Course 12

Code Coverage

(Statement, Branch, Toggle, Expression Coverage)

Rev. Oct 11, 2012 Applies to Active-HDL 9.2 and later

Table of Contents

Overview

Code Coverage aids the verification process by providing information in details whether and how the design is verified or which parts of the design are still untested. Code Coverage can provide several different types of information related to the verification process of your design:

- **Statement Coverage** gives detailed information on statements that are executed during the design simulation. It examines each executable statement and checks how many times it has been executed. This information provides feedback on which parts of the design were verified and which are untested. It also helps to locate dead code.
- **Branch Coverage** examines branches of the 'if' or 'case' statement and checks how many times a true or false condition was met by each branch during the simulation.
- **Toggle Coverage** measures design activity in terms of changes in signal logic values.
- **Expression Coverage** and **Condition Coverage** factorize all conditional logical expressions used in statements and monitor them during simulation.
- **Functional Coverage** shows statistics for OVA, PSL, and SystemVerilog assertions and cover statements. It shows how many assertions started to evaluate during simulation, how many failed, and how many succeeded. For the cover statements, the statistics show the number of "matches".

The results of statement, branch, functional, and expression/condition coverage processes are generated and stored in a file after the simulation has been finished and the standalone viewer presents the results in a graphical form.

Statement and Branch Coverage

Overview

Statement Coverage shows execution branches for each HDL statement. This information provides feedback on which parts of the design were verified and which are untested. It also helps to locate dead code. **Branch Coverage** collects execution branches for ***if*** and ***case*** constructs.

Statement Coverage and **Branch Coverage** data can be acquired by simply passing the ***-cc*** argument to the ***asim*** command.

```
asim –cc [<library>].<unit>
```

Code Coverage data collected during simulation is saved to an output directory. The default name of the output directory is **coverage**. It is also possible to write results at any time during simulation using the **coverage write** command at the simulator prompt.

```
coverage write [-src] <directory>
```

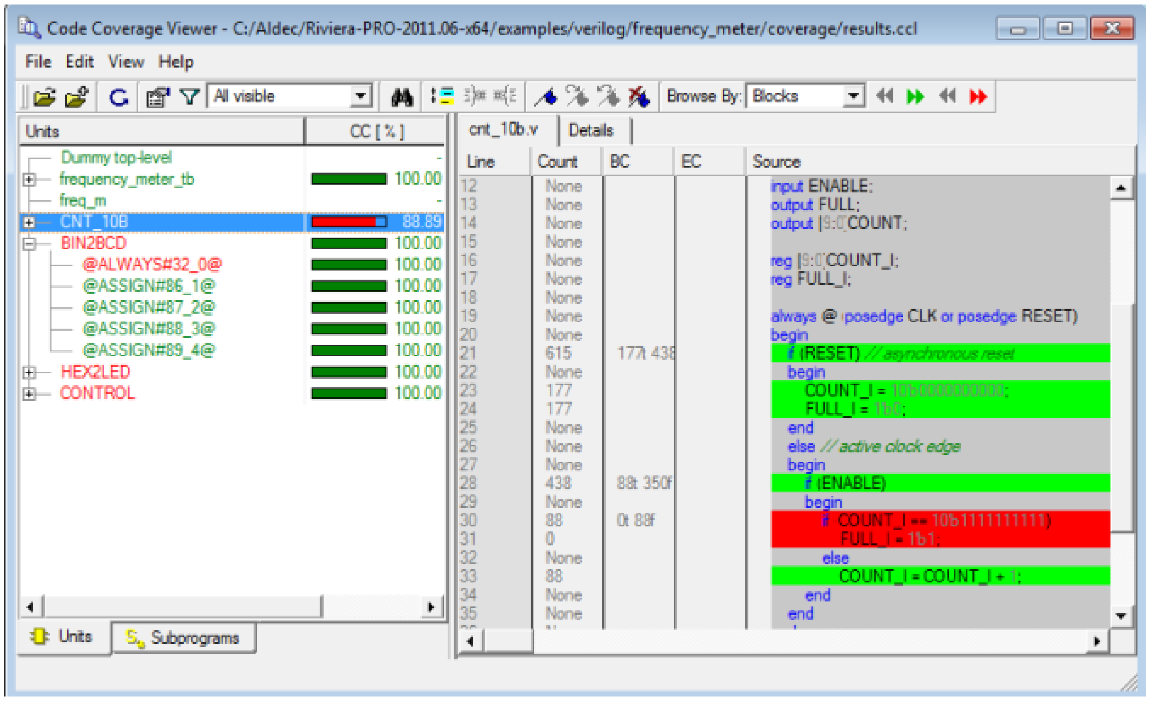
Code Coverage results can be viewed in a dedicated **Code Coverage Viewer**, by going to menu **File | Open** and open the results.ccl or exc_data.exd file from the directory where coverage data was written.

Additionally, you can generate Code Coverage reports using the **coverage report** command. Reports can be created in html, plain text or csv (comma separated value) formats.

```
coverage report [-<type> <filename.type>]
```

The Code Coverage engine in Active-HDL offers a variety of advanced functions:

- Code Coverage data can be collected either on a per-unit or per-instance basis. If a design contains several instantiations of a library unit, Code Coverage data can either be merged for all instance of that unit or collected separately for each instance. Data can be collected on a per instance basis also for objects created with the generate statement.



Code Coverage Viewer - C:/Aldec/Riviera-PRO-2011.06-x64/examples/verilog/frequency_meter/coverage/results.ccl

Units	CC [%]
Dummy top-level	-
frequency_meter_tb	100.00
freq_m	-
CNT_10B	88.89
BIN2BCD	100.00
@ALWAYS#32_0@	100.00
@ASSIGN#86_1@	100.00
@ASSIGN#87_2@	100.00
@ASSIGN#88_3@	100.00
@ASSIGN#89_4@	100.00
HEX2LED	100.00
CONTROL	100.00

```
Line  Count  BC        EC  Source
12    None                 input ENABLE;
13    None                 output FULL;
14    None                 output [9:0]COUNT;
15    None
16    None                 reg [9:0]COUNT_I;
17    None                 reg FULL_I;
18    None
19    None                 always @ (posedge CLK or posedge RESET)
20    None                 begin
21    615    177t 438          if (RESET) // asynchronous reset
22    None                     begin
23    177                          COUNT_I = 10'b0000000000;
24    177                          FULL_I = 1'b0;
25    None                     end
26    None                     else // active clock edge
27    None                     begin
28    438    88t 350f              if (ENABLE)
29    None                         begin
30    88     0t 88f                    if (COUNT_I == 10'b1111111111)
31    0                                    FULL_I = 1'b1;
32    None                             else
33    88                                   COUNT_I = COUNT_I + 1;
34    None                         end
35    None                     end
```

Figure 1 Code coverage Viewer

- Code Coverage sessions can be controlled interactively with macro commands during the simulation process. Collection of Code Coverage data can be disabled for any design region at any time during simulation. Once the Code Coverage session is disabled, the user can enable it again when simulation reaches the desired time point.
- Code Coverage data can be collected only for objects from the current library or include objects from the resource libraries as well.
- Code Coverage results collected in two simulation runs (e.g. with two different set of test vectors) can be merged using the coverage merge command.
- Code Coverage data can be viewed with a stand-alone Code Coverage Viewer. The viewer shows a synchronized display of the design structure and the source code with Code Coverage data. Optionally, statistics displayed by the Code Coverage Viewer can be supplemented with Expression Coverage data.

There are two preconditions for collecting Code Coverage data:

- Generation of debug information should not be disabled during compilation of source files (both VHDL and Verilog). The debug information is created if the -dbg argument is passed to the VHDL compiler (acom) and/or Verilog compiler (alog).
- SLP acceleration should be disabled with the –O2 argument for the asim command (if –O5 is used, Code Coverage will not be available for SLP accelerated Verilog objects)

The following commands and arguments are used to control Code Coverage sessions.

