (which is usually the case), the lower level data would be redundant.

The levels of DS data collection can be controlled using the `espcv` command-line option: `-covdslevel n`. You should leave this switch at the default level (n=1) at the `espcv` command line. If you need information for lower levels, change the level to 0 to dump all levels, or choose a level number that dumps the area you are interested in. For information about how to handle capacity issues under coverage, see “Frequently Asked Questions (FAQs)” on page 5-26.

If you want to find out exactly where a symbol was dropped, look at the dropped symbol information for the sub-instances of the top-level scope that drops the symbol. Locate the sub-instance that drops this symbol. You can repeat this process successively to zero in on the logic that actually dropped the symbol.

The dropped symbol information loses its accuracy in the following cases:

- Some or all the ports of a module are `inout`

  In this scenario, ESP treats an `inout` port as an input as well as an output port. Thus any symbol that “comes in” on such a port is also assumed to be “going out”. For such modules, the dropped symbol information is optimistic and incorrect (a symbol might be assumed as propagated where it was actually dropped).

- A port of a module is multi-driven and at least one of the drivers does not belong to this module

  This typically happens at fine granularity where an instantiation is a transistor and the output of the transistor is wire-anded or wire-ored with outputs of other transistors. For such scenarios, a symbol that is not at the input port of a transistor might appear at the output port through another transistor. Even if one transistor drops a symbol, another one might propagate it.

You should set the scope of your investigation of dropped symbols down only if the ports of the lower-level module do not violate the preceding criteria. ESPS2V determines the direction of an `inout` port (whether it is input or output) on a best effort basis.

Binary Nets With Symbolic Fanin (Without -hc)

A binary net is where a net can be toggling between binary values 0, 1, X, Z but never gets a symbolic equation value. For example, if you have a 2-input AND that has a data input with the binary value 0 but the other data input is symbolic, the output is reported in this subsection because a symbolic value was not propagated on the output of the AND. This section does not show up when using hierarchical compression because the information cannot be determined in that environment. This is usable for ESP-LV where hierarchical compression is not used.

Stuck At (Toggle) Coverage Results

This reports the nets in every module that do not have complete toggle coverage. It reports, per module, the nets within that module that do not have a rise or a fall transition. From that you can infer that the net is stuck at a particular value because the transition to that value has not occurred.

A rise transition is any transition to the value 1. This includes the transitions from 0, X, or Z to 1. A fall transition is any transition to the value 0. This differs from the Verilog definition of `posedge` and `negedge`.

Nets are also marked as being S (symbolic) or B (binary). The definition of binary is obvious in that the net has never had a symbolic value associated with it. The net has a value of 0, 1, X or Z.

The definition of symbolic is more complex. If a net has been marked as symbolic, an X in the `fall` column means that a transition to 0 never occurred and the net has a symbolic value on it at some point. For example, assume a net has an initial value of 0 at time 0 and has a formula (F) that contains only 0, X or Z at time t. If the value changes to ~F at time t+1, only a rise transition could have occurred.

The initial value of a net is determined as follows: at time 0, simulation starts with all nets as X. A number of initial values can be asserted at time 0 that cause some nets to get assigned a value other than X before the next time step is started. At the point where the next time step after time 0 is begun, coverage samples all the net values and uses that value as the initial value of the net.

In another example of a symbolic net, assume the net has an equation, F, at time t that can be 0 or 1 or X or Z. If the formula changes to ~F at t+1, both a rise and fall transition have occurred at the same time. If a signal has both a rise and fall, it is not reported as having a missing toggle.

Add the command-line option `-printdetail 'SA'` or the spec file command `printdetail SA` to enable the detailed output for this section.

An example of the report is shown in Figure 5-3. The top level being reported on is `test`. The hierarchical path to the module named `mod1` in this example is `test.a.b` and the path to the module `gate2` is `test.c`.

