



Simulation and Verification with ModelSim/Questasim Code Coverage and Profiling



Universidad Nacional del Centro
de la Provincia de Buenos Aires

Elías Todorovich

etodorov@exa.unicen.edu.ar

June 2010

2

Introduction

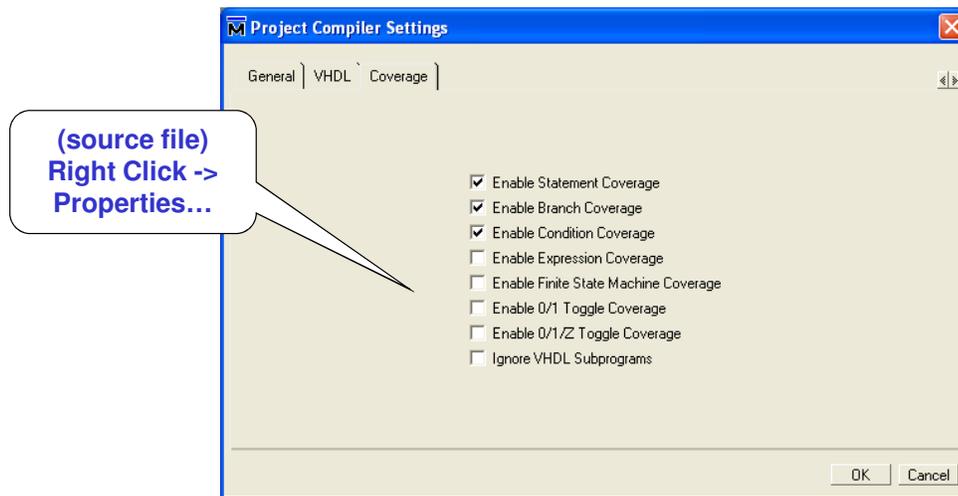
- ModelSim **Code Coverage** gives you graphical and report file feedback on which
 - executable statements,
 - branches,
 - conditions, and
 - expressions in your source code have been executed.
- It also measures bits of logic that have been toggled during execution.
- Enabling Code Coverage is a two step process.
 1. Identify which coverage statistics you want and compile the design files.
 2. Load the design and tell ModelSim to produce those statistics.

Compile for Code Coverage

- A) Command **vcom**. For VHDL type
vcom -cover bcsxf alu.vhd regs.vhd pc.vhd
- The **-cover** argument instructs ModelSim to collect specific coverage statistics:
 - **b** branch,
 - **c** condition,
 - **e** expression,
 - **s** statement,
 - **t** toggle,
 - **x** extended toggle,
 - **f** and finite state machine
- Refer to the section Enabling Code Coverage in the User's Manual for more information.

Compile for Code Coverage

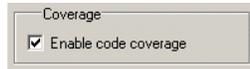
- B) GUI



Load a design for Code Coverage

- A) Command vsim:
vsim -voptargs="+acc" -coverage test_sm

- B) GUI



- ModelSim adds several columns to the Files and sim tabs in the Workspace:

Filename	Fullpath	Type	Stmt Count	Stmt Hits	Stmt %	Stmt Graph	Branch Cou
sim	vsim.wlf						
sm.v	sm.v	verilog	22	19	86.364		
sm_seq.v	sm_seq.v	verilog	16	15	93.750		
beh_sram.v	beh_sram.v	verilog	6	5	83.333		
test_sm.v	test_sm.v	verilog	77	70	90.909		

Run a design for Code Coverage

- By default, ModelSim also displays three Code Coverage panes in the Main window:

Statement	Branch	Condition	Expression	Toggle	FSM
31			#5		
31			into = {4'b0001,28'b0};		
32			@ (posedge clk)		
33			#5		
33			into = data;		
134			#100		

Instance: /test_sm
Signal: into
Node count: 32

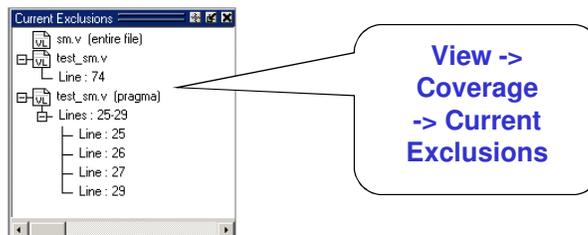
->0: 71870
->1: 71876

Toggle Coverage: 34.38%
0/1 Coverage: 34.38%
Full Coverage: 34.38%
Z Coverage: 34.38%

