Synopsys VCS
Jumpstart Training

Simon Liu
Application Consultant
simonliu@synopsys.com
Agenda

❖ Overview

❖ VCS Coverage Metrics (VCM)
  ❖ Basic Commands
  ❖ Coverage Metrics, DirectC and NIV
  ❖ PLI Usage
  ❖ SDF Backannotation
  ❖ Graphical Debugging
  ❖ Advanced Features

❖ Summary
Verification Challenge

- Design: 25%
- Verification: 75%

Source: Collett International, Synopsys, Dataquest

1995: 1M Gates
2001: 10M Gates
2007: 100M Gates

Simulation Vectors

©2001 Synopsys, Inc. (3) CONFIDENTIAL
Synopsys Verification Strategy

- Faster
  - Unix & Linux farms
  - H/W Assisted
- Integrated
  - Testbench Automation
  - Coverage Analysis
- Smarter
  - Formal Methods
  - Static Timing Analysis

Simulation Vectors:
- 1995
- 1M Gates
- 10B
- 100M
- 100T

Simulation Performance:
- 1M
- 10M
- 100M

©2001 Synopsys, Inc. (4) CONFIDENTIAL
Synopsys Complete SoC Verification Solution

Testbench and Quality Measurement

HDL Simulation

VCS™
MX
Scirocco™

Models

DesignWare®

Synopsys Professional Services
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
  - Advanced Features
- Summary
VCS – Verilog Compiled Simulator

- Event-driven, with cycle-based technology
- Compile once, run many times
- Provides incremental compilation for fast design turnaround time
- Offers a powerful, easy-to-use debugging environment (batch mode/GUI)
- Profile design information
  - Searching for bottleneck of simulation
- One simulator for all phases of design verification
  - behavioral, RTL, debug, gate, full timing
  - SDF, sign-off
- Support UNIX, NT, Linux and crossplatform
Compiled vs. Interpreted Simulation

- **Compiled** (e.g. : VCS, NC-Verilog)
  - Fast (1x to 20x faster)
  - Memory efficient (2x to 10x better)
  - Incremental compile for fast turnaround
  - The first compile takes a few time
  - Must recompile for more debug information

- **Interpreted** (e.g. : Verilog-XL)
  - Fast startup
  - Full debug visibility at all times
  - Large install base for interpreted simulation methodology
  - Slow!
  - The debug capability makes it a memory hog
  - Must re-parse all the Verilog every time you run a simulation
VCS – One Simulator for All

- Design
- Explore
- RTL
- Regression
- Debug
- Gate-level
- Regression
- Signoff
- Telcom/
- Networking
- Periph.
- Co-
- processor
- Full-
- Custom
- Processor
- Graphics

**VCS!**

- Interpreted simulator
- Other NC
- Cycle Sim
Complete Debugging Environment

- Interactive, text-based and post-processing
- Backward or forward execution in time
- Drag & drop and synchronization of data between windows
  - Hierarchy Browser
  - Waveform Viewer
  - Logic Browser
  - Source Level Debugger
  - Register Window
  - Interactive Control
High-performance, flexible, low-risk regression solution

- VCS on Linux is rock solid
  - Fastest growing platform
- VCS treats Linux as Unix
  - Same release schedule, support model & licensing
- New PCs are very fast (1.5 GigaHertz!)
  - Great for simulation server farms
- Full verification flow supported!
  - VCM / VERA / MemPro / Scirocco
- All other SNPS tools soon!
VCS - Continuing to Deliver on Value

simulation performance

Compiled Code


VCS 1.X

VCS 2.X

VCS 3.X

VCS 4.X

VCS 5.X

VCS 6.0

Cycle & 2-state, Compiled SDF, ASIC Sign-off

Radiant Optimizations, RoadRunner, Cycle Optimizations, DKL, Profile debug

Gate-level Timing Improvement, Static race Detection, Direct C/ C++ interface

All Free Upgrade!
VCS - Technical Innovation Continues

• VCS 6.0.1 - 2001.Q3
  ▪ Coverage Analysis within VCS (2X faster)
  ▪ DirectC Interface for C/C++
  ▪ Integration with VERA

