

# VHDL Code Coverage Examination of a Xilinx Design

Steven P. Micallef

TransEDA Ltd.  
Pilgrim's Close, Chandlers Ford,  
Eastleigh, Hampshire SO53 4ST,  
U.K.

## Abstract

This paper describes the analysis of VHDL test benches using TransEDA's code coverage tool, VHDLCover.

A short description of the implementation is also given. TransEDA converted the design from VHDL to Xilinx XNF format using TransGATE, TransEDA's synthesis tool, and implemented the design using a Xilinx 4010 FPGA. The VHDL source and VHDL test benches were supplied by the client.

## Section 1. Introduction

The design consisted of 4 components, each supplied with a separate test bench. Each block was analysed individually and the results are supplied in this report. The test benches were found to be of a good standard. However, some flaws were detected and these flaws together with suggestions on how to improve the test benches are detailed here.

Although there are detailed test benches for each of the blocks there was none for the complete system. This meant that no analysis could take place with the blocks interacting.

The analysis of the test benches was started by first simulating the blocks using VHDLCover to investigate the length of time each simulation should be run to gain the best results.

After analysing the results some experimental changes were made to the test benches and to one of the circuits and the simulations run again.

An in depth analysis was then carried out and those details are now reported here.

A feature of VHDLCover is the ability to trace one or more signals. Whenever a change in any of the selected signals is detected the current value of all signals are logged. To gain any useful information from this requires a detailed knowledge of the circuit and therefore this feature could not be made use of within the scope of this project.

A brief description of VHDLCover and the results it produces are described next.

## Section 2. VHDLCover

VHDLCover is a software tool used to determine the quality of VHDL test benches

and to identify redundant hardware. A test bench is the set of data used to test VHDL models by simulation. The test bench quality is determined by measuring several different code coverage criteria.

A test bench would ideally include all possible combinations of input values which is not practicable due to the resulting enormous number and length of simulation runs. A representative sample must be taken and some measure made as to the effectiveness of this sample. VHDLcover does this by considering the following coverage criteria:

#### Statement Coverage

This gives details of the VHDL statements which are executed during the run of the test bench. The number of times each statement is executed is counted. Statements which have not been executed are highlighted.

#### Branch and Condition Coverage

This criterion gives details of all VHDL branches which have been executed by the test bench, including conditional and selected signal assignment. Branches occur for 'IF' and 'CASE' clauses, and correspond to the possible outcomes of the conditional statements within the blocks. Condition coverage is carried out on any sub-conditions to see which combinations forced the statement to true.

#### Path Coverage

This checks a subset of all possible paths through a process known as Process Decision Path Pairs (PDPPs). These are paths through the VHDL made up of adjacent pairs of blocks of executable statements and possible branches. These are usually between sequential 'if' or 'case' clauses.

#### Triggering Coverage

This involves monitoring signals present in the sensitivity lists of PROCESS and WAIT statements and condition clauses in WAIT statements. Checks are made to see if the statements were triggered by a single change or by a combination of signals.

Before launching into the detailed analysis of the test benches, a brief description of how the implementation was carried out is given next.

### Section 3. Design Implementation

The implementation of the design followed the normal IC design route, i.e. simulate - synthesise - simulate - place&route - back annotate - simulate. The VHDL description was passed through TransGATE, a synthesis tool, and mapped to the Xilinx 4010 technology. Both VHDL netlist and XNF format files were produced. The XNF description was then passed through the XACT Xilinx place and route software and an XNF file with more detailed timing data produced.

At each stage of the above process the outputs from the various tools were simulated against the original VHDL description using VEDA's Vulcan simulator.

The main features of the implementation process have been summarised in Figure 3.1. below.

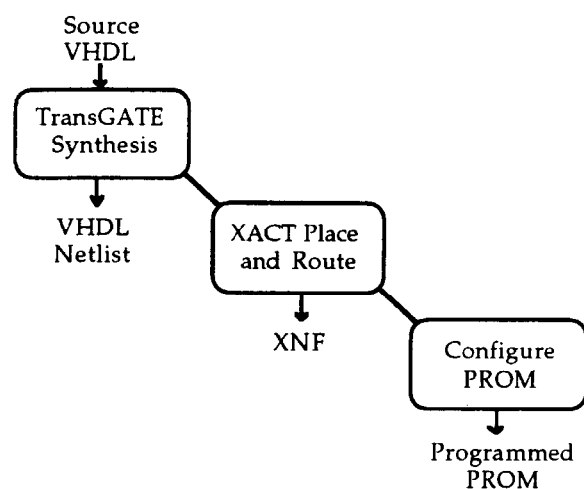


Figure 3.1. Main Implementation Flow.

Several CAD tools were used in the course of this project and their details are shown in Table 3.1. below.

Tool	Function	Ver.	Supplier
TransGATE	Synthesis	1.2.7	TransEDA
Vulcan	Simulation	2.1	VEDA
XACT	Place, Route & Timing	5.1.0	Xilinx
VHDL Cover	Code Coverage	2.0.1	TransEDA

Table 3.1. CAD Tools Used.

### 3.1. VHDL Source to VHDL Netlist and XNF Implementation

To convert the supplied behavioural VHDL description into a VHDL netlist and into a format from which a Xilinx FPGA can be produced, the VHDL must be passed through a synthesis tool. The tool used was TransGATE.

#### *TransGATE Logic Synthesis*

There are three stages to the synthesis process using TransGATE. After the circuit description has been modified, analysed and built, the Boolean equations which describe the circuit are transformed into a more efficient representation of the circuit.

The next stage is to map the Boolean equations into the chosen technology. At this point TransGATE allows trade-offs between area and timing to be specified.

The final stage in the TransGATE synthesis process is to optimise the circuit description for the chosen technology.

TransGATE can then write out the circuit description in various formats. The ones chosen for this project were VHDL netlist and XNF formats.

All four blocks were put through this process and VHDL netlists and XNF format descriptions for the individual blocks were created.

The implementation flow is summarised in Figure 3.2. below.

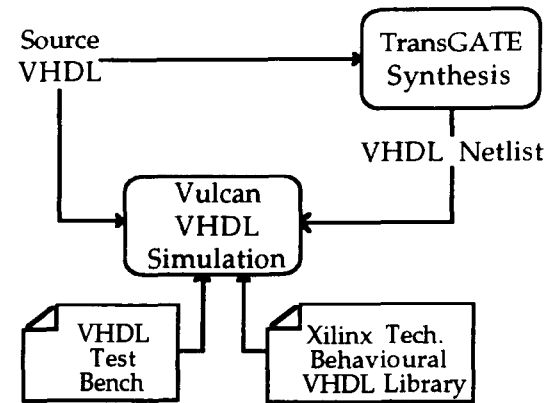


Figure 3.2. Source to VHDL Netlist and XNF.

After the blocks had been synthesised, the gate level VHDL netlist descriptions were then simulated with the behavioural description and the gate level functionality checked using the Vulcan Simulator.

### 3.2. VHDL Source to XNF Place and Route

One of the possible outputs from TransGATE, the XNF format file, can be read into the XACT place and route software. This is shown in Figure 3.3. below.

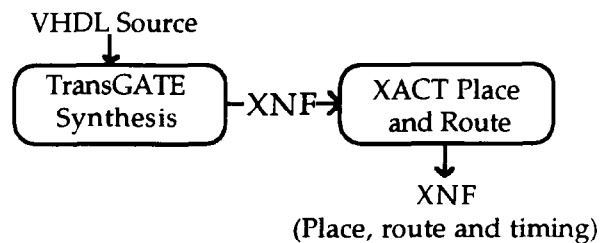


Figure 3.3. Source to XNF Place and Route.

#### *XACT Layout and Routing*

The XACT software allows design conversion, implementation and verification for Xilinx FPGAs.

After the XNF format file from TransGATE was read into XACT, the place and route options were activated. Some design verification and timing analysis was carried out and an XNF file with detailed timing and layout information was produced.

A file in the format for configuring the PROMs was also produced.

### 3.3. VHDL and VITAL Simulation

Simulation using Vulcan occurs at several places in the design process. After the blocks had been synthesised, the netlist block descriptions were simulated against the original VHDL to ensure that the design was still functioning correctly.

#### *Vulcan VITAL Simulation*

VITAL is an acronym for VHDL Initiative Toward ASIC Libraries. It is intended by the IEEE that all technology libraries will be described in VITAL, validated by the foundry for "sign-off" purposes and then used with any VHDL-1076 compliant simulator.

Using Vulcan's novel ability to simulate netlist descriptions using VITAL compliant third party libraries, the XNF output from the XACT tool is simulated to ensure that it matches the function of the original behavioural level VHDL description.

The use of Vulcan for the VITAL simulation in the design implementation is shown in Figure 3.4. below.

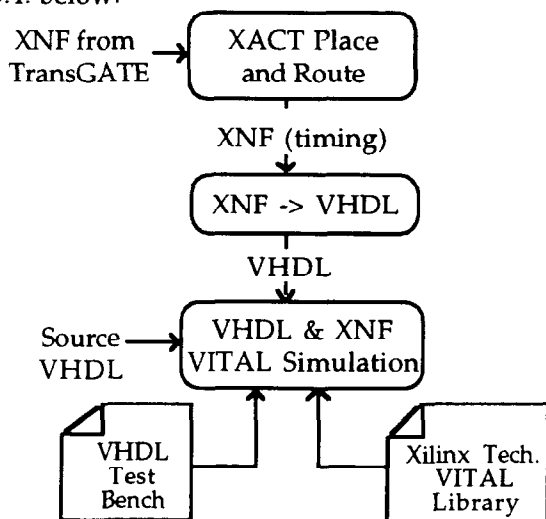


Figure 3.4. VHDL and VITAL Simulation.

The test bench used was the same as the one used at the RTL level. Thus in this case it is doubly important to know the effectiveness of the test bench.

The test benches for each of the blocks were analysed separately. This analysis is described in the next section.

## Section 4. Analyses of Test Benches

Each of the blocks are treated separately and the details of the analysis summarised at the end of each sub-section.

### 4.1. Analysis of Block 1 Test Bench

The complete testing of Block1 took a considerable length of time, because the test bench is of an iterative nature. However using VHDLcover it is possible to determine when further simulation produces no increase in the confidence of the quality of the test bench.

If the test bench is run for 10ms the following results are obtained:

Original:

Coverage information summary ...	
Number of statements executed :	78
Number of executable statements :	83
Statement coverage is :	94.0 %
Number of branches executed :	67
Number of branches :	72
Branch coverage is :	93.1 %
Number of triggering combinations executed :	59
Number of triggering combinations :	65
Triggering coverage is :	90.8 %
Number of paths executed :	5
Number of paths considered :	18
Path coverage is :	27.8 %
Signal trace coverage was not requested.	
Number of branch condition combinations executed :	22
Number of branch condition combinations :	48
Branch condition coverage is :	45.8 %

However some analysis was carried out using VHDLcover and the test bench was modified as suggested below. This improved the quality of the tests without increasing the length of simulation. This was done and the following results were obtained for a simulation run of 10ms:

Modified :

Number of statements executed	: 79
Number of executable statements	: 83
Statement coverage is	: 95.2 %
Number of branches executed	: 68
Number of branches	: 72
Branch coverage is	: 94.4 %
Number of triggering combinations executed	: 62
Number of triggering combinations	: 65
Triggering coverage is	: 95.4 %
Number of paths executed	: 5
Number of paths considered	: 18
Path coverage is	: 27.8 %
Signal trace coverage was not requested.	
Number of branch condition combinations executed	: 22
Number of branch condition combinations	: 48
Branch condition coverage is	: 45.8 %

If the block is run for 100ms then the following results are obtained:

Number of statements executed	: 83
Number of executable statements	: 83
Statement coverage is	: 100.0 %
Number of branches executed	: 72
Number of branches	: 72
Branch coverage is	: 100.0 %
Number of triggering combinations executed	: 65
Number of triggering combinations	: 65
Triggering coverage is	: 100.0 %
Number of paths executed	: 8
Number of paths considered	: 18
Path coverage is	: 44.4 %
Signal trace coverage was not requested.	
Number of branch condition combinations executed	: 43
Number of branch condition combinations	: 48
Branch condition coverage is	: 89.6 %

The testing of the execution of statements and that of branches in 'if' and 'case' clauses is 100%. The testing of triggering combinations, i.e. signals in process sensitivity lists changing, is also 100%. The testing of PDPPs, combinations of paths made up from the interaction between two 'if' clauses in a couple of the processes are the only weak areas.

Consideration of the test bench showed that the figures for 100ms could not be improved. However the figure for the 10ms simulation times could be improved and reducing the need to simulate the block for 100ms.

#### 4.1.1. PDPPs

VHDLcover identified two processes containing two interdependent 'if' clauses as causing the problems. See analysis at the end of paper, lines labelled 328 to 338. The test bench was not exercising all possible paths. In both sets of cases of 'if' clauses, the second 'if' clause drives its signal to the opposite value of the first 'if' clause and vice-versa.

Further, the two processes were dependent on changes in the other process.

#### 4.1.2. Suggestions for Improving the Test Bench

The assignment of the signals which trigger the two inter-dependent processes were more carefully assigned so that there was a difference in the time at which the signals which are driven within the processes are assigned. 'After' clauses were added together with varying transport delays for signals that are driving these processes.

#### 4.1.3. Summary

The fundamental nature of asynchronous circuits had not been fully explored and in particular the possibility of race conditions occurring had not been investigated.

To improve the testing of Block 1 'after' statements were added to the test bench and the circuit description.

#### 4.2. Analysis of Block 2 Test Bench

The testing of the execution of statements and branches in 'if' and 'case' clauses is 100%. The testing of triggering combinations, i.e. signals in process sensitivity lists changing, is good but there are a couple missing. The testing of PDPPs, combinations of paths made up from 'if' and 'case' clauses appears to be a problem.

A summary of the figures are given below:

```
Number of statements executed : 71
Number of executable statements : 71
Statement coverage is : 100.0 %

Number of branches executed : 58
Number of branches : 58
Branch coverage is : 100.0 %

Number of triggering combinations executed : 20
Number of triggering combinations : 23
Triggering coverage is : 87.0 %

Number of paths executed : 22
Number of paths considered: 242
Path coverage is : 9.1 %

Signal trace coverage was not requested.

Number of branch condition combinations
executed : 3
Number of branch condition combinations : 3
Branch condition coverage is : 100.0 %
```

##### 4.2.1. Triggering Combinations

The signals causing the problems for triggering combinations were three signals which were being assigned in a single process, via 'CASE' statements, even though they were independent of each other.

##### 4.2.2. PDPPs

The process which contained the three independent signals described in 4.2.1 was also causing the poor value for the PDPP figure.

##### 4.2.3. Suggestions for Improving the Circuit

The process described in Sections 4.2.1 and 4.2.2 would have been better written as three separate processes to produce a more modular design, possible of independent verification. This suggestion was followed and the following change in the VHDLcover figures were obtained:

```
...
Number of triggering combinations executed : 19
Number of triggering combinations : 19
Triggering coverage is : 100.0 %

No paths have been considered.

...
```

i.e. there are no longer any PDPP paths which exist and therefore none that require testing.

##### 4.2.4. Summary

Apart from the improvement achieved in the VHDLcover figures following the suggestion in sub-section 4.2.3 the designer will also have got a better insight into the way the parts of this block are interacting. As a by-product the synthesis tool will be more effective.

The coverage has been achieved by the minimum exercise of the VHDL code since many of the statements are only executed once. Therefore possible races, glitches, etc., occurring due to the order in which statements are executed have not been explored.

#### 4.3. Analysis of Block 3 Test Bench

The testing of the execution of statements and branches in 'if' and 'case' clauses is 100%. The testing of triggering combinations, i.e. signals in process sensitivity lists changing, is good but there is one missing. There is no problem in the testing of PDPPs.

VHDLcover summary information follows:

```

Number of statements executed : 60
Number of executable statements : 60
Statement coverage is : 100.0 %

Number of branches executed : 45
Number of branches : 45
Branch coverage is : 100.0 %

Number of triggering combinations executed : 60
Number of triggering combinations : 61
Triggering coverage is : 98.4 %

No paths have been considered.

Signal trace coverage was not requested.

Number of branch condition combinations
executed : 18
Number of branch condition combinations : 18
Branch condition coverage is : 100.0 %

```

#### 4.3.1. *Triggering Combinations*

There is a signal in the sensitivity list of a process highlighted by VHDLcover which the test bench does not change independently of the other signals in the sensitivity list. This can be observed in the test bench.

#### 4.3.2. *Suggestions to Improve the Test Bench*

The signal described above and the other signals which trigger the problem process should not be assigned at the same time. An 'after' clause should be added together with varying transport delays for these signals.

The following change was observed:

```

...
Number of triggering combinations executed : 61
Number of triggering combinations : 61
Triggering coverage is : 100.0 %
...

```

#### 4.3.3. *Summary*

This is an asynchronous system and driving signals at exactly the same time will hide the problems caused by this type of circuit.

#### 4.4. **Analysis of Block 4 Test Bench**

The simulation, according to the test bench, is scheduled to finish after approximately 1.5ms, for which it was run with these results:

```

Number of statements executed : 108
Number of executable statements : 130
Statement coverage is : 83.1 %

Number of branches executed : 90
Number of branches : 109
Branch coverage is : 82.6 %

Number of triggering combinations executed : 74
Number of triggering combinations : 81
Triggering coverage is : 91.4 %

Number of paths executed : 1
Number of paths considered: 4
Path coverage is : 25.0 %

Signal trace coverage was not requested.

Number of branch condition combinations
executed : 22
Number of branch condition combinations : 27
Branch condition coverage is : 81.5 %

```

Some experimentation was carried out using VHDLcover and it was found that after about 4ms of simulation the circuit has been tested as much as it is possible with the current test bench, except for some long counters.

The problems discussed here are with the simulation time set at 4ms, the VHDLcover results of which are as follows:

```

Number of statements executed : 119
Number of executable statements : 130
Statement coverage is : 91.5 %

Number of branches executed : 101
Number of branches : 109
Branch coverage is : 92.7 %

Number of triggering combinations executed : 78
Number of triggering combinations : 81
Triggering coverage is : 96.3 %

Number of paths executed : 1
Number of paths considered: 4
Path coverage is : 25.0 %

Signal trace coverage was not requested.

Number of branch condition combinations
executed : 22
Number of branch condition combinations : 27
Branch condition coverage is : 81.5 %

```

The testing of the execution of statements, branches, possible triggering combinations, and of PDPPs all pose problems.

#### 4.4.1. *Statement and Branch Execution*

There were 6 processes in which a total of 10 conditions were not tested by the supplied test bench.

#### 4.4.2. *Triggering Combinations*

There were 2 processes which each had a signal which were not driven separately, by the supplied test bench, from other signals in the sensitivity list.

#### 4.4.3. *PDPPs*

All the problems noted in 4.4.1 also show up as problems in this section.

#### 4.4.4. *Summary*

Some changes to the test bench were made to overcome the problems discussed in 4.4.1 - 4.4.3. However it was found that it is possible to put this circuit into oscillation by changing two signals at the same time, first to '0' and then to '1' some time later.

An improvement to the VHDLCover figure was achieved using the new test bench as shown below:

Number of statements executed	: 121
Number of executable statements	: 130
Statement coverage is	: 93.1 %
Number of branches executed	: 103
Number of branches	: 109
Branch coverage is	: 94.5 %
Number of triggering combinations executed	: 80
Number of triggering combinations	: 81
Triggering coverage is	: 98.8 %
Number of paths executed	: 4
Number of paths considered	: 4
Path coverage is	: 100.0 %
Signal trace coverage was not requested.	
Number of branch condition combinations executed	: 22
Number of branch condition combinations	: 27
Branch condition coverage is	: 81.5 %

Analysis of the output from VHDLCover also shows that the conditions causing the values for statement and branch coverage can be accounted for.

Assignment of signals at the same time for asynchronous processes allows problems to be hidden and so 'after' clauses should be used.

## 5. Conclusions

Using VHDLCover allows a qualitative and objective analysis of VHDL test benches, without which there is no way of knowing how good the test benches are.

The test benches in general were of a good quality. However the function of asynchronous circuits or circuits with multiple clocks function has not been explored. Changing the value of signals in the test benches in the same simulation 'delta' could be hiding any possible race conditions, glitches or oscillations. Block 4 could be put into some sort of oscillation, cured by using 'after' clauses.

Block 2 has been tested in a superficial manner with many statements only being visited once.

The code coverage tool, VHDLCover, has been developed and is marketed by TransEDA. VHDLCover is distributed by VEDA Design Automation in Europe & USA. TransLogic in Holland & Belgium. SoftSmiths in Australia.

## 6. Acknowledgements

TransEDA Ltd. would like to acknowledge the support of DRA Malvern, Xilinx Corp. and VEDA Design Automation Ltd. in the implementation of this project.

## 7. References

TransEDA Ltd. "VHDLCover User Guide", Version 2.0.1, July 1995.

## Appendix - VHDL Cover Sample File

```

*****
VHDL Cover : the VHDL code coverage measurement tool
Copyright 1994,1995 TransEDA Limited
Version number : 2.0.1
*****
This file contains coverage details for: block_1.vhd
-----Statements-----
Statement coverage information ...
Count      Line #      Text
...
          25          entity BLOCK_1 is
...
          77          end BLOCK_1;
...
          82          architecture BEHAVIORAL of BLOCK_1 is
...
          128         begin
...
          325         COUNTER_F_5: process(SETF1L, CNT_F_NET2, RSET10, CNT_F_NET3)
11          326         begin
          328             if (SETF1L = '1' or CNT_F_NET2 = '0') then
****0****          329                 BUF_FON <= '1';
          330             elsif (RSET10 = '1' or CNT_F_NET3 = '0') then
10          331                 BUF_FON <= '0';
          332             end if;
          334             if (RSET10 = '1' or CNT_F_NET3 = '0') then
10          335                 BUF_F1N <= '1';
          336             elsif (SETF1L = '1' or CNT_F_NET2 = '0') then
****0****          337                 BUF_F1N <= '0';
          338             end if;
          340         end process;
...
          448         end BEHAVIORAL;
-----Branches-----
Branch coverage information ...
Count      Line #      Branch text
...
10          334             if (RSET10 = '1' or CNT_F_NET3 = '0') then
****0****          336             elsif (SETF1L = '1' or CNT_F_NET2 = '0') then
1
          All false condition
Total number of times construct entered : 11
Condition coverage details:
Decision:      if (RSET10 = '1' or CNT_F_NET3 = '0') then
Count          Conditions
3              : only RSET10 = '1' was true
2              : only CNT_F_NET3 = '0' was true
5              : more than one condition was true
Decision:      elsif (SETF1L = '1' or CNT_F_NET2 = '0') then
Count          Conditions
****0****     : only SETF1L = '1' was true
****0****     : only CNT_F_NET2 = '0' was true
****0****     : more than one condition was true

```

-----Sensitivity Lists-----  
 Sensitivity list triggering coverage information ...

...  
 -----  
 Line #      Statement  
 325            COUNTER\_F\_5: process(SETF1L, CNT\_F\_NET2, RSET10, CNT\_F\_NET3)  
 Count        Signal transitions  
 1            : only SETF1L changed  
 \*\*\*\*0\*\*\*\*   : only CNT\_F\_NET2 changed  
 4            : only RSET10 changed  
 4            : only CNT\_F\_NET3 changed  
 2            : more than one signal changed  
 -----  
 ...

-----Paths-----

Path coverage information ...

Count	Line #	Text
0		if (SETF1L = '1' or CNT_F_NET2 = '0') then if (RSET10 = '1' or CNT_F_NET3 = '0') then
0		if (SETF1L = '1' or CNT_F_NET2 = '0') then elseif (SETF1L = '1' or CNT_F_NET2 = '0') then
0		if (SETF1L = '1' or CNT_F_NET2 = '0') then "All false condition" for if (RSET10 = '1' or CNT_F_NET3 = '0') then
10		elseif (RSET10 = '1' or CNT_F_NET3 = '0') then if (RSET10 = '1' or CNT_F_NET3 = '0') then
0		elseif (RSET10 = '1' or CNT_F_NET3 = '0') then elseif (SETF1L = '1' or CNT_F_NET2 = '0') then
0		elseif (RSET10 = '1' or CNT_F_NET3 = '0') then "All false condition" for if (RSET10 = '1' or CNT_F_NET3 = '0') then
0		"All false condition" for if (SETF1L = '1' or CNT_F_NET2 = '0') then if (RSET10 = '1' or CNT_F_NET3 = '0') then
0		"All false condition" for if (SETF1L = '1' or CNT_F_NET2 = '0') then elseif (SETF1L = '1' or CNT_F_NET2 = '0') then
1		"All false condition" for if (SETF1L = '1' or CNT_F_NET2 = '0') then "All false condition" for if (RSET10 = '1' or CNT_F_NET3 = '0') then

-----Signal Traces-----

Signal trace coverage information ...

No signal traces have been requested

-----Summary-----

Coverage information summary ...

Summary information can be found labelled "Modified" in Section 4.1.