Enabling Code Coverage using script

The Code Coverage engine must be enabled when simulation is initialized. To enable Code Coverage, pass the `-cc` or `-coverage` argument to the `asim` command.

Additionally, you can use the `-cc_all` and `-cc_hierarchy` arguments to control the behavior of the Code Coverage engine.

The **`–cc_all`** argument enables collection of Code Coverage data for objects located in resource libraries. By default, Code Coverage data is collected only for objects from the working library.

The **`–cc_hierarchy`** argument enables the hierarchical mode. In the hierarchical mode Code Coverage data is collected separately for every instance of each library unit. (If a design includes 100 instances of the CB4 counter, the Code Coverage data will be collected separately for each of the 100 instances.) By default the hierarchical mode is not enabled and Code Coverage data is merged for all instances of each library unit.

For instance, you may use the following asim command to enable code (statement and branch) coverage generation:

```
asim –o2 -lib work -coverage -cc_all -cc_hierarchy frequency_meter_tb
```

Note that for Verilog designs you have to use –o2 switch to disable SLP acceleration. Collection of code coverage data is not supported in the SLP accelerated simulations.

Controlling Code Coverage Sessions

Code Coverage sessions can be controlled with the **coverage** commands:

coverage clear | disable | enable | merge | off | report | write

The commands are used to:

- temporarily disable and enable Code Coverage engine (coverage disable and coverage enable)
- temporarily disable and enable collection of Code Coverage data for the specified regions of the design (coverage disable <instance_path> and coverage enable <instance path>)
- switch the Code Coverage engine off (coverage off)
- write Code Coverage data to the specified location (coverage write)
- clear Code Coverage data (coverage clear)

Processing Code Coverage Data

The **coverage** command is also used to process Code Coverage data. Use the coverage command to:

- generate reports with Code Coverage data; you can generate html, plain text, and comma-separated value reports (cc report)
- merge or manipulate Code Coverage data from reports (cc merge)

Viewing Code Coverage Data

Code Coverage data can be viewed in a dedicated **Code Coverage Viewer**. To start the coverage viewer go to menu **Tools** and select **Code Coverage Viewer**. Coverage viewer opens with an empty window. To load the Code Coverage data, open the ***results.ccl*** file from the directory where Code Coverage data was written.

Coverage Pragmas for the Compilers

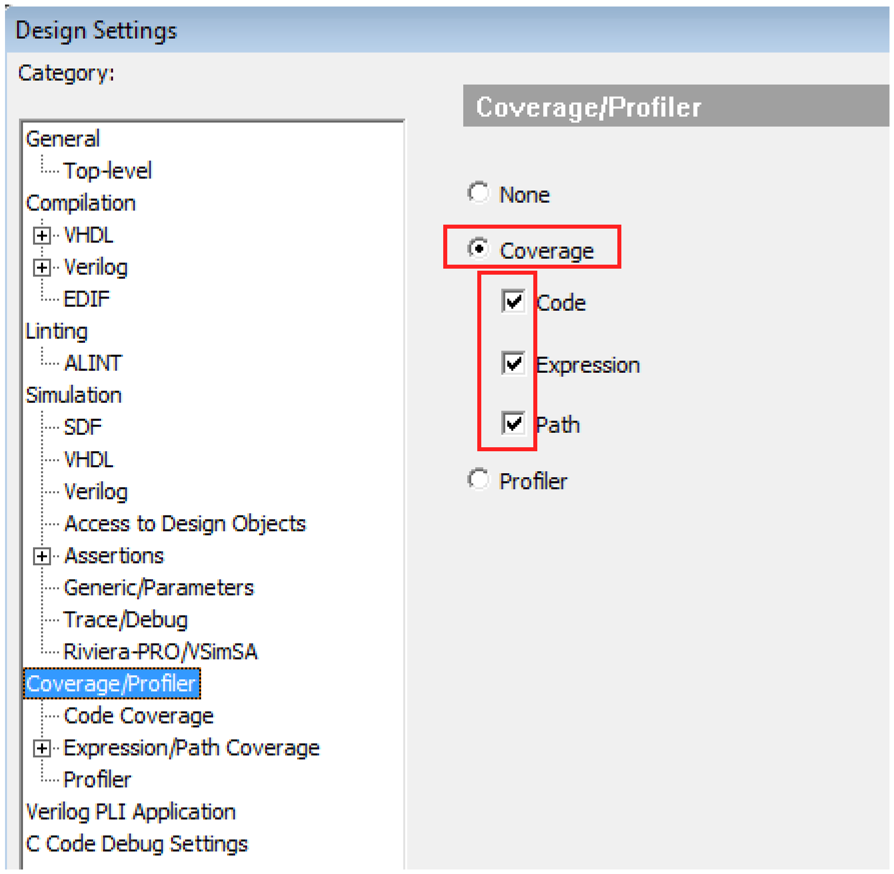
It is possible to disable collection of Code Coverage data for selected code blocks using special compiler pragmas in VHDL or Verilog files. This is useful in the following cases:

- Disabling Code Coverage for units that do not require Code Coverage statistics; for example IP blocks or units reused from previous designs and known to be thoroughly tested and verified.
- Disabling Code Coverage for statements that are supposed never to execute; for example statement that report assertion failures. If all such statements are masked, then you can aim for 100% Code Coverage.

The pragmas are available both for VHDL (`--vhdl_cover_off` and `--vhdl_cover_on`) and Verilog (`//verilog_cover_off` and `//verilog_cover_on`).

Enabling Code Coverage using GUI

Before initializing simulation you need to enable code coverage by selecting coverage from **Design | Settings | Coverage/Profiler**. Here you can select which type of coverage you want to run for respective design. You can select code, expression and path (for VHDL only) coverage.