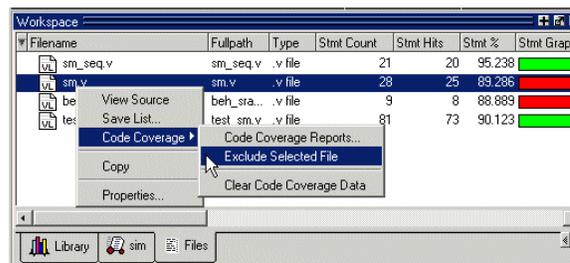
Stmt %	Stmt graph	Branch count	Branch hits	Branch misses	Branch %	Branch graph	Con
1	90%	8	7	1	87.5%		
3	90%	20	17	3	85%		
1	95.5%	14	13	1	92.9%		
13	84.3%						

Run a design for Code Coverage

- For advanced users, you can list all files and lines that are excluded from coverage statistics.

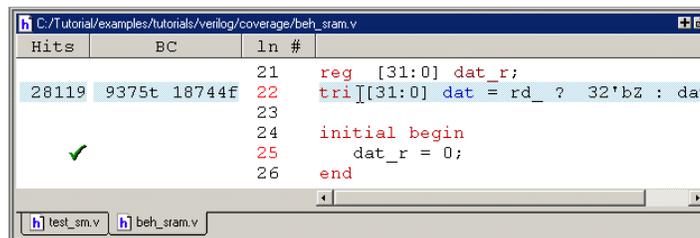


- You can exclude lines and files (You can also exclude the selection for the current instance only)



More Analysis...

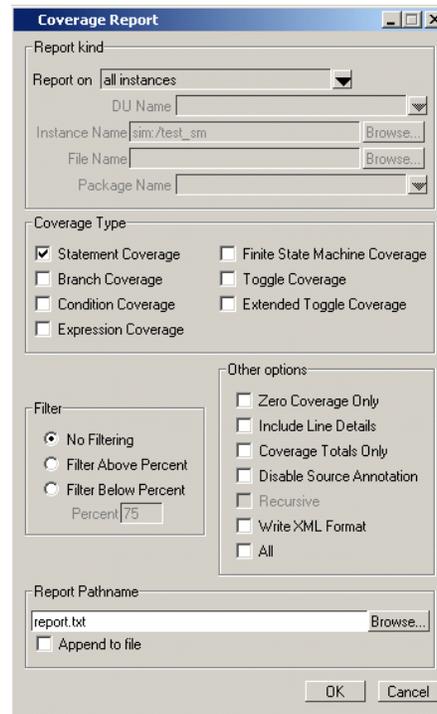
- Coverage Statistics in the Source Window
 - If you hover the mouse pointer over a line of executable code with a green checkmark in the Hits or BC columns, the icons change to numbers that indicate how many times the statements and branches in that line were executed.



- Toggle Statistics in the Objects Pane

Name	Value	Kind	Mode	1H>0L	CL>1H	0L>Z	Z>0L	1H>Z	Z>1H	#Nodes	#Togglecl	% Togglecl	% 01
ino	0000...	Packed Array	Internal	71870	71876	0	0	0	0	32	11	34.38%	34.38%
outof	0000...	Packed Array	Internal	12493	12493	0	0	0	0	32	6	18.75%	21.88%
rst	0	Packed Array	Internal	2	1	0	0	0	0	1	1	100%	100%
clk	0	Packed Array	Internal	50000	50000	0	0	0	0	1	1	100%	100%
out_wiie	0000...	Net	Internal	12493	12493	0	0	0	0	32	6	18.75%	21.88%
dat	0000...	Net	Internal	14055	17186	378058	381216	71862	63736	32	6	18.75%	21.88%

Creating Code Coverage Reports



Tools -> Code Coverage -> Reports

Modelsim Profiler

- The Profiler identifies
 - the percentage of simulation time spent in each section of your code and
 - the amount of memory allocated to each function and instance.
- With this information, you can identify bottlenecks and reduce simulation time by optimizing your code.
- For example, the statistical sampling profiler might show the following:
 - objects in the sensitivity list that are not required, resulting in a process that consumes more simulation time than necessary,
 - a testbench process that is active even though it is not needed, etc.
- With this information, you can make changes to the source code that will speed up the simulation.

Modelsim Profiler

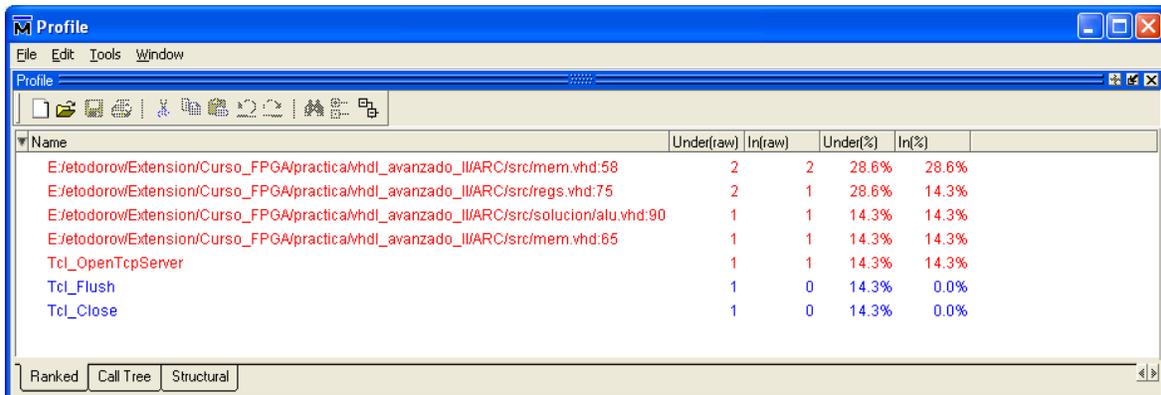
- The profiler's statistical sampling profiler samples the current simulation at a user-determined rate
 - every <n> milliseconds of real or "wall-clock" time, not simulation time
 - and records what is executing at each sample point.
- The `profile interval` command selects the frequency with which the profiler collects samples during a run command.
 - Example: `profile interval 2`
 - An integer value from 1 to 999 represents how many milliseconds to wait between each sample collected during a profiled simulation run.
 - Default is 10 ms.
- An entire simulation need not be run to get good information about what parts of your design are using the most simulation time.
 - A few thousand samples, for example, can be accumulated before pausing the simulation to see where simulation time is being spent.

Enabling the Performance Profiler

- After loading a design use 
- Run the simulation and observe the status bar:

Now: 100 ps Delta: 0 Profile Samples: 18 Memory: 65.4KB

- You can see the results: View -> Profiling -> Profile



The screenshot shows the Modelsim Profiler window with a table of results. The table has columns for Name, Under(raw), In(raw), Under(%), and In(%). The results are as follows:

Name	Under(raw)	In(raw)	Under(%)	In(%)
E:/etodorow/Extension/Curso_FPGA/practica/vhdl_avanzado_IIVARC/src/mem.vhd:58	2	2	28.6%	28.6%
E:/etodorow/Extension/Curso_FPGA/practica/vhdl_avanzado_IIVARC/src/regs.vhd:75	2	1	28.6%	14.3%
E:/etodorow/Extension/Curso_FPGA/practica/vhdl_avanzado_IIVARC/src/solucion/alu.vhd:90	1	1	14.3%	14.3%
E:/etodorow/Extension/Curso_FPGA/practica/vhdl_avanzado_IIVARC/src/mem.vhd:65	1	1	14.3%	14.3%
Tcl_OpenTcpServer	1	1	14.3%	14.3%
Tcl_Flush	1	0	14.3%	0.0%
Tcl_Close	1	0	14.3%	0.0%

At the bottom of the window, there are tabs for 'Ranked', 'Call Tree', and 'Structural'.

Viewing Profiler Results

Column	Description
Under(raw)	the raw number of Profiler samples collected during the execution of a function, including all support routines under that function; or, the number of samples collected for an instance, including all instances beneath it in the structural hierarchy
In(raw)	the raw number of Profiler samples collected during a function or instance
Under(%)	the ratio (as a percentage) of the samples collected during the execution of a function and all support routines under that function to the total number of samples collected; or, the ratio of the samples collected during an instance, including all instances beneath it in the structural hierarchy, to the total number of samples collected
In(%)	the ratio (as a percentage) of the total samples collected during a function or instance
%Parent (not in the Ranked view)	the ratio (as a percentage) of the samples collected during the execution of a function or instance to the samples collected in the parent function or instance



Simulation and Verification with ModelSim/Questasim Code Coverage and Profiling



Universidad Nacional del Centro
de la Provincia de Buenos Aires

Elías Todorovich

etodorov@exa.unicen.edu.ar

June 2010