Figure 5-3 Sample Coverage Report

```
************************************
test (test)
                 :
                 :
Toggle Coverage: 50%
 Untoggled Nets: 7 / Total Nets 14
                Missing toggles
    rise fall type  name
          X    S    a.b.Co
     X    X    B    a.b.b
     X         B    a.b.y
          X    S    c.net103
          X    S    c.net106
     X         S    c.net136
     X         S    c.net89
```

This report shows that there is a signal named `a.b.Co`, which is a symbolic signal and it is stuck at non-zero. The X under the `fall` column indicates that a fall transition has not occurred or is missing. From that, you can infer that it has transitioned to a nonzero value since it never had a fall transition.

The signal named `a.b.b` never had a rise or fall on it. The initial X or Z value remains throughout the simulation.

Another signal named `a.b.y` is a binary signal and it is stuck at non-one. It had a fall transition from X to 0 but never had a rise transition.

If a net has been marked as symbolic (S), an X in the `fall` column still means that a transition to 0 never occurred. In the case of module `gate2`, the signal `c.net103` could have changed from X -> 1 -> X -> 1 -> Z -> 1 when evaluating one possible solution to the equation, but it did not get assigned the 0 value.

Similarly, `c.net89` had a fall transition of X -> 0 -> X -> 0 but never had a transition to the value 1. A symbolic `negedge` is defined as a transition from X or Z to 0.

Toggle Coverage for Primitive Cells

If you have a cell with internal nodes that you do not want coverage to report on, use the `‘innocov_off ‘... endinnocov_off` construct to ignore those nodes. This also affects line coverage. Any uncovered lines within the primitive cell is reported as the primitive instance line being uncovered.

```
‘ifdef _ESP_
      ‘innocov_off
      ‘endif
module inv (y, a);
      input a;
      output y;
      m1 y a vcc vcc p
      m1 y a vss vss n
endmodule
‘ifdef _ESP_
‘endinnocov_off
‘end if
```

If you need to run simulation using these cells in a simulator other than in a Synopsys product, you need to qualify these commands using the `_ESP_ compile` definition.

Line Coverage Results

This section reports the lines that have never been simulated. For behavioral code, you can determine which branches have not been tested. For switch-level models, you can find out which transistors have gates that are always off during the part of simulation that is gathering coverage data.

This section is only scope-based with regard to gathering the data. The output has no scope and all uncovered lines are identified. Only one line coverage section is shown in the output of `espcov` even if more than one scope has been specified with `$espcovscope`.

Line coverage is reported as the percentage of lines covered per total lines present. It does not matter how many times a module has been instantiated.

The line coverage report identifies which specific instances have not had a line covered, rather than which instances have that line covered. Unused modules are not reported as uncovered.

If different paths are used for the same file names and you are trying to merge the data, the full path name is used to match the coverage information to be recorded.

Example Coverage run 1 references file `f.v` using `a/b/c/d/f.v`. Coverage run 2 references the same file using `a/lnk/d/f.v`. The line coverage report shows them as two separate files after the merge.

If two identical designs are represented by two different unidentical files, the information is merged, with both files being considered as separate files and reported separately.

If you are doing divide-and-conquer simulations in different directories, the line coverage might not be accurate unless the file name and path are identical. (Relative file paths are recognized when used.)

Add the command-line option `-printdetail 'LC'` or the spec file command `printdetail LC` to enable the detailed output for this section.

The coverage output consists of one or more sections based on the files being compiled. The first part of each section indicates the file name and path and is followed by one or more line numbers that have never been encountered during simulation. The line number is followed by one or more hierarchical instances that reference that line and have not been executed.

```
     <file name> has the following uncovered lines
 <line> <list of instances which do not have this line
covered>
```

Line coverage is reported by blocks of sequential RTL code. Lines within the block are not reported as uncovered if the first line is reported as uncovered. Since all lines have to be executed sequentially, you can assume that the following lines in that block also have not been executed.