• VCS 6.1 - 2002.Q1
  ▪ Global optimizations and more
  ▪ Verilog 2001 - like Design Compiler
  ▪ Initial PLI 2.0 (VPI) support
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
  - Advanced Features
- Summary
VCS Directory Structure

$VCS_HOME

- **bin**
  - (vcs shell script)
  - (vcs shell script)

- **doc**
  - users guide (PDF)
  - man pages
  - release notes

- **<arch>**
  - (for each OS)

- **ovi**
  - (OVI LRM)

- **virsimdir**
  - VirSim GUI files
  - (help, resource files, .tab)

- **flexlm**
  - license server files
  - drivers (NT)

- **bin**
  - vcs1, cmView1
  - VirSim GUI
  - MS C-compiler(NT)

- **util**
  - (utilities)
  - vcat, vcdiff
  - vcd2vpd, vpd2vcd
Setting Up VCS

• In Synopsys Common License (SCL)
  ▪ set SNPSLMD_LICENSE_FILE <path_that_license_file_resides>
  ▪ Include VCS features in license file

• In user’s environment setup file
  ▪ setenv VCS_HOME <VCS_installed_path>
  ▪ set path = ($VCS_HOME/bin $path)

• Use your site’s script
VCS Working Directory Tree

./work

/csrc
- Makefile
- Descriptor Files
- Intermediate Files
- Incremental & Shared

/simv.daidir
- source files
- command files
- simv ; Executable
- logfiles
- other

- Files for PLI/ACC
- Files for Interactive (+cli)
- Files that have structure described

!! Important : If you move or rename the executable, you need to move or rename the /simv.daidir directory
How VCS Works

• **Compile**
  - Converts Verilog source into an executable
  - `> vcs design.v [other_switches]`
    - The VCS compiler parses the source to look for syntax errors
    - It then re-parses to resolve references and build hierarchy.
    - Checks switches used for debug visibility
    - Flattens parts of the design, eliminate redundancies, optimize.
    - Generates code (objects, assembly, or c files)
    - Link object code, run-time routines, and any user PLI to make `simv` executable

• **Simulation**
  - Run `simv` to perform simulation
Incremental Compile

• Should be used for all facets of simulation
• Faster turnaround time for debugging
• Turn on via any -M switch
  • -Mupdate -Mdir=csrc_debug
  • -Mmakeprogram=pmake
• Creates subdirectory called csrc/ (by default)
• Stores object files, incremental compile checksums and a Makefile to build and link
• Only recompiles/regenerates code for modules that have functional changes or different debug capabilities
• Local vs. network mounted code generation area
Incremental Compile

- Enables faster subsequent compilation by compiling only the modules you have changed.
  - If you make a change to module test
    - VCS compiles only module test
    - Verilog-XL interprets the entire source description over again

Compile using `-Mupdate` option
Methodology for Compiled Simulation

• Compile Once, Run Many Times
  ▪ Verilog source contains design and some testbench scaffolding
  ▪ PLI is used to apply stimulus and monitor results
  ▪ Use $readmem(b|h) to apply stimulus from a file

• Used by most high-end companies

• Avoid interpreted simulation methodologies:
  ▪ applying stimulus interactively (command line scripts)
  ▪ using multiple testbenches which causes recompilation

• Create Configurable Testbenches
  ▪ $test$plusargs()
  ▪ $readmemb/$readmemh
A Typical VCS Run

```
vcs des.v -f file.f -y lib/dir +libext+.v pli.o -P pli.tab -Mupdate -l log.run
```

des.v  Verilog source code

-f file.f  A file that contains a list of other Verilog files or VCS switches

-y lib/dir  Library directory

+libext+v  Look at all files in the lib dir that end in .v

pli.o  Precompiled PLI source code to be linked in

-P pli.tab  Dispatch table between PLI calls ($myfunc) and C code. Also turns on PLI access & visibility

-Mupdate  Use incremental compile and make sure to update the Makefile

-l log.run  Generate log file called log.run
Other Common Switches

- **-I** Compile for interactive simulation
- **+cli** Turn on command line interface for debug
- **+cli+mod=3** Isolate cli to just module mod
- **-line** Allow line stepping capabilities in the debugger
  *(Not necessary if you want to just view the source code)*
- **-RI** Link in VIRSIM GUI and run executable (simv) in interactive mode immediately after compile
- **-RIG** Skips compile and interactively runs simulation
- **-PP** Compile for fast Post Processing mode
- **-RPP** Run post processing mode - no compile or simulation
Other Common Switches

- **+vcsd** Enables a direct link for the dumping VCD+ files for faster recording. This option is not supported when you run VirSim interactively with the -RI or -RIG option.

- **-o name** Rename the simv executable to *name*

- **+define+mac** Define macro *mac*

- **-v lib/file.v** Look in file.v to resolve references

- **+acc** Globally turn on PLI access visibility on the whole design. *Never use this switch as performance will suffer. Use -P instead*

- **+timopt+<clock_period>** Enable timing optimization during gate-level simulation.
Switches That Provide Information

vcs -ID  Get host machine license information (submit this info when you enter a support call)

vcs -platform  Echo name of platform, e.g: sun_sparc_sollaris_5.4, can be use in scripts (`vcs –platform`) (submit this info when you enter a support call)

vcs -h  Show common command options (help on switches)

vcs a.v ... -Vt  Compile verbosely, prints the commands vcs executes

vcs a.v ...-notice  Give more information for warnings etc

vcs -f filelist -Xman=4  Create single output source file tokens.v (testcases)

vcs ... -l logfile_name  Creates logfile of compilation, good to send to support when entering a call.
Switches for fast simulation

• Compile switches for speed
  
  vcs -Mupdate [+2state] [+rad] [+nospecify] [+notimingchecks] [+nbaopt]

  ▪ The +2state switch is for designs that can take advantage of 2 state simulation (at least 2X speedup!)
  ▪ The +rad switch turns on radiant optimizations (not recommended for debug, n/a for simulations w/ timing, sdf, specify)
  ▪ The +notimingcheck and +nospecify options take all the timing information away and essentially makes it a functional simulation
  ▪ The +nbaopt ignores any intra-delay specifications, RHS # delays
  ▪ Also do not use any +define options which turn on any dumpvars internally to improve speed

• Run time switches for speed

  ▪ simv +vcs+dumpoff +nospecify [+notimingchecks] +vcs+nostdout
Testbench Methodology

- Consider using runtime conditional blocks to create configurable testbenches.
- Write blocks of tests into the verilog testbench surrounded by the \texttt{$test$plusargs} construct.

This allows the use of unique runtime switches to control what gets simulated. --- Runtime command: \texttt{simv +test2}

Note: To pass runtime arguments during compile time with -R, add \texttt{+plusargs \textit{argument}} at end of compile command line.

```verilog
module testbench;
initial begin
    if ($test$plusargs("test1"))
        [source for test one]
    else if ($test$plusargs("test2"))
        [source for test two]
    ...
```
Watch Out For Large Stimulus Blocks

- Large stimulus blocks
  - initial blocks with thousands of procedural assigns are NOT efficient in a compiled simulators
  - use pattern based stimulus e.g.: $readmemb, $readmemh or custom PLI task
    example:
    ```vhdl
    module test;
    reg [15:0]  data [0:1023]
    initial $readmemb("vector.dat",data);
    endmodule
    ```
  - vectors.dat can be linked to a test file that contains stimulus. for each run just change the link to the new test file
  - Can use free PLI to specify the name of the file on the command line
Debug - Performance Tradeoff

- By default, VCS gives minimum visibility and maximum performance.
- Adding CLI or PLI calls causes VCS to compile in hooks onto all signals so the debug tool or PLI routine can read, write, or trace it.
- These calls add overhead to the simulator because you are stating that all signals are important and can’t be optimized out.
- From VCS4.1 on, VCS enables some optimizations.
- Minimize the modules with visibility!
Options That Affect Performance

• Interactive and debug simulations require VCS compile switch specifications.

• Interactive simulations increase simulation overhead and require more resources - therefore limiting performance. Such common switches:

  +acc For acc specification
  +cli+n Command line Interface/Debug
  -I (capital "eye") Interactive control
  -PP Post-process (waveform dumping)
  -line Event and Line Stepping/Tracing
  (not necessary if just want to view source)
  +nopliopt Prevents in-lining of modules w/PLI
  -P filename Adding PLI calls/capability
  +prof Profiling
  +race Race detection
Command Line Interactive Debugging

- VCS supports an interactive non-graphical textual debugger
  - different commands from Verilog-XL
  - set breakpoints
  - examine the values of register and wires
  - change register values
  - debugger is enabled using the cli switch
    > vcs design.v +cli
- To enter interactive mode either
  - use the -s flag at run time, e.g. simv -s - this will halt the simulation at time zero and enter interactive mode
  - ^c will halt the simulation while it is running
  - use $stop in the Verilog source
  - in interactive mode VCS prompts you for interactive commands
Interactive Command Switches

- The +cli switch takes an optional switch that specifies the level of debug
  - Rule of thumb, +cli+2 increases runtime by 10%, +cli+3 by 50%

- Using compile time option, user has control of debug capability versus runtime performance trade off, on a module/module basis
  - Can choose fast batch runs, or slower interactive runs

+cli+1 or cli+  Enable reads of nets and registers and writing of registers
+cli+2  Enable callbacks (e.g., break @val)
+cli+3  Enable force and release of nets (e.g., force var = 0, release), not including registers
+cli+4  Enable force and release of nets and registers
+cli -line  Enable line, next and trace commands
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
  - Advanced Features
- Summary
Code Coverage Analysis

- Develop Test Cases
- Simulate
- Debug
- CoverMeter
- Instrument
- Monitor
- Analyze
- Done?

Transitions | States | Reachability
---|---|---

<table>
<thead>
<tr>
<th>from/to</th>
<th>idle</th>
<th>five</th>
<th>ten</th>
<th>fifth</th>
<th>twenty</th>
<th>paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>100.00</td>
<td>100.00</td>
<td>33.33</td>
<td>20.00</td>
<td>41.67</td>
<td></td>
</tr>
<tr>
<td>five</td>
<td>21.43</td>
<td>14.29</td>
<td>0.00</td>
<td>11.11</td>
<td>14.29</td>
<td></td>
</tr>
<tr>
<td>ten</td>
<td>37.50</td>
<td>12.50</td>
<td>20.00</td>
<td>25.00</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>fifth</td>
<td>25.00</td>
<td>25.00</td>
<td>0.00</td>
<td>0.00</td>
<td>5.33</td>
<td></td>
</tr>
<tr>
<td>twenty</td>
<td>50.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>12.50</td>
<td></td>
</tr>
<tr>
<td>paid</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>0.00</td>
<td>20.00</td>
<td></td>
</tr>
</tbody>
</table>

List of sequences: idle -> fifth
VCS Coverage Metrics (VCM)

- Built-in code-coverage analysis

- FSM, Condition, Line, Toggle coverage
  - +FSM, +COND, +LINE, +TGL
  - +ALL, +DFLT

- Procedure
  - `vcs +CM+<types_of_coverage_metrics>`
  - `simv +CM+<types_of_coverage_metrics>`
  - `cmView +CM+<types_of_coverage_metrics>`, or
  - `cmView –b +CM+<types_of_coverage_metrics>`

Develop Test Cases

Compile Simulate Debug
Analyze Coverage

VCS Verilog Simulator

Done? No
VCS - DirectC

- DirectC is an additional interface to VCS
  - cfunctions
  - cmodules
- It is meant to ease C/C++ models integration in a VCS-based verification environment
- Easy to use, easier to maintain environment
  - No PLI knowledge needed
- Debug the testbench (VERA/VCS) earlier
  - C/C++ behavioral reference models are available before Verilog RTL
  - C/C++ behavioral reference models run faster than Verilog RTL

vcs -R +vc str.v str.c
/* File name: str.c       */
#include <stdio.h>
char *get_string()
{
    return("a sample string");
}
void print_string(char *s)
{
    if (s) {
        printf("string '%s'
", s);
    } else {
        printf("<<NULL>>\n");
    }
    return;
}

/* File name: str.v       */
extern string get_string();
extern void print_string(string);
module top;
reg [31:0] r32;  //integer r32;
reg [63:0] r60;  //integer r60;
initial begin
    r32 = get_string();
    print_string(r32);
    r60 = get_string();
    print_string(r60);
end
endmodule

† Introducing new keyword extern
† Call your C function from anywhere in Verilog
Full-Chip Verification with NanoSim

- SPICE Netlist
- Verilog-A Models
- ADFMI C Models
- Verilog-D Netlist

NanoSim → Waveform Viewer → Simulation output

VCS → Simulation output
Partition file in NIV

- Sample vcsAD.init file:

```
partition -cell pll;
choose nanosim -nspice pll.spi -C cfg;
set rmap resis.map;
set bus_format <%d>;
```
Running NIV

• To run the VCS compile:
  ▪ vcs +ad top.v
    • the “+ad” option looks for the file “vcsAD.init”
  ▪ The VCS compile will output the file “simv” by default.
    • use -o =<filename> to compile to a different filename

• To run the VCS simulation:
  ▪ simv

• -R option allows one step simulation
  ▪ vcs +ad top.v -R
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
  - Advanced Features
- Summary
Programming Language Interface

• The interface between arbitrary C language functions and the Verilog Simulator
  - part of the Verilog standard IEEE 1364
  - very simple usage with VCS

• Very simple usage with VCS, requires:
  - pli.tab table
  - which C routines match $tasks
  - controlling access into VCS structures (like SDF annotation)
    - C code of your PLI calls
    - $task call that initiates a call to the C PLI code
What is pli.tab file?

- Dispatch table between Verilog simulation and C routine
- It is the VCS form of the XL file veriuser.c
  - `$VCS_HOME/platform/util/veriuser_to_pli_tab` utility to convert
  - 3rd Party usually has pli.tab file for VCS
- Defines the new task or function and its entry points
- One to three entry points (routines) to be associated with new `$task`
  - check: called at time 0 (check for legal args, etc.)
  - call: called during simulation at time that `$task` is invoked (does the actual work of the task or function)
  - misc: called at various times, some under user control (at the end a timestamp, when argument changes value, etc.)
  - the check, call and misc routines use the library routines to communicate with the simulation
pli.tab File Format

• Format:

// is a comment
$task_name [pli_spec] [acc_spec]

$foo check=foo_check call=foo_call size=64 acc=r:%TASK

- all fields are optional
- PLI table and object file(s) containing check, call and misc routines are specified on VCS command line

> vcs test.v -P pli.tab my_routines.o

- multiple PLI table (-P options) may be specified for convenience
PLI Example

- PLI Function
  - Retrieves a numeric argument from the run time command line to Verilog simulation

Usage example:

VCS compile command line:
> vcs test.v -P pli.tab my_routines.o

VCS simulation executable command:
> simv +foo+40

Verilog use and call to function:

```verilog
integer val;
initial val = $get_plusarg_num("foo+");  
```

Result:

'value' will equal 40 (decimal)
DKI with Debussy

• Compile with DKI .tab file provided by Debussy:

```bash
vcs -P <debussy_inst_dir>/share/PLI/vcsd/SOLARIS2/vcsd.tab \ <debussy_inst_dir>/share/PLI/vcsd/SOLARIS2/pli.a +vcsd \ <other_compile_options.>
```
Useful PLI Resources

- Sutherland’s “The Verilog PLI Handbook”
- Chris Spear’s website: http://www.chris.spear.net/pli/index.htm
- Reading & Writing Files from Verilog
  - Strobe compare
  - $value$plusargs: The PLI application allows you to read the value part of the plus argument in decimal, binary, octal, hexadecimal, floating point, or string.
  - Find X - Locate the unknowns in your simulations. Great for debugging reset problems.
  - Math - Use $sin, $log, and other math functions in your Verilog simulations.
  - Write Stdio - How to use the VCS file pointer for stdout - useful for C programmers.
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
  - Advanced Features
- Summary
Back Annotating SDF Files

- Two methods for back annotating SDF files
  - Compiled SDF
    - As long as no SDF configuration file present
    - As long as string literal is used in $sdf_annotate task
    - As long as scale arguments NOT used in $sdf_annotate task
  - Otherwise use +oldsdf for traditional runtime annotation
- Advantages of compiled SDF vs. run-time SDF:
  - Parses compiled version must faster than ASCII version
  - lowers memory usage up to 2-5x
  - faster simulation by up to 2-5x
  - don’t have to create complex sdf.tab file
Precompiled SDF

• New switch: `+csdf+precompile`
  ▪ Creates the precompiled SDF file in the same directory as the ASCII text SDF file and differentiates the precompiled version by appending ".c" to the ASCII text SDF file's extension.

• Only takes affect when the SDF file changes
  ▪ So you can keep it in most scripts

• When you recompile your design, VCS will find the precompiled SDF file is the same directory as the SDF file specified in the `$sdf_annotate` system task. You can also specify the precompiled SDF file in the `$sdf_annotate` system task.

• `+csdf+precomp+dir+<dir>` option to specify the path where you want VCS to write the precompiled SDF file.
+allmtm

- If SDF annotation file has MTM triplets, by default, VCS compiles a “typical version”
- To have VCS create compiled SDF files for all triplet versions, i.e. min:typ:max, use the +allmtm switch at compile time.
- Then at runtime, specify the delay mode to use
  
  +maxdelays or +mindelays or +typdelays
  (default)
Runtime SDF

• Reading the ASCII SDF file at run-time
  ▪ uses the ACC capability
  ▪ requires table file and -P compile-time switch and
    +oldsdf for version before v5.1
    > vcs design.v -P sdf.tab +oldsdf
  ▪ reads in the ASCII file specify in the $sdf_annotate
    system task at run-time
  ▪ disadvantages
    • slower simulation time
    • larger memory capacity
    • requires sdf.tab file
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - **Graphical Debugging**
  - Advanced Features
- Summary
Complete Debugging Environment

- Interactive, text-based and post-processing
- Backward or forward execution in time
- Drag & drop and synchronization of data between windows

- Hierarchy Browser
- Waveform Viewer
- Logic Browser
- Source Level Debugger
- Register Window
- Interactive Control
VirSim Interface Windows

- Provides intuitive drag-and-drop interface
- Enables interactive, text-based debug
- Supports incremental compilation
- Delta Cycle feature
- User Expressions
- Event Searching in either time direction
- Offers event origin tracking
- Supports configuration files
VCD+

- VCD+ is PLI generated binary history file
  - History file used to record
    - time of transition
    - values of nets and registers
    - order of source code execution
    - design hierarchy
  - Built-in system tasks are provided for controlling the contents of the VCD+ file and size
- Significant advantage over standard VCD ASCII format
  - Dramatically improves files sizes (5-10x reduction)
- VCD+ options to control performance and files sizes
- For better Performance, try and use DKI to bypass PLI to capture .vpd files (good for non-interactive/debug sim that just dumps waves)
  - Compile with -PP +vcsd switch
  - Cannot have any other interactive switches i.e (-line -l, etc)
VCD+ Rules

• `$vcdpluson` and `$vcdplusoff` tasks accept the same arguments as `$dumpvars` task

• `$vcdpluson` and `$vcdplusoff` tasks executed in the same simulation time period may execute in any order

• `$vcdpluson` and `$vcdplusoff` tasks may be inserted in source files or entered at the simulation interactive prompt

• Requires compile time switches `-PP` or `-RI` (links `xvcs.tab` file)

• Default file name is `vcdplus.vpd`. Default file name can be changed with `+vpdfile+filename.vpd` on `simv` command line

• Use `+vcsd` for non-interactive compiles and simulation (No `-RI -RPP -RIG`) for faster dumps
**VCD+ Command Line Switches**

- **+vpdfile+filename.vpd**: Defines the name of the VCD+ file
- **+vpdbufsize+<MB>**: Change internal buffer size (default is calculated)
- **+vpdfilesize+<MB>**: Limit the size of the VCD+ file
- **+vpdupdate**: Allows simultaneous writes and reads to the VCD+ file
- **+vpddrivers**: Includes driver information in the VCD+ file, enables show drivers command
- **+vpdports**: Includes port information in the VCD+ file
- **+vpdignore**: Ignores any $vcdplus calls in source code
**VCD+ Built-in System Tasks**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vcdpluson</td>
<td>Begin/Stop recording value changes for the specified scopes.</td>
</tr>
<tr>
<td>$vcdplusoff</td>
<td></td>
</tr>
<tr>
<td>$vcdplusdeltacycleon</td>
<td>Enables delta cycle recording</td>
</tr>
<tr>
<td>$vcdplustraceon</td>
<td>Begin recording Verilog statements execