A P P L I C A T I O N   N O T E

Verifying the Quality of Your Testbench with Code Coverage

www.model.com

Mentor Graphics

The Importance of Testbench Verification

Testbenches have become an integral part of the design process, enabling you to verify that your HDL model is sufficiently tested before implementing your design and helping you automate the design verification process. It is essential, therefore, that you have confidence your testbench is thoroughly exercising your design. Collecting code coverage statistics during simulation helps to ensure the quality and thoroughness of your tests.

In the past, collecting coverage data with Model*Sim* required third party coverage tools that had to be integrated via the PLI (Programming Language Interface) for Verilog or the FLI (Foreign Language Interface) for VHDL. The use of such interfaces required the source code to be instrumented so that facilities for collecting code coverage statistics could be added to the compiled object. This created another step in the verification flow.

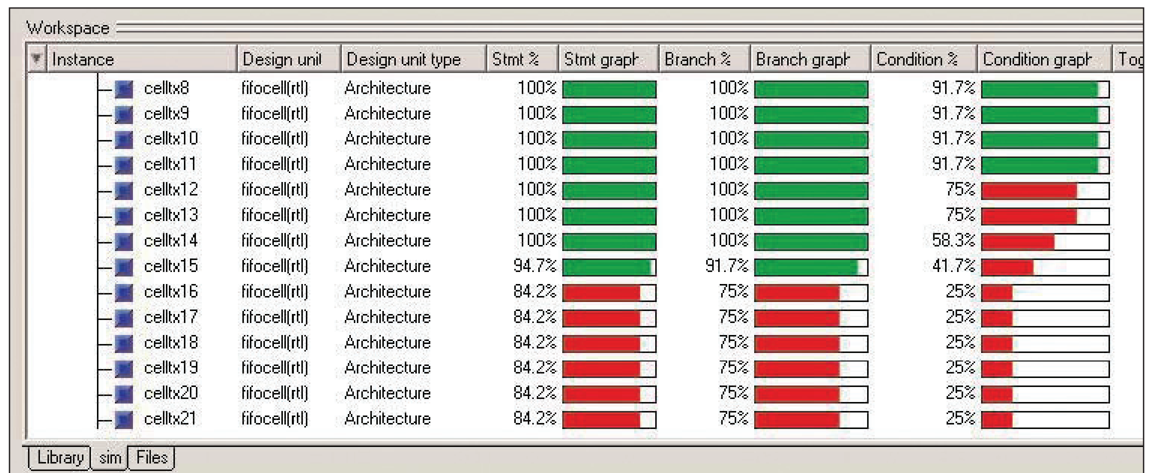
With Model*Sim* version 5.3, Mentor Graphics introduced integrated line coverage built into the kernel of the simulator, giving you the best performance possible without the overhead of an interface layer. In version 5.7, instance based statement and branch coverage were added. Beginning in Model*Sim* 5.8, code coverage was enhanced even further to include expression, condition and toggle coverage. With these integrated coverage collection capabilities, Model*Sim* eliminates the need to maintain a separate design database and provides additional tools for ensuring the quality of your testbench.

Instance Based Coverage

When line coverage was introduced in Model*Sim* 5.3, coverage statistics were based on the source file. This meant that when a line of code was executed from two separate instances of the same design unit, the coverage count was incremented twice. It was not possible to deduce which instance caused a particular piece of code to be executed.

Beginning in Model*Sim* 5.8, instance based coverage was provided for all supported metric types, making it easy to determine which statements within a particular instance have been executed by a particular test. This capability is especially useful for analyzing designs that contain multiple instances of the same design unit.

Figure 1 shows a design with multiple instances of *fifocell*. The structure view in the Workspace pane of the Main window displays the results for each instance. The graph columns provide an immediate visual indication of coverage results. By default, green indicates coverage above 90 percent; red indicates coverage below 90 percent. In figure 1 it's easy to see that all statements and branches in instances *celltx8* through *celltx14* have been executed, whereas coverage in *celltx16* through *celltx21* has not been so complete. This display makes it possible for you to quickly determine which parts of your design are being thoroughly tested by your testbench and which are not.