Example section of code:

```
initial begin // line 200 in test.q (module is ex1)
      X = 0;
      #10 $finish();
end
initial begin
      A = 1;
      B = 2;
      #100
      C = 0;
      D = 1;
end
```

Example of a report file:

```
Line Coverage: 99.7%
 Total lines: 1000, Uncovered lines: 3

          tb_esp.v
   35    test.a.b
  208    test.q.c
```

In this example, the file `tb_esp.v` line 35 has never been encountered during simulation. The hierarchical instance path, which includes this line, is reported as `test.a.b`. Also, lines 208 and 209 have never been executed and these are reported via the report for `tb_esp.v:208` line. Since C and D are executed together and cannot be interrupted, only the first line of that code block is reported. However, the number of uncovered lines does count all of the uncovered lines. In this example, three lines are reported as uncovered.

Example of Filtered Output File Format

Filtering the symbolic net coverage results increases the coverage metric so that you can identify what is not covered. This reduces the amount of output generated, revealing unexpected uncovered nets.

Filtering of results sometimes requires feedback on the effectiveness of the filter. You might want to report on things filtered out and filter statements not used. Reporting on these is not the default because the idea is to reduce the amount of data that is being reported. These

sections also show up only if a filter for a section has been specified in the spec file. Reporting of the filter details is controlled by the command-line option `-filterdetail`. If that option is not specified, only the summary result appears, not the detailed results.

The symbols, nets, and lines that get filtered out are the same format as if they were not removed by the filter. They are just placed in another section of the report. The total number of Nets/Symbols remains the same with or without filtering. The number of (nonsymbolic/ dropped symbol/untoggled net/ uncovered lines) is reduced by the filtered number. If you add those two numbers, you should get the unfiltered value.

If filter specifications have not been used to filter the coverage report data, these are reported in the unused filter section of the report. This section is printed once at the end of the report and only if the command-line option `-unusedfilters` has been specified to report it.

Example of output for unused filters:

```
Unused Filters in specfile: 4 / 20
    filter NS inno_tb_top.ref.clk
    filter SA inno_tb_top.ref.MX0.clk norise
    filter DS inno_tb_top.a_sym1[0-2]
    filter LC abc.gv:5
```

Example

```
module inno_tb_top ();
  `ifdef ESPCOV
  initial begin
      $espcovfile("esp.cov");
      $espcovscope(0,xtor);
      $espcovscope(0,ref);
  end
  `endif
endmodule
```

Limitations

The following sections describe the limitations of the coverage report.

Randomization of Symbols

Randomization is the conversion of randomly selected input symbols into constant binary values to work around capacity problems. If we use `-r` to impose binary values for randomized symbols before starting the coverage run, these nets are reported as nonsymbolic. The binary value on these signals can cause other symbols to be dropped because they are now disabled due to binary value used.

Signals in the `esp.Random` file that are read by the `-r` command-line option are not reported as dropped or propagated.

Output Checking

When analyzing coverage, consider whether the output signals are checked at the appropriate time. You can have symbols propagated throughout the design and your coverage numbers look good, but if you are not checking the outputs at the appropriate time, or not checking them at all, you do not detect any mismatches that are present. Therefore, coverage does not help you determine if your testbench is complete in that respect. This issue is not addressed by the current coverage implementation.

Toggle/Stuck-at Reporting Issues

There are nets in some circuits that can never achieve both a 0 and a 1 value and therefore show up in the SA report.

For example, suppose you have a NAND gate with the following SPICE code:

```
MN1 Y B I1 VSS N
MN2 I1 A VSS VSS N
```

The net `I1` in the bottom leg of the NAND gate never gets a 1 value under certain circumstances. The Verilog translated by `esps2v` with `-bidir auto` would be:

```
tranif1 MN1 (Y, I1, B);
nmos MN2 (I1, VSS, A);
```

If the `tranif1` is replaced with `nmos`, net `N1` can never have a 1 value. This might happen if you turn off bidirectional translation or do the translation yourself.

```
nmos MN1 (Y, I1, B);
nmos MN2 (I1, VSS, A);
```

The only value that net `I1` can have in this case is 0 or Z.

If the `tranif1` is used, the bidirectional translation of `tranif1` causes this net to go to a value of 1 if input A was off and input B was on. This might happen due to the way signals A and B interact.

Frequently Asked Questions (FAQs)

Q) Some nets of my design were not reported in the coverage results. Why?

By default, ESP has a gate extraction optimization. When the tool is run, some nets are optimized away. This improves its capacity.

Q) What can I do to force coverage results for all my nets (in other words, no nets should be optimized)?

To generate coverage for an exact netlist with no net optimization under coverage, use the switch `-noopt`, which shuts off gate extraction. Adding `-noopt` results in a decrease in performance and capacity. You should not use it in a simulation run with coverage turned on, if you did not use it in a simulation run without coverage.

Q) When I run `espcv` with coverage enabled, it randomizes or goes out of memory. The normal run without coverage does not. What can I do?

It is only logical to expect that the coverage enabled run affects the capacity of ESP negatively. Extra bookkeeping and computation might result in out of memory or randomization. The following workarounds are suggested for the coverage run to get around this problem:

- Always use `-randomize sysmem` in the coverage run.
- Try not to use the `-hc` option. Coverage result computation is expensive when the `-hc` option is used. If you need to use `-hc`, try the lowest `-hc` option (`-hc low`).
- Disable the dumping of symbol (DS) information collection with the `$espcovfile` command. (See “$espcovfile” on page 3-2 for a description of this command.)

```
$espcovfile( “<espcovfile>”, “NS   SA  LC”)
```

A

Coverage Syntax Summary

This appendix provides a summary for the commands you use when generating a coverage report. It includes the following sections:

Verilog PLI Statements in the Testbench

```
$espcovfile( "<espcovfile>" [, "<listofcovtypes>" ] ) ;
                <listofcovtypes> ::= <covtype>+
                <covtype ::= 'NS' | 'DS' | 'SA' | 'LC'
                NS - Non-Symbolic
                DS - Dropped Symbols and propagated symbols
                SA - Stuck At binary values and toggle coverage

                LC - Line Coverage

$espcovscope [ ( [<level> , <instancepath>] ) ] ;
                <instancepath> ::= [ <modulename> '.' ]
                                   <instanceid> [ '.' <instanceid>
]

$esp_exclcovscope [ ( [<level> , <instancepath>] )] ;
                <instancepath> ::= [ <modulename> '.' ]
                                   <instanceid> [ '.' <instanceid>
]

$espcovoff;

$espcovon;
```

Coverage Utilities

```
espcovmerge [-spec <file>] <coverage db input file>+
                                                [-db <file>] [-l <log>]
espcov [<options>] [-spec <file>] [<coverage_db_file>]
                                                              [-l <log>]
  OPTIONS
       -printdetail "<listofcovtypes>"
       -filterdetail
       -unusedfilters
```

Coverage Specification File for espcovmerge

```
 - Comment lines in the specification file start with # in
column 1
 - map <dumpscope> <newscope>
 - rename <oldname> <newname>
```

Coverage Specification File for espcov

```
 - Comment lines in the specification file start with # in
column 1
 - select (<num>,<fullinstancepath>)
 - printdetail <covtypelist>
 - filter <covtype> <fully_scoped_identifier>
[<missingtoggle>]
     <missingtoggle> ::= norise|nofall (Only available for
covtype SA)
<fully_scoped_identifier> ::= [<instancepath>'.']
<net_name>
                                                    [<bit_enum>]
<bit_enum> ::= '[' <enum_range> [ ',' <enum_range>] ']'
<enum_range> ::= <num> | <num> '-' <num>
```

Instance Path Definition

An instance path usually begins with a module name, typically the top module name. If the first element is not a module, it must be an instance identifier that exists in the current module. Each element in the instance path is separated by a period (.).

When used in a filter statement, the first element must be a module name. When used in the `$espcovscope` statement, the first instance identifier is the name of an instance in the module where the `$espcovscope` is located.

Index

Symbols

B

C

D

E

T

U

W