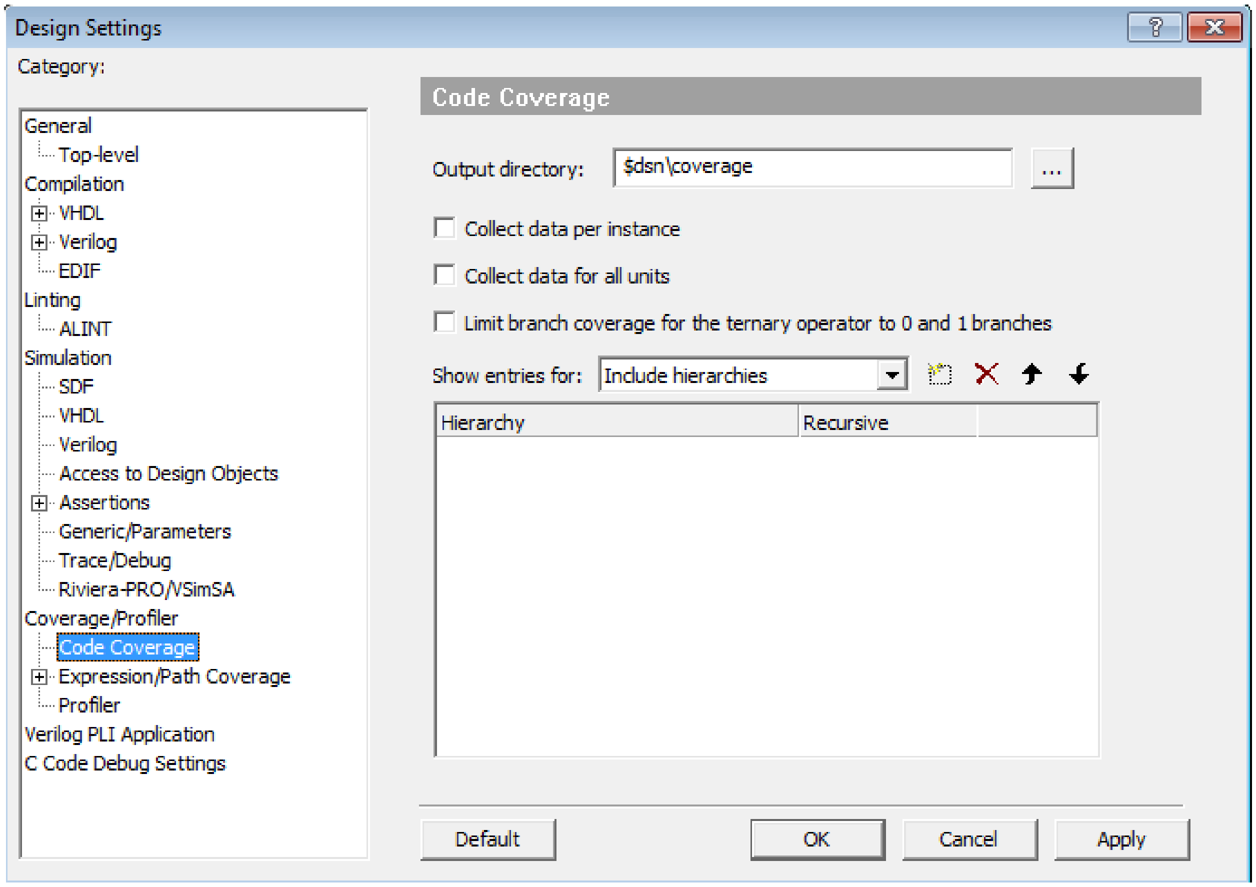


None [✓ Coverage] Profiler

[✓] Code
[✓] Expression
[✓] Path

Figure 2 Design settings for code coverage

You can set different options by using **Design | Settings | Code Coverage** menu. You can set option for region you want to collect code coverage. You select per instance, for all units and you can exclude hierarchies for coverage by adding them in the list.



Output directory: $dsn\coverage

☐ Collect data per instance

☐ Collect data for all units

☐ Limit branch coverage for the ternary operator to 0 and 1 branches

Show entries for: Include hierarchies

Figure 3 Code coverage options using GUI

An Example

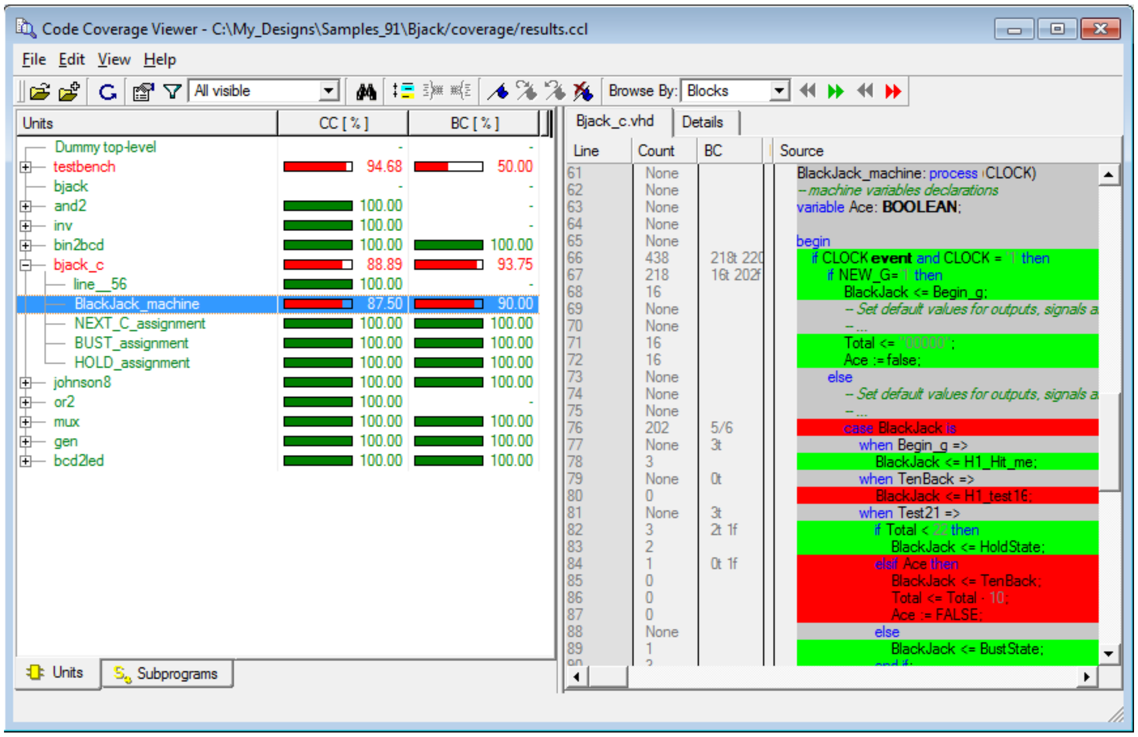
Open the sample design/workspace “bjack” from C:/My_Designs/Samples_91/Bjack folder. Open the Functional.do file from Design Manager and notice how coverage is enabled by the following command:

```
asim  -advdataflow  -cc  -cc_dest  $dsn/coverage  -cc_hierarchy  -cc_all  testbench
      testbench_arch
```

Also take note that you can use system command to open the coverage report file:

```
      ! "$aldec/BIN/ccviewer.exe" $dsn/coverage/results.ccl
```

When you execute above command the coverage report will be open as shown below:



Units	CC [%]	BC [%]
Dummy top-level	-	-
testbench	94.68	50.00
bjack	-	-
and2	100.00	-
inv	100.00	-
bin2bcd	100.00	100.00
bjack_c	88.89	93.75
line__56	100.00	-
BlackJack_machine	87.50	90.00
NEXT_C_assignment	100.00	100.00
BUST_assignment	100.00	100.00
HOLD_assignment	100.00	100.00
johnson8	100.00	100.00
or2	100.00	-
mux	100.00	100.00
gen	100.00	100.00
bcd2led	100.00	100.00

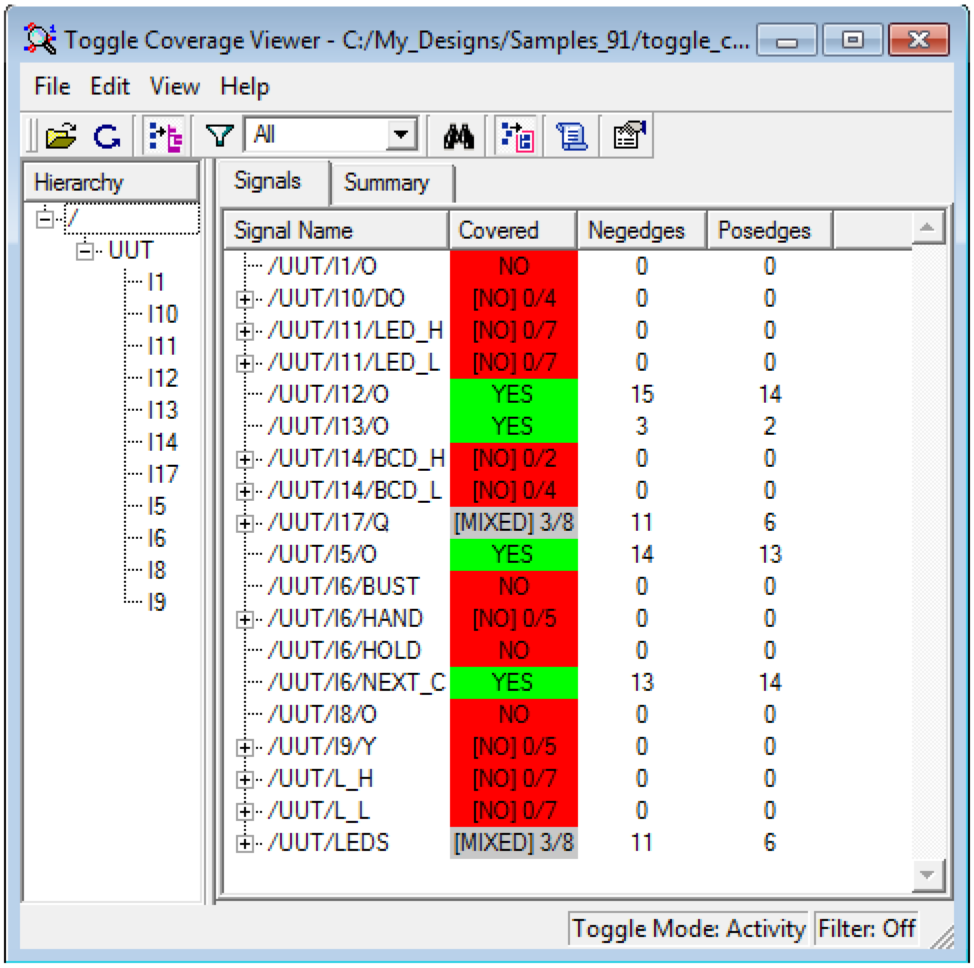
```
Line  Count  BC         Source
61    None              BlackJack_machine: process (CLOCK)
62    None              -- machine variables declarations
63    None              variable Ace: BOOLEAN;
64    None
65    None              begin
66    438    218t 220     if CLOCK event and CLOCK = '1' then
67    218    16t 202f       if NEW_G='1' then
68    16                      BlackJack <= Begin_g;
69    None                    -- Set default values for outputs, signals a
70    None                    -- ...
71    16                      Total <= "00000";
72    16                      Ace := false;
73    None                  else
74    None                    -- Set default values for outputs, signals a
75    None                    -- ...
76    202    5/6              case BlackJack is
77    None   3t                 when Begin_g =>
78    3                           BlackJack <= H1_Hit_me;
79    None   0t                 when TenBack =>
80    0                           BlackJack <= H1_test16;
81    None   3t                 when Test21 =>
82    3      2t 1f                if Total < 22 then
83    2                             BlackJack <= HoldState;
84    1      0t 1f                elsif Ace then
85    0                             BlackJack <= TenBack;
86    0                             Total <= Total - 10;
87    0                             Ace := FALSE;
88    None                        else
89    1                             BlackJack <= BustState;
```

Figure 4 Stand alone Code Coverage viewer

Toggle Coverage

Toggle coverage measures design activity in terms of changes in signal logic values. Toggle coverage reports provide the following information:

- whether monitored signals were initialized
- whether monitored signals experienced rising and/or falling edges
- the number of rising and falling edges during simulation



Signal Name	Covered	Negedges	Posedges
/UUT/I1/O	NO	0	0
/UUT/I10/DO	[NO] 0/4	0	0
/UUT/I11/LED_H	[NO] 0/7	0	0
/UUT/I11/LED_L	[NO] 0/7	0	0
/UUT/I12/O	YES	15	14
/UUT/I13/O	YES	3	2
/UUT/I14/BCD_H	[NO] 0/2	0	0
/UUT/I14/BCD_L	[NO] 0/4	0	0
/UUT/I17/Q	[MIXED] 3/8	11	6
/UUT/I5/O	YES	14	13
/UUT/I6/BUST	NO	0	0
/UUT/I6/HAND	[NO] 0/5	0	0
/UUT/I6/HOLD	NO	0	0
/UUT/I6/NEXT_C	YES	13	14
/UUT/I8/O	NO	0	0
/UUT/I9/Y	[NO] 0/5	0	0
/UUT/L_H	[NO] 0/7	0	0
/UUT/L_L	[NO] 0/7	0	0
/UUT/LEDS	[MIXED] 3/8	11	6

Toggle Mode: Activity Filter: Off

Figure 5 Stand alone Toggle Coverage viewer

Toggle coverage reports help to verify the quality of the stimulus and locate dead or unused structure in the design. Signals which were not initialized during simulation or are not exercised properly by the testbench can be easily identified.

Toggle coverage sessions are started with the **`toggle`** command after simulation has been initialized. Signals which should be monitored must be specified explicitly on the command line.

```
toggle <object>
```

A toggle coverage report is written automatically when the **`endsim`** command is used or when the toggle coverage engine is switched off with the **`toggle -off`** command. Toggle reports can also be written at any time during simulation with the **`toggle -write`** command.

The **`toggle -disable`** command temporarily disables collection of toggle coverage data. Toggle coverage can be re-enabled with the **`toggle -enable`** command. The **`toggle -clear`** command clears toggle coverage data collected so far without saving it to a file.

Toggle Coverage Modes

The toggle coverage engine can generate different types of reports depending on the mode of operation. Four modes are available:

`Init Mode`

In the init mode (`-toggle_type init`) a signal is considered toggled if it had a known value (i.e. 0 or 1) after the last delta cycle at any simulation time. If there are several deltas at one simulation time, signal values before the last delta are ignored.

If the toggle coverage engine is configured not to ignore delta cycles then a signal is considered toggled if it has a known value after any delta. (Deltas are not ignored if the toggle command is invoked with the -deltas switch.)

Signals which meet the criterion for being considered toggled are no longer monitored to minimize simulation overhead.

Assign Mode

The assign mode (`-toggle_type assign`) provides information on the number of pulses that happened on a signal while the signal was being monitored. Data collected in the assign mode is more elaborate than in the init mode, albeit at the cost of increased simulation overhead. All signals specified by the user are monitored till the end of simulation (or until the `toggle -off` command is used).

In Verilog designs, the toggle coverage engine counts 0 and 1 pulses. In VHDL the L and H values of the std_logic type are also included in the statistics. Delta cycles are by default ignored. If a 101 transition happens on a signal at one simulation time, no zero pulse is recorded. The `-deltas` switch changes this behavior and includes delta cycles in the statistics.

Full Mode

In the full mode (`-toggle_type full`), a monitored signal is considered toggled if both a rising and a falling edge happen on it during simulation. The requirement for both a rising and a falling edge can be weakened with the `-single_edge` switch. If the `-single_edge` switch is used, only one edge, either rising or falling, is required to recognize the monitored signal as toggled.

Signals which met the criterion to be considered toggled are no longer monitored. This minimizes the simulation overhead.

Activity Mode

The activity mode (`-toggle_type activity`) provides information on how many rising and falling edges happened on a signal while the signal was being monitored. Data collected in the activity mode is more elaborate than in the full mode, albeit at the cost of increased simulation overhead. All signals specified by the user are monitored till the end of simulation (or until the `toggle -off` command is used).

Delta cycles are by default ignored. A 010 transition happening at one simulation time is not treated as a rising edge followed by a falling edge. The`-deltas` switch changes this behavior and includes delta cycles in the statistics.

The mode of operation is specified with the **`–toggle -type`** switch passed to the toggle command. If the switch is omitted, the mode of operation is set in the **Simulation | Toggle Coverage | Parameters | Toggle type** option in the **Preferences** dialog box. By default, the init mode is used.

An Example

Using Toggle Command

The toggle command enables collecting of signal toggle data after simulation has been initialized. The following line of command shall request toggle coverage data collection in **Activity Mode** and specify the format (xml) and location of the report file:

```
toggle -xml -o ./toggle/tgl_report.xml -toggle_type activity -rec *
```

Using GUI

Before enabling toggle coverage you need to set your preferences such as toggle coverage mode, reporting formats, etc. To access the settings in GUI you can go to Structure tab right click on any instance and click Toggle On under Toggle Coverage option of the right click context menu. This can be done only after simulation is initialized.