Workspace

Instance	Design unit	Design unit type	Stmt %	Stmt graph	Branch %	Branch graph	Condition %	Condition graph	Tog
celltx8	fifocell(rtl)	Architecture	100%		100%		91.7%		
celltx9	fifocell(rtl)	Architecture	100%		100%		91.7%		
celltx10	fifocell(rtl)	Architecture	100%		100%		91.7%		
celltx11	fifocell(rtl)	Architecture	100%		100%		91.7%		
celltx12	fifocell(rtl)	Architecture	100%		100%		75%		
celltx13	fifocell(rtl)	Architecture	100%		100%		75%		
celltx14	fifocell(rtl)	Architecture	100%		100%		58.3%		
celltx15	fifocell(rtl)	Architecture	94.7%		91.7%		41.7%		
celltx16	fifocell(rtl)	Architecture	84.2%		75%		25%		
celltx17	fifocell(rtl)	Architecture	84.2%		75%		25%		
celltx18	fifocell(rtl)	Architecture	84.2%		75%		25%		
celltx19	fifocell(rtl)	Architecture	84.2%		75%		25%		
celltx20	fifocell(rtl)	Architecture	84.2%		75%		25%		
celltx21	fifocell(rtl)	Architecture	84.2%		75%		25%		

Library | sim | Files

Figure 1. A design with multiple instances of fifocell.

Statement Coverage

Statement coverage is a measure of the number of executable statements within the model that have been executed during the simulation run. Executable statements are those that have a definite action during runtime and do not include comments, compile directives or declarations.

Statement coverage counts the execution of each statement on a line individually, even if there are multiple statements on that line. Statement coverage statistics are recorded and displayed in the Workspace and Instance Coverage panes the Main window as follows:

- **Stmt count** — the number of executable statements in each file
- **Stmt hits** — the number of executable statements that have been executed in the current simulation
- **Stmt misses** — the number of executable statements that were not executed in the current simulation
- **Stmt %** — the current ratio of Stmt hits to Stmt count,
- **Stmt graph** — a bar chart of Stmt % Statement coverage is a more robust metric than line coverage because it is not affected by coding style. For example, line coverage would treat the following piece of code as one executable line:

```
if RST = ‘0’ then LD <= ‘0’; elsif
CNT = “00” then LD <= ‘1’; else LD <=
‘0’; end if;
```

If the same code were written in the following manner, however,

```
if RST = '0' then
LD <= '0';
elsif CNT = "00" then
LD <= '1';
else
LD <= '0';
end if;
```

line coverage would view it as five executable lines, making code analysis easier to understand. It would also provide a better indication of testbench effectiveness.

With statement coverage, it does not matter how the code is written. Statistics are kept on each statement, individually; making it possible to see which parts of this "if" statement have been executed, even when all five statements are on the same line.

Branch Coverage

Branch coverage, sometimes referred to as decision coverage, counts the execution of expressions and case statements that affect the control flow of HDL execution. Branch coverage statistics are recorded and displayed in the Workspace and Instance Coverage panes of the Main window as follows:

- **Branch count** — the number of executable branches in each file
- **Branch hits** — the number of executable branches that have been executed in the current simulation
- **Branch misses** — the number of executable branches that were not executed in the current simulation
- **Branch %** — the current ratio of Branch hits to Branch count
- **Branch graph** — a bar chart of Branch % In the following example code, the branch can be exercised in two ways.

```
if (z == 0) begin
...
else begin
...
end
```

Branch coverage will report both true and false executions of the *if* statement.

Note that decision coverage still applies if the else statement doesn't exist, as in:

```
if (z == 0) begin
...
end
```

In this case, the if statement can still be evaluated as both true and false. Branch coverage will verify that both the explicit branch (z = 0) and the implicit branch (z != 0) are executed.

Branch coverage also applies to case statements, like the following:

```
case (z)
1: x = 0;
2: x = 10;
3: x = 20;
endcase
```

In this example, branch coverage will identify whether or not each of the three branches of the case statement is taken. It should be noted that the same information could be obtained from statement coverage, since each leg of the case statement is a separate statement. However, in the example:

```
case (z)
1,2: x = 0;
3: x = 10;
endcase
```

statement execution coverage will consider the line 1,2: x = 0; covered if z is 1 or 2. Branch coverage is needed to verify that z has been both 1 and 2 during simulation. The following is another example that shows how branch coverage provides information that statement coverage might miss.

```
IF address = address_bus THEN
choice := 1 ;
END IF ;
data := choice ;
```

If the testbench forced address to always equal *address_bus*, the statement coverage for this fragment of HDL would be 100 percent, since every statement would be executed. However, branch coverage would only be 50 percent, since the FALSE branch (corresponding to "not equal to *address_bus*") would not have been taken during simulation.

Condition Coverage

Condition coverage is an extension of branch coverage. Condition coverage breaks down branch conditions into the elements that make the result true or false. Condition coverage statistics are recorded and displayed in the Workspace and Instance Coverage panes of the Main window as follows:

- **Condition rows** — the number of conditions in each file
- **Condition hits** — the number of times the conditions in a file that have been executed
- **Condition misses** — the number of conditions in a file that have not been executed
- **Condition %** — the current ratio of Condition hits to Condition rows
- **Condition graph** — a bar chart of Condition %

Model*Sim* Condition coverage eliminates unimportant conditions from the display so you don't waste time chasing down irrelevant conditions. This is sometimes referred to as "sensitized condition coverage" or "focused condition coverage," and is illustrated with the following example.

```
PROCESS (ina, inb, inc, ind, datin)
BEGIN
if ina = '1' or inb = '1' or inc =
'1' or ind = '1' then
datout <= datin;
ELSE
datout <= '1';
END IF;
END PROCESS;
```

The *if* statement above has a condition made up of four inputs: *ina, inb, inc,* and *ind*. From these four inputs, a truth table would have 16 different possibilities. We can see from the condition, however, that there are only five conditions of interest:

"ina" controlling the true condition
"inb" controlling the true condition
"inc" controlling the true condition
"ind" controlling the true condition
"ina" "inb" "inc" "ind" all controlling the false condition

You can view the details of a missed condition as a truth table in the Details pane of the Main window. This pane allows the detailed analysis of missed coverage for any particular statement, branch, condition, expression or signal transition (toggle). Figure 2 shows the focused condition coverage truth table for a missed condition in our example.

The truth table contains only those entries that are important to the condition, along with a count value to indicate how many times a particular condition has been ‘hit’ by your testbench during the simulation run.

Expression Coverage

Expression coverage is the same as condition coverage but instead of covering branch decisions it covers concurrent signal assignments. Other code coverage tools combine condition and expression coverage in a single function, but this does not provide the flexibility to control each independently. Model*Sim* treats them separately, giving you more control of the metrics during simulation.

Expression coverage builds a focused truth table based on the inputs to a signal assignment using the same technique as condition coverage. If, for example, we rewrite the condition coverage example above as an expression it will be:

```
internal <= ina or inb or inc or ind
```

And the focused expression coverage truth table will look much like the focused truth table for the condition, shown in Figure 2.

Toggle Coverage

Toggle coverage — the ability to count and collect changes of state on specified nodes — has been a part of Model*Sim* for many years. Model*Sim*’s toggle coverage includes Verilog nets and registers and VHDL signals of type bit, *bit_vector*, *std_logic*, and *std_logic_vector*. In addition, version 5.8 also introduced a number of enhancements to toggle coverage functionality. First, toggle coverage has been integrated as a metric into the coverage tool so that the use model and reporting become the same as the other supported metrics. Second, the basic toggle coverage has been enhanced to have two modes of operation — standard and extended. Standard toggle coverage only counts *low* <-> *high* and *0* <-> *1* transitions. Extended toggle coverage counts these two transitions plus the following four:

```
X or Z --> 1 or H
X or Z --> 0 or L
1 or H --> X or Z
0 or L --> X or Z
```

```
Details
File: test.vhd
Line: 22
Truth table for:
   if ina = '1' or inb = '1' or inc = '1' or ind = '1' then

                ina
                | inb
                | | inc
                | | | ind
                | | | | ((((ina eq '1') or (inb eq '1')) or (inc eq '1')) or (ind eq '1'))
                | | | | | (((ina eq '1') or (inb eq '1')) or (inc eq '1'))
                | | | | | | ((ina eq '1') or (inb eq '1'))
        count   | | | | | | |
        -----   --------------
        49981   1 - - - 1 - -
            0   - 1 - - 1 - -
            0   - - 1 - 1 - -
            0   - - - 1 1 - -
           21   0 0 0 0 0 0 0
            0   unknowns
```

Figure 2. Focused condition coverage truth table in the Details pane.

This extended coverage allows a more detailed view of testbench effectiveness and is especially useful for examining coverage of tri-state signals. It helps to ensure, for example, that a bus has toggled from high 'Z' to a '1' or '0', and a '1' or '0' back to a high 'Z'.

The Signals window has been enhanced to show detailed toggle information, as shown in Figure 3 below. As you can see, there is a column for each of the six transition types.

The last three columns show the percentage toggle figures for each displayed signal. The *%01* column shows the percent coverage of the '*1*' or '*H*' <--> '*0*' or '*L*' transitions. If a signal is a single bit type and has made both transitions it is considered 100 percent covered. If it has only made one of the transitions then it will be 50 percent covered. If the signal is a bus, each bit of the bus will be considered in this percentage. For example, the address signal is a 4-bit bus and one of its bits has not made the '*1*' or '*H*' <--> '*0*' or '*L*' transition, giving an 87.5 percent coverage.

The *%FULL* column shows the calculated percentage of all transitions that have occurred. *%FULL* is calculated by dividing the number of transitions that have occurred by the total number of possible transitions that could have occurred.

The *%XZ* column shows the calculated percentage of *'Z'* or *'X'* transitions that have occurred – a useful measurement for tri-state signals. The calculated percentage is the number of *'Z'* and '*X'* transitions that have occurred during simulation divided by the total number of possible transitions that could have occurred.

Name	Value	1H->0L	0L->1	0L->X	XZ->0L	1H->X	XZ->1	% 01	% Full	% XZ
reset	1	0	1	0	0	0	0	50%	16.67%	0%
address	1000	12	13	0	0	0	0	87.5%	29.17%	0%
rdb	1	521	521	0	0	0	0	100%	33.33%	0%
csb	1	16	16	0	0	0	0	100%	33.33%	0%
wrb	1	519	519	0	0	0	0	100%	33.33%	0%
fifo_ram_datad	UUU...	0	0	0	0	0	0	0%	0%	0%
fifo_full_indicate	0	1	0	0	0	0	0	50%	16.67%	0%
fifo_empty_indicate	1	0	1	0	0	0	0	50%	16.67%	0%
iom_dd	1	38	38	0	0	0	1	100%	50%	25%
memcs	1	1024	1024	0	0	0	0	100%	33.33%	0%
txd	1	100	100	0	0	0	0	100%	33.33%	0%
pdatad	1010...	8	3	1032	1031	1024	1033	43.75%	62.5%	71.88%
(7)	1	0	0	1	1	256	257	0%	66.67%	100%
(6)	0	0	0	257	258	0	0	0%	33.33%	50%
(5)	1	1	0	1	0	256	258	50%	66.67%	75%
(4)	0	0	0	257	258	0	0	0%	33.33%	50%
(3)	1	1	0	1	0	256	258	50%	66.67%	75%
(2)	0	1	0	257	257	0	1	50%	66.67%	75%
(1)	1	3	2	1	0	256	258	100%	83.33%	75%
(0)	0	2	1	257	257	0	1	100%	83.33%	75%
variable_datad	0000...	0	0	21741	21747	25083	25085	0%	66.67%	100%
crcd	ZZZZ	0	0	1754	1754	1174	1174	0%	66.67%	100%

Figure 3. Extended toggle coverage results displayed in the Signals window.

New User Interface

The Model*Sim* user interface is constructed to facilitate code coverage analysis of your testbench. All panes in the Main window, including those specifically for displaying code coverage data — Missed Coverage, Current Exclusions, Instance Coverage and Details — are dockable. You can drag them to any position within the Main window, or tear them out to view them separately.

The structure (sim) tab within the Main window Workspace includes coverage data for each instance, providing information about statement, branch, condition, expression, and toggle coverage within the design. Each column can be selectively toggled on or off to suit your analysis.

The Files tab displayes coverage data on a file by file basis.

The Instance Coverage pane displays a flattened list of instances. The hierarchical block selected in the structure tab, and a user definable filter setting, controls the content of this window. All the instances that are children of the selected instance are displayed in this window as well. A filter threshold can be applied to this list of instances, allowing only blocks with certain coverage percentages above or below a specified threshold to be viewed. This filter can be found by either right-clicking in the Instance Coverage pane and opening the *Filter instance list* dialog, or through the *Threshold* setting in the Main window toolbar, as shown in Figure 4.

It is also possible to sort the Instance Coverage pane by any column. For example, you can sort for the number of missed branches by clicking in the 'Missed Branches' column title. This causes the instance with the highest number of missed branches to be listed first — allowing you to immediately identify instances that are not being exercised by your testbench.

The Missed Coverage pane includes a tab for each coverage metric — statement, branch, condition, expression and toggle. Each tab includes a list of lines that contain missed coverage for the selected metric. The lines displayed in these tabs are determined by the instance selected in either the Workspace Structure (sim) tab or the Instance Coverage pane.

When you select any line in the Missed Coverage tabs, the details of the missed coverage are displayed in the Details pane of the Main window. The format of the Details information depends on the coverage metric selected. For Statement coverage, the Details pane will simply show the selected line of code with a zero count. For Branch coverage, it will show the line of code and the number of times the true and false paths have been taken. For Condition and Expression coverage, it will display the focused truth table and the counts for each of the entries, as we've shown above. And for Toggle coverage it will display the counts and percentage numbers for each transition type.

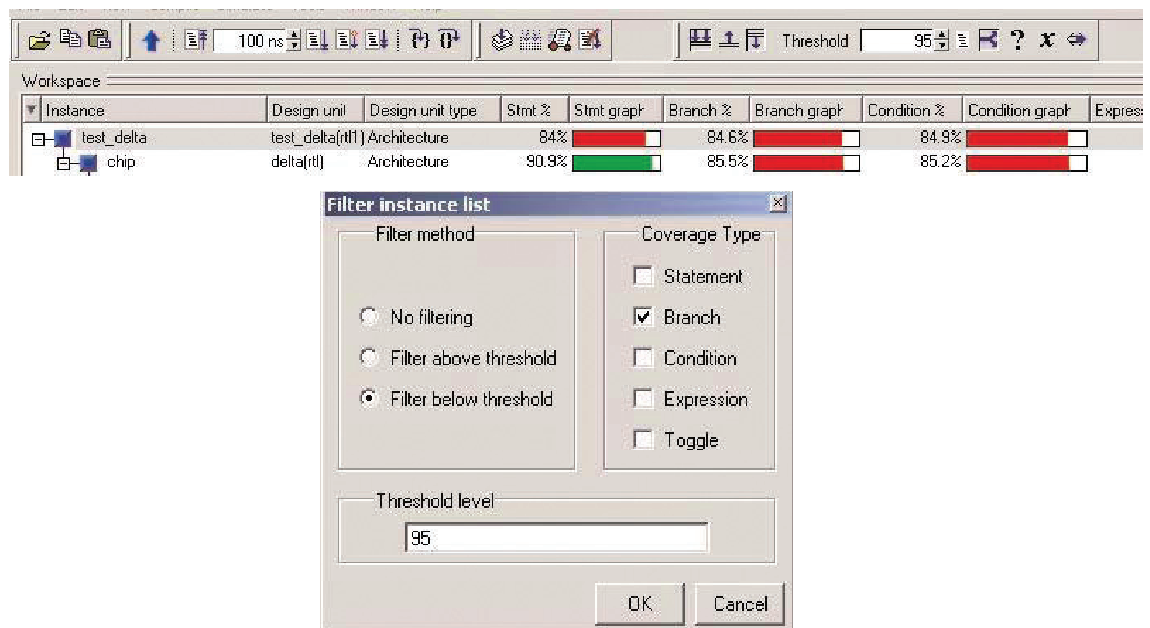


Figure 4. Two methods for setting the coverage threshold for the Workspace and Instance Coverage displays.

Select any line in the Missed Statement, Branch, Condition or Expression tabs and that line will be displayed and highlighted yellow in the Source window. The Source window includes a Hits column for statement coverage and a BC column for branch coverage. A green checkmark indicates coverage; a red 'X' indicates missed coverage. All lines that contain missed statements, branches, conditions or expressions are highlighted pink. Hovering the mouse over

these highlighted lines will cause the actual coverage counts (the number of times a statement has been executed or the number of times a true or false branch has been taken) to be displayed in the Hits and BC columns, as shown in Figure 5.

Coverage Exclusions

Model*Sim* allows you to exclude lines and files from code coverage statistics with popup menus in the GUI, by inserting pragmas in your source code, or by creating an exclusion filter file. To exclude files with popup menus in the GUI, right-click any file in the Main window Workspace and select **Coverage > Exclude Selected File**. To exclude a specific line, rightclick the line in the Missed Coverage pane and click the **Exclude Selection** button that appears. You can also exclude lines and files in the Source window by right-clicking in the Hits column and selecting **Exclude Coverage Line xxx** or **Exclude Entire File**.

Model*Sim* also supports the use of source code pragmas to selectively turn coverage off and on. In Verilog, the pragmas are:

```
// coverage off
// coverage on
```

In VHDL, the pragmas are:

```
-- coverage off
-- coverage on
```

To exclude code, bracket the code with pragmas like this:

```
// coverage off
a = 1'b0;
// coverage on
```

Exclusion filter files specify files and line numbers that you wish to exclude from the coverage statistics. You can create the filter file in any text editor or save the current filter in the Source window by selecting **File > Save >**.

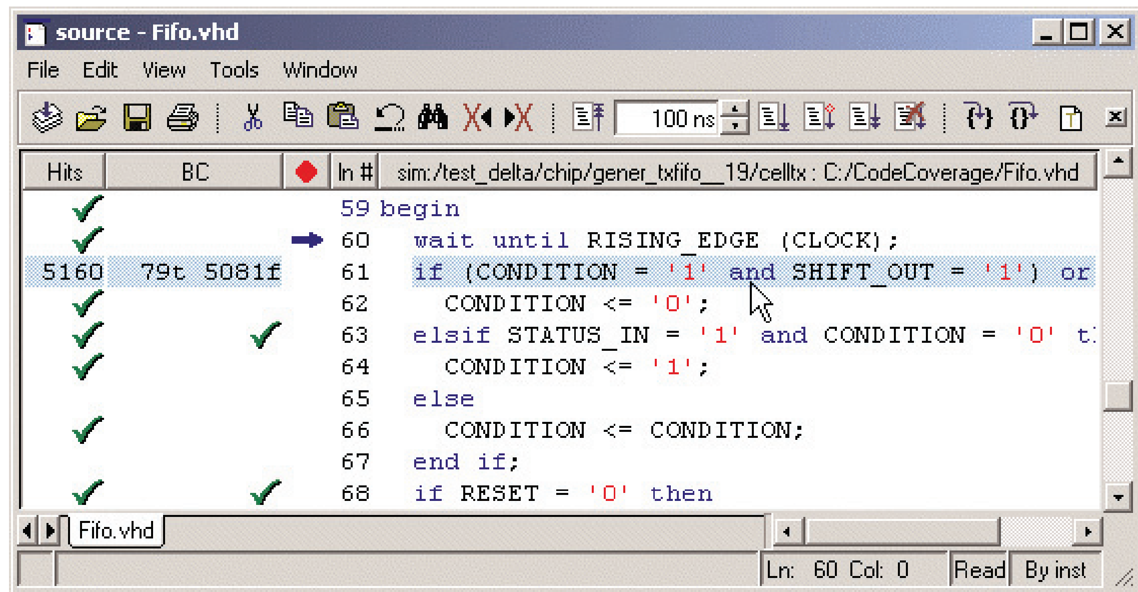


Figure 5. Coverage data in the Hits and BC columns of the Source window.

Exclusion File (Main window). To load the filter during a future analysis, select **File > Open > Exclusion File** (Main window).

Coverage Control

Code Coverage is configured when you compile your design and enabled when you load it. You specify the coverage metrics you want to collect with the *vcom* or *vlog* commands, or with the **Compile > Compile Options > Coverage** (Main window) selection in the GUI. Then you enable code coverage with the *vsim -coverage* command, or by selecting the **Enable code coverage** option in the Simulate dialog box. If you do not specify coverage metrics during compile, only statement coverage will be enabled when you invoke *vsim -coverage*. The other coverage metrics must be explicitly selected with the *-cover* switch with *vcom* or *vlog*. The correct syntax is:

vcom -cover <stat>

or

vlog -cover <stat>

where <stat> is one or more of the following characters:

b - branch statistics
c - condition statistics
e - expression statistics
t - toggle statistics
x - extended toggle statistics

Note that both ‘x’ and ‘t’ cannot be used at the same time. In addition, when you compile with coverage enabled, Model*Sim* disables certain optimizations, depending on which coverage metrics you choose (see Model*Sim* User’s Manual for details). This produces more accurate coverage statistics but may slow down your simulation.

New Coverage Utilities

Model*Sim* includes a coverage utility called *vcover* — a separate executable that can be run from a shell or from the *Modelsim>* prompt. The *vcover* utility includes:

- a **merge** capability that allows multiple coverage reports to be merged together without having to elaborate the design
- a **statistics** mode that prints the coverage summary for each file as well as the incremental numbers from the base file
- a **convert** command that allows instance names to be modified

Merge

The follow command is used to merge three coverage reports — *run1.dat, run2.dat, run3.dat* — into a single report, *merged.dat*, without requiring that the design be elaborated.

```
vcover merge -verbose merged.dat run1.dat
run2.dat run3.dat
```

The verbose output from the command is shown below. The base statistics from the first file to be merged, *run1.dat*, are followed by the incremental statistics for each preceding file, then the complete merge.

```
Establishing file run1.dat as the merge base

Coverage Report Totals BY INSTANCES:
Instances 101
Statements 4218, Hits 3350 Percent 79.4
Branches 2141, Hits 1834 Percent 85.7
Conditions 1150, Hits 976 Percent 84.9
Expressions 186, Hits 162 Percent 87.1
Toggle Nodes 1956 Toggled 1416 Percent 72.4

Merging file run2.dat
INCREMENTAL Coverage Report Totals BY
INSTANCES: Instances +0

Statements +0, Hits +152 Percent +3.604
Branches +0, Hits +31 Percent +1.448
Conditions +0, Hits +1 Percent +0.087
```

```
Expressions +0, Hits +8 Percent +4.301
Toggle Nodes +0 Toggled +11 Percent +0.55

Merging file run3.dat

INCREMENTAL Coverage Report Totals BY
INSTANCES: Instances +0
Statements +0, Hits +4 Percent +0.095
Branches +0, Hits +5 Percent +0.234
Conditions +0, Hits +1 Percent +0.087
Expressions +0, Hits +0 Percent +0.000
Toggle Nodes +0 Toggled +201 Percent +10.27

Writing merged result to merged.dat

Coverage Report Totals BY INSTANCES:
Instances 101
Statements 4218, Hits 3506 Percent 83.1
Branches 2141, Hits 1870 Percent 87.3
Conditions 1150, Hits 978 Percent 85.0
Expressions 186, Hits 170 Percent 91.4
Toggle Nodes 1956 Toggled 1628 Percent
83.6
```

Statistics

The following command is used to print the statistics for multiple coverage runs. This allows analysis of coverage results so that your testbench can be graded and optimised for the best results from the minimum number of runs. Results are shown below.

```
vcover stats -verbose run1.dat run2.dat
Printing stats for file run1.dat:
Coverage Report Totals BY INSTANCES:
Instances 101
Statements 4218, Hits 3350 Percent 79.4
Branches 2141, Hits 1834 Percent 85.7
Conditions 1150, Hits 976 Percent 84.9
Expressions 186, Hits 162 Percent 87.1
Toggle Nodes 1956 Toggled 1416 Percent 72.4
Printing stats for file run2.dat:
Coverage Report Totals BY INSTANCES:
Instances 101
Statements 4218, Hits 2515 Percent 59.6
Branches 2141, Hits 1086 Percent 50.7
Conditions 1150, Hits 204 Percent 17.7
Expressions 186, Hits 93 Percent 50.0
Toggle Nodes 1956 Toggled 478 Percent 24.4
```

Convert

The *convert* function is used to modify instance names within coverage reports. This allows the merge command to be used on reports from different simulation environments. For example, a chip might be simulated with two different testbenches named 'DataTest' and 'RandomTest.' The hierarchical names will be different only at the top level of each testbench. The *convert* command is used to modify the instance names in one report to match those in the second so they can be merged. If we want to merge the results into the 'DataTest' testbench, the following commands would be used:

```
vcover convert -verbose -strip 1 -install
/DataTest convert.dat RandomTest.dat
```

This command will strip the 1st level hierarchical names from the instance paths in the *RandomTest.dat* file and will insert */DataTest* into the front of these path names. The output of this command, the report *convert.dat*, can now be merged into the *DataTest.dat* results.

```
vcover merge -verbose merged.dat
convert.dat DataTest.dat
```

The *merged.dat* file can now be used to reload the '*DataTest*' test environment. These commands also make it possible to use the results from tests on low level blocks. The convert command simply matches the instance names to allow the merging process.

Conclusion

With Code Coverage integrated directly into the simulator kernel, Model*Sim* gives you minimum performance overhead while supplying the coverage metrics you need to analyse the effectiveness of your test suites. This direct integration eliminates the need for third party tools and allows comprehensive code coverage analysis to fit seamlessly into your design flow. The new graphic user interface allows you to quickly identify and fix areas of missed coverage areas. And the new *vcover* utility expands your coverage reporting abilities and allows you to compare the results from different testbenches.

For more information, call us or visit: www.model.com

Copyright © 2004 Mentor Graphics Corporation. This document contains information that is proprietary to Mentor Graphics Corporation and may be duplicated in whole or in part by the original recipient for internal business purposed only, provided that this entire notice appears in all copies. In accepting this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use of this information. Mentor Graphics is a registered trademark of Mentor Graphics Corporation. All other trademarks are the property of their respective owners.

Corporate Headquarters
Mentor Graphics Corporation
8005 S.W. Boeckman Road
Wilsonville, Oregon 97070USA
Phone: 503-685-7000
North American Support Center
Phone: 800-547-4303
Fax: 800-684-1795

Silicon Valley
Mentor Graphics Corporation
1001 Ridder Park Drive
San Jose, California 95131 USA
Phone: 408-436-1500
Fax: 408-436-1501

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics Taiwan
Room 1603, 16F,
International Trade Building
No. 333, Section 1, Keelung Road
Taipei, Taiwan, ROC
Phone: 886-2-27576020
Fax: 886-2-27576027

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Hills
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140
Japan
Phone: 81-3-5488-3030
Fax: 81-3-5488-3031

C&A 7-04 TECH6430-w