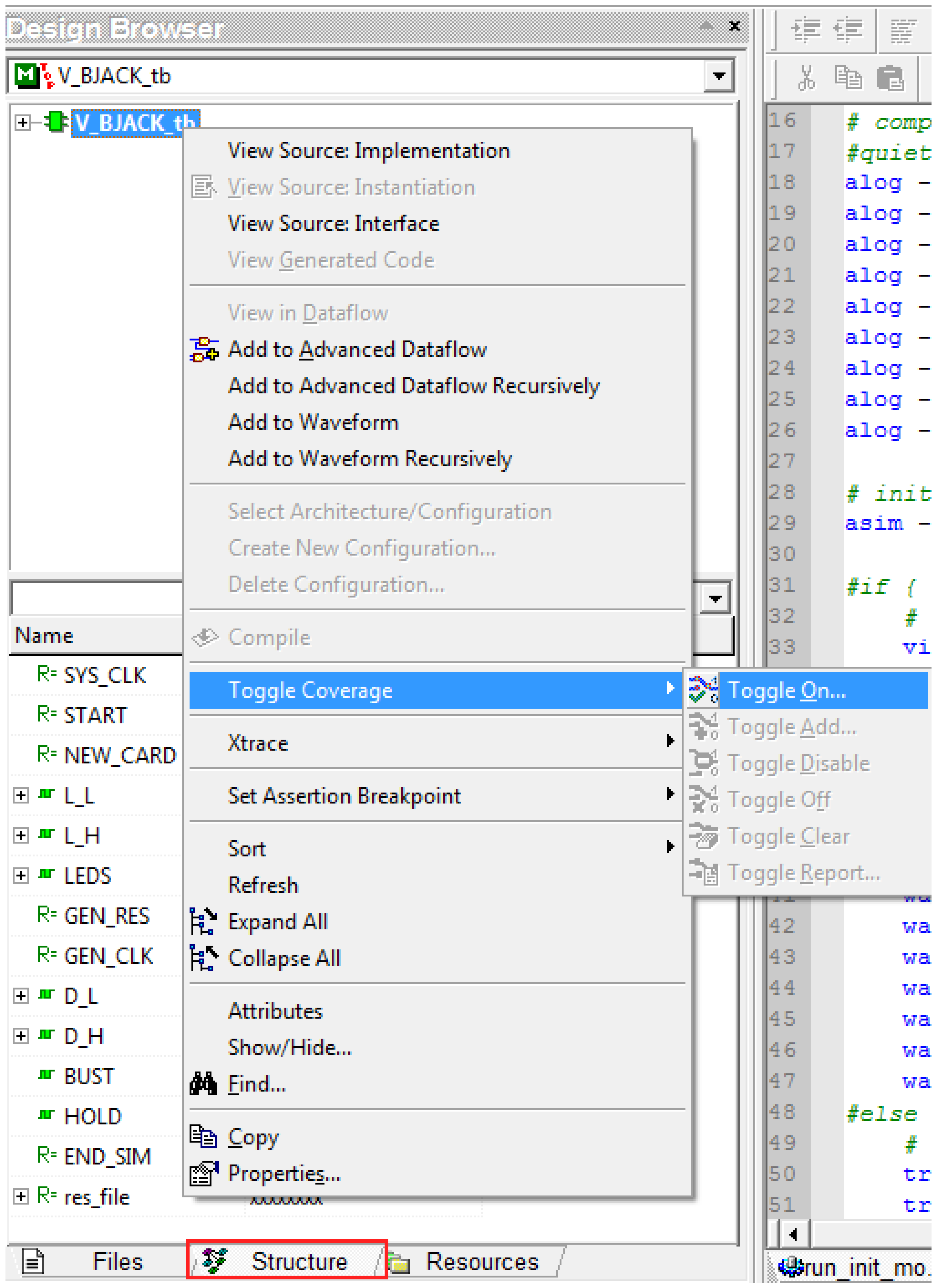
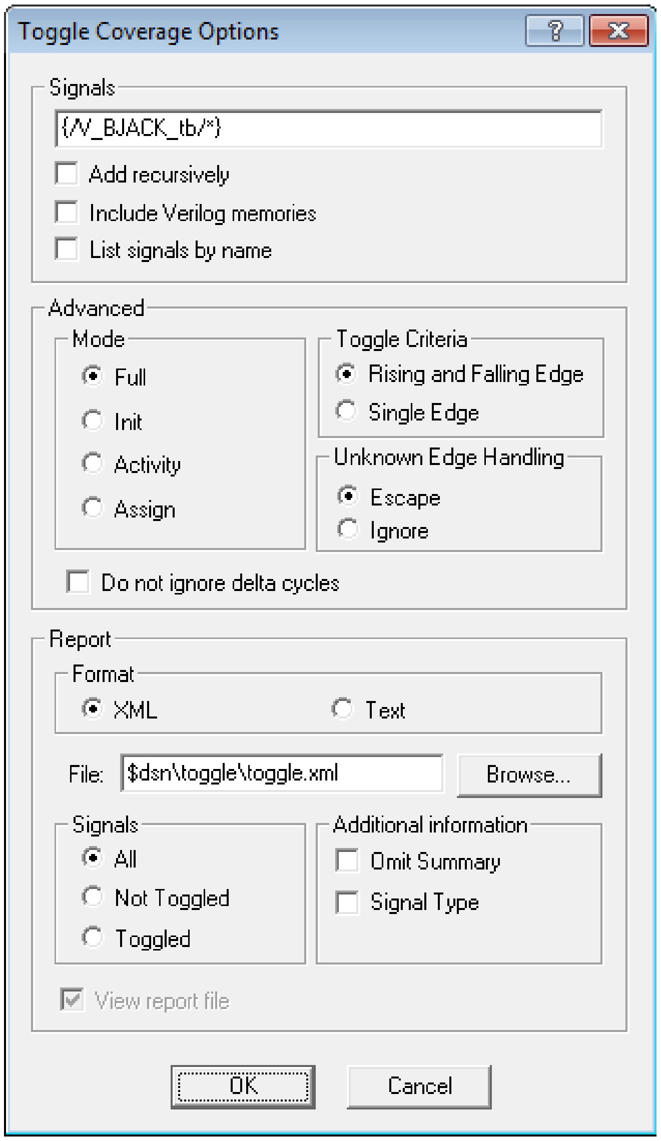


Figure 6 Enabling Toggle settings

Once you click Toggle On, Toggle Coverage Options window will open. Here you can set different options related to Toggle Coverage.


Toggle Coverage Options

Signals

{/V_BJACK_tb/*}

☐ Add recursively
☐ Include Verilog memories
☐ List signals by name

Advanced

Mode: [✓ Full] Init Activity Assign

Toggle Criteria: [✓ Rising and Falling Edge] Single Edge

Unknown Edge Handling: [✓ Escape] Ignore

☐ Do not ignore delta cycles

Report

Format: [✓ XML] Text

File: $dsn\toggle\toggle.xml Browse...

Signals: [✓ All] Not Toggled Toggled

Additional information: ☐ Omit Summary ☐ Signal Type

☑ View report file

OK Cancel

Figure 7 Toggle coverage Options

Once you have set all your preferences you are ready to enable toggle coverage. For that, you will need to let the Active-HDL tool know which signals/units to monitor by right click on the signals/units within **Structure** tab in Design Browser window. For example, if you would like to monitor all signals under UUT, you can simply right click on it and select to Add to Toggle Coverage, as shown below:

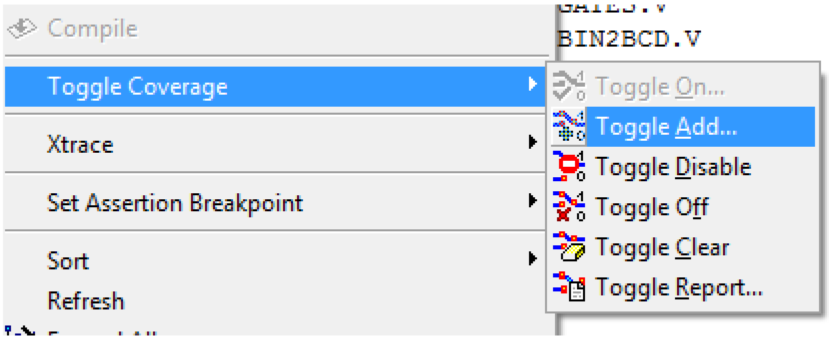


Figure 8 Toggle Coverage menu

Expression Coverage

Expression Coverage is a debugging tool that factorizes logical expressions and monitors them during simulation. An expression is fully covered when all of the expression cases are exercised.

An obvious thorough test is to apply all available test-vectors (combinations of ones and zeros) to the inputs of this expression. However, due to a number of possible combinations to simulate, this type of the test is not efficient and rarely used in practice.

Instead of doing this, you can provide your own set of test-vectors and Expression Coverage will help you find out how optimal they are and ensure that the expression has been fully exercised. The tool will also provide statistics for all logical assignment statements. Be aware that when a value of one of expression operands is x or z in Verilog or in VHDL other than 0, 1, L, or H during simulation then a whole expression is passed over by the coverage tool. (This rule may not apply to relational operators used in expressions employing operands of enumeration types.)

Supported Operators

The following operators are supported by Expression Coverage:

VHDL

Operator(s)	Name
= /= > >= < <=	Relational
And	Logical AND
Or	Logical OR
Nand	Negation of logical AND
Nor	Negation of logical OR
Xor	Exclusive OR
Xnor	Negation of exclusive OR

Verilog HDL

Operator(s)	Name
> >= < <=	Relational
&&	Logical AND
||	Logical OR

==	Logical Equality
!=	Logical Inequality
===	Case Equality
!==	Case Inequality
&	Bit-wise AND
|	Bit-wise Inclusive OR
^	Bit-wise Exclusive OR
^~ or ~^	Bit-wise Equivalence
&	Reduction AND
~&	Reduction NAND
|	Reduction OR
~|	Reduction NOR
^	Reduction XOR
~^ or ^~	Reduction XNOR

SystemVerilog

In addition to operators of Verilog HDL, the following SystemVerilog operators are supported by Expression Coverage:

Operator(s)	Name
&=	Bit-wise AND
|=	Bit-wise Inclusive OR
^=	Bit-wise Exclusive OR

==?	Wild card Equality
!=?	Wild card Inequality

Expression Coverage Limitations

Expression Coverage does not collect statistics for:

VHDL

- Units using selected VITAL procedures and functions

Verilog

- Conditional (?:) operator
- event_control specification of procedural assignments
- Statements in parallel blocks (fork-join)
- Arguments for PLI/VPI function calls and PLI/VPI tasks

Expression Coverage Working Modes

Active-HDL allows collecting expression statistics for the entire design or enable expression scoring for selected entities or modules of your design (by compiling all or selected source files with the **`-exc`** argument of the **`acom`** or **`alog`** command). The coverage tool offers two scoring modes to evaluate expression testing. You can score your design by using only one selected mode by specifying it explicitly on initialization of simulation. Should you change the working mode, re-initialization of simulation is required.

The following operating modes can be selected for Expression Coverage:

Control

This mode controls expressions of single bit signals and checks whether each input of an expression has contributed to an expression result during simulation. The Control mode also allows evaluation of vector expressions (in this case, reduction OR on each operand is performed).

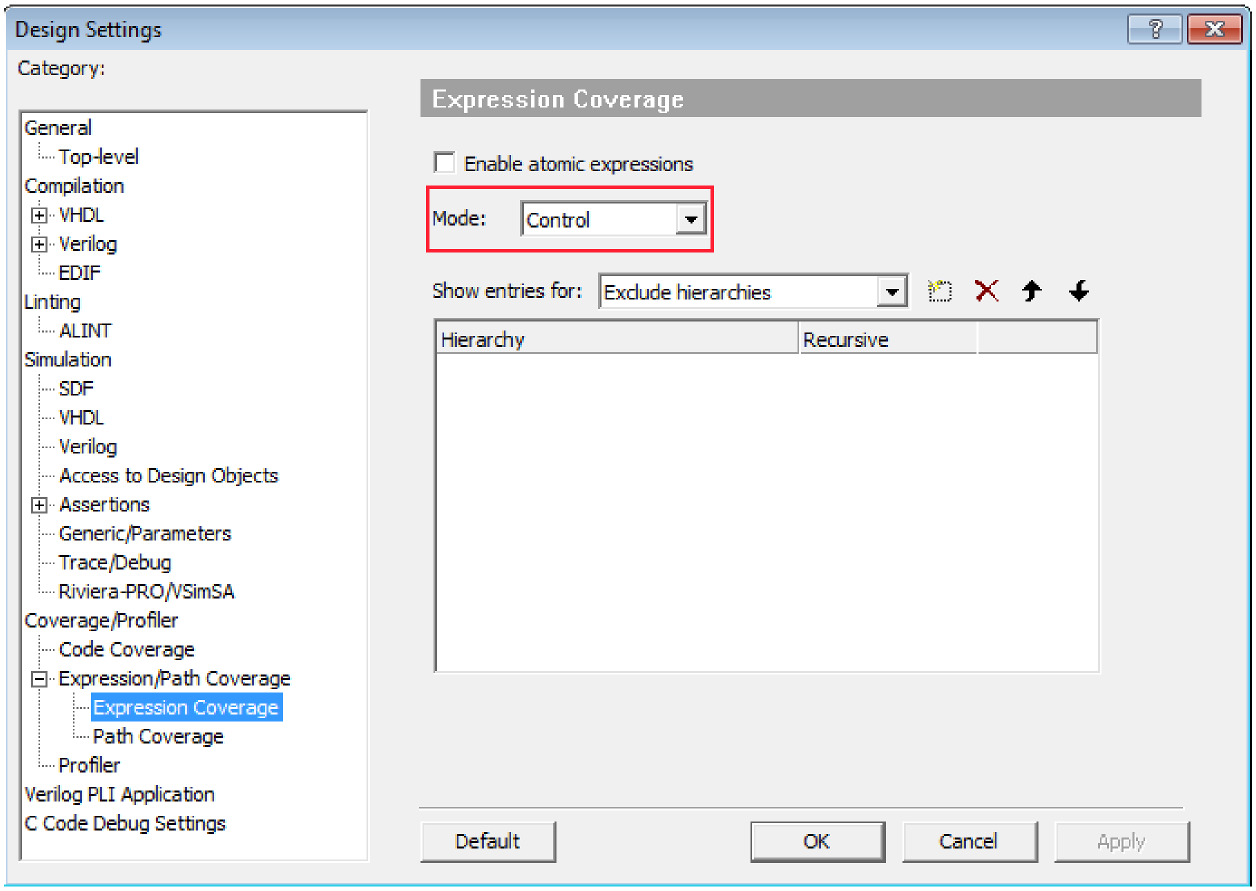
When Control is selected, coverage is scored provided that an expression contains a term that controls a result of an expression. If a term controls a result of the expression, coverage scoring is done and presented in the corresponding coverage truth table listed in an Expression Coverage report.

In case of n-level expressions, the coverage tool will proceed to the next- (lower) level of the expression, and check it for terms controlling the result. Next, it will check if the current level's terms control the expression. If so, statistics will be gathered and saved to the coverage database when you terminate the simulation session.

Vector

This mode is an extension of the Control mode and can be used in case of expressions employing vectors. When you choose this mode, each bit of a multi-bit signal is exercised separately by using the control-based scoring style.

In order to specify the working mode, select it from the **Mode** list box in the **Design Settings** dialog box or pass the **`–exc_control`** or **`–exc_vector`** argument to asim / vsim command.



Enable atomic expressions: ☐

Mode: Control

Show entries for: Exclude hierarchies

Figure 9 Expression coverage settings

Collecting Expression Coverage Data

Expression Coverage factorizes logical expressions in VHDL or Verilog and monitors them during simulation. An expression is fully covered when all of the expression cases are exercised. Collecting Expression Coverage statistics requires just a few steps to perform:

1. Compile source files with the Enable Expression Coverage option enabled (in the Advanced VHDL Compilation or Advanced Verilog Compilation subcategory) or pass the `-exc` argument to the acom or alog command.
   Using the GUI:

   Using the script command:
   ```
   alog -exc ./src/uart.v ./src/transmit_tb.v ./src/receive_tb.v
   ```
2. When initializing simulation, pass the `-exc control | vector` argument to the asim command (i.e. enable Expression Coverage and specify its scoring mode) or specify the settings for Expression Coverage in the **Design Settings** dialog box.

   Using the script command:
   ```
   asim -exc control top_tb
   ```
3. Run the simulation session.
   You can either click on one of the RUN buttons in GUI or use **run** command using script command.
4. After all test-vectors are passed, save Expression Coverage data with the **excoverage write** command. The syntax of the command allows you to customize the format and target location (path) of a coverage report file.

If you use GUI, the report file will be generated in specified location. Or, you can use as an example the command line shown below:

```
excoverage write -html ./expcov/topLevel.html
```

5. In order to create the textual or HTML report (convert it from the binary format), use the **excoverage report** command or run the 'excreport' external program stored in the /bin subdirectory of Active-HDL. For instance, you could use:

```
excoverage report -html ./expcov/topLevel.exd report_01.html
```

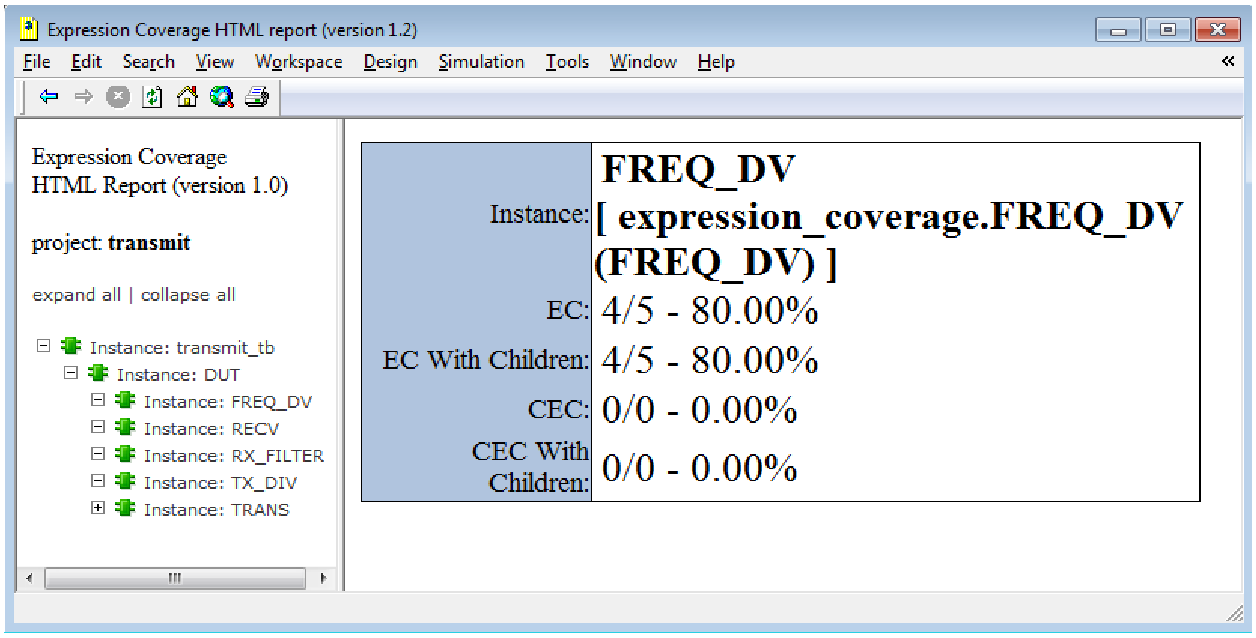
Viewing Expression Coverage Data in coverage viewer

Expression coverage data is saved in .exd format file. This file can be opened using code coverage viewer. To open expression coverage report, go to menu **Tools** and click on **Code Coverage Viewer.** Once the code coverage viewer is opened, go to menu **File | Open** and select generated .exd file.

Expression Coverage Report

The coverage report starts with a header followed by the structural list of instances and statements exercised during simulation. The structure and order of items listed in the report is equivalent to the hierarchy of your design (visible in the Hierarchy Viewer). The form of the report file is similar regardless of the selected working mode.

The HTML report displays coverage data in two frames. The left frame presents the design hierarchy tree while the right one shows coverage statistics for expressions located in the selected design region. The textual report provides the same coverage data but statistics are presented only in the plain text form. You can review the textual report directly in the HDL Editor or any external text editor.


Expression Coverage HTML report (version 1.2)
File Edit Search View Workspace Design Simulation Tools Window Help

Expression Coverage
HTML Report (version 1.0)

project: **transmit**

expand all | collapse all

- Instance: transmit_tb
  - Instance: DUT
    - Instance: FREQ_DV
    - Instance: RECV
    - Instance: RX_FILTER
    - Instance: TX_DIV
    - Instance: TRANS

Instance:	**FREQ_DV [expression_coverage.FREQ_DV (FREQ_DV)]**
EC:	4/5 - 80.00%
EC With Children:	4/5 - 80.00%
CEC:	0/0 - 0.00%
CEC With Children:	0/0 - 0.00%

Figure 10 Expression coverage report

Condition Coverage Report

Condition Coverage data is a subset of statistics produced by the Expression Coverage engine. Condition Coverage shares the database with Expression Coverage, i.e. both coverage statistics are collected simultaneously during the same simulation session.

Condition Coverage statistics can be displayed in the Code Coverage Viewer window, side by side Code Coverage. To load Condition Coverage data, choose **File | Open**, locate an .exd file, and set the filter type in the Open dialog box to Condition Coverage (.exd).

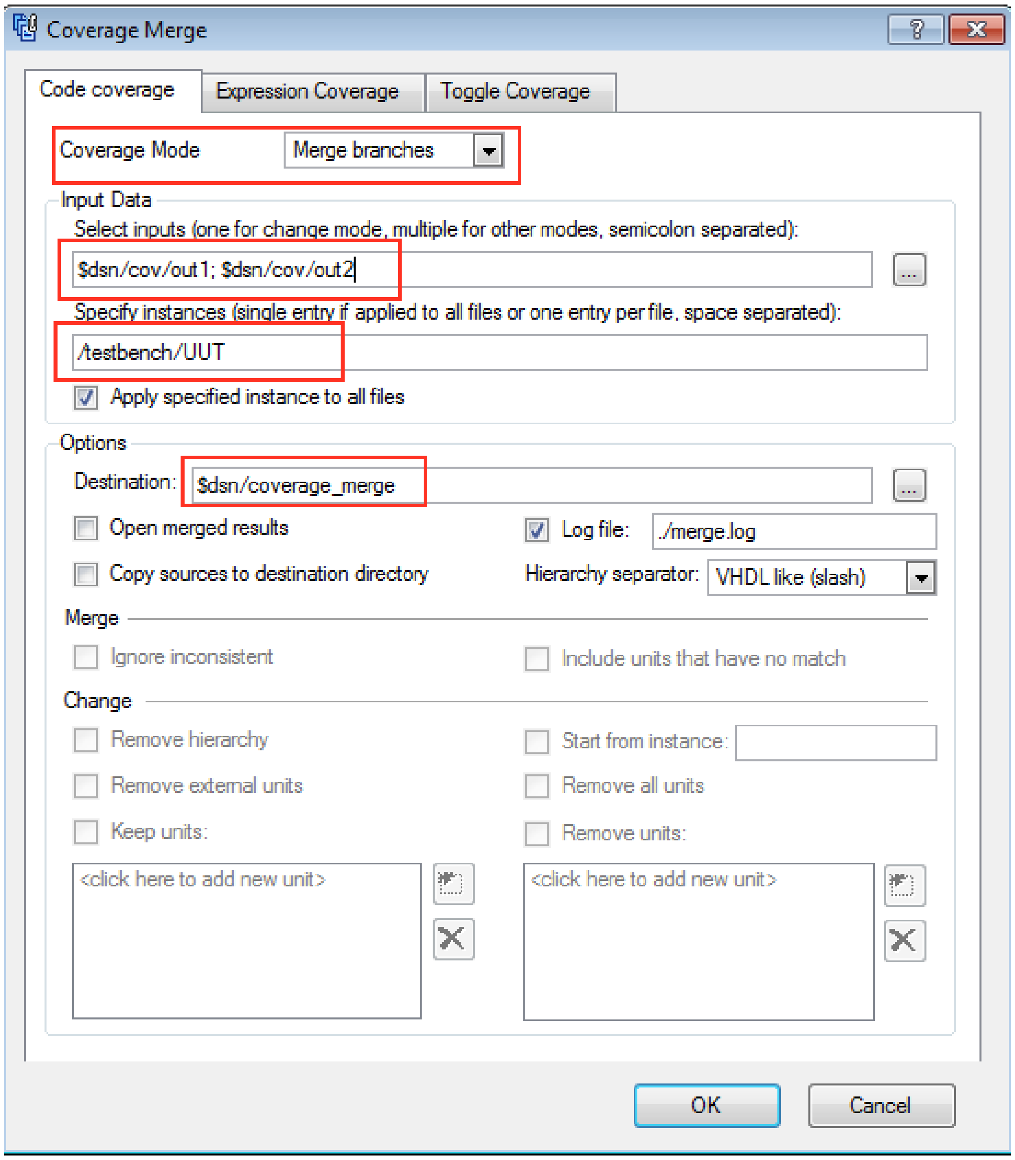
Merging Coverage Results

Coverage results from different simulation runs for example with different set of stimuli. Coverage data can also be removed for selected design regions etc. These operations can be performed two ways: GUI and Script.

Using GUI:

Merging Statement, Branch, Expression and Toggle Coverage reports

Open the **Coverage Merge** dialog box by choosing the **Coverage Merge** option from the **Tools** menu, specify merge options, and press the **OK** button. Open merge results in code coverage viewer.



Coverage Mode: Merge branches

Select inputs (one for change mode, multiple for other modes, semicolon separated): $dsn/cov/out1; $dsn/cov/out2

Specify instances (single entry if applied to all files or one entry per file, space separated): /testbench/UUT

☑ Apply specified instance to all files

Destination: $dsn/coverage_merge

☐ Open merged results ☑ Log file: ./merge.log

☐ Copy sources to destination directory Hierarchy separator: VHDL like (slash)

☐ Ignore inconsistent ☐ Include units that have no match

☐ Remove hierarchy ☐ Start from instance:

☐ Remove external units ☐ Remove all units

☐ Keep units: ☐ Remove units:

<click here to add new unit> <click here to add new unit>

Figure 11 Code coverage Merge

Coverage Mode (Code Coverage/Expression Coverage)

This menu allows you to select the working mode for the Code Coverage or Expression Coverage tool. For more detail information about different mode please refer **Coverage Merge Dialog Box** section under **Product Help > Active-HDL Help > Dialog Box** References.

Select inputs (...)

It allows you to enter the paths and names of input data files to be merged. You should specify exactly two sets of input data when merging Expression Coverage results and two or more sets of input data when merging Code Coverage results in the *Merge Branches* mode.

Specify instances (...)

Specifies hierarchy branches for which the merge should be performed. This option is available only in the *Merge Branches* mode in both Code Coverage and Expression Coverage merge. One hierarchy branch is required for each input listed in the **Select inputs** edit box, unless the **Apply specified instance to all files** option is selected. Hierarchy branches should be separated with a space character.

Destination

Specifies the location for coverage output merged data. The default is *$dsn\coverage_merge* for Code Coverage, *$dsn\excoverage\exc_data.exd* for Expression Coverage, and *$dsn/toggle/toggle_merge.xml* for Toggle Coverage.

Toggle Mode (Under Toggle coverage tab)

It specifies the mode of the Toggle Coverage report (*Init*, *Assign*, *Full*, or *Activity*). Typically, you will select the same mode for the merged output that was used for input reports. If the input reports were generated using different modes, this option can be used to downgrade input reports to the lowest common denominator. Not all reports can be merged. For example, you can "downgrade" Toggle Coverage report in the *Activity* mode to a report in the *Full* mode but a report in the *Assign* mode cannot be combined with the report in the *Full* mode. For a description of all available report modes, see Toggle Coverage Modes.

Using Script:

Merging Statement and Branch Coverage reports

Coverage `merge` or `cc merge` are used to merge or modify coverage results. You can also merge coverage results in the system shell using the `ccmerge` executable from the bin/ subdirectory in the Active-HDL installation directory. Sample commands are shown below:

```
        coverage merge -change -dir <directory>|-ccl <.ccl>
```

or

```
        coverage merge -merge -dir <directory>|-ccl <.ccl>
```

Merging Expression Coverage reports

The command merges binary databases (*.exd*) containing expression coverage data.

The syntax of the command is shown below:

```
excoverage merge -merge_branches { -file <database1> -path <branch1> {-file <database2> -path <branch2>|-rpath <branch2> <replaced_branch1>} ...} -dest <outputfile> [-ignore_inconsistent]
```

or

```
excoverage merge -merge_hierarchies -file <database1> -file <database2> ... -dest <outputfile> [-ignore_inconsistent]
```

Merging Toggle Coverage reports

The command merges two or more toggle coverage reports. The input files for the utility are toggle coverage reports in the XML format. Toggle reports in the text format cannot be combined. The merged output is also in the XML format.

```
toggle -merge [-ignore_inconsistent] [-toggle_type init|assign|full|activity] [-v] -in <xmlfile> -in <xmlfile> [...] -out <xmlfile>
```

Arguments

`-ignore_inconsistent`

It ignores inconsistencies during merge. An inconsistency can happen if the design hierarchies stored in the input report files are not identical, for example some signals can be found in one input file but not in the other. When `-ignore_inconsistent` is used, such inconsistencies are not reported to the console.

`-in <xmlfile>`

The name of input file with the toggle coverage report. This argument must be used at least twice, once for each input report file.

`-out <xmlfile>`

The name of the output file with combined statistics from the first and the second input file.

`-toggle_type init | assign | full | activity`

Specifies the type of the toggle coverage report. This argument can be used to downgrade input results to the lowest common denominator. If the argument is omitted, the command guesses the type of the output automatically.

`-v`

Enables the verbose mode of operation. Extended information about the progress of toggle coverage merge process will be printed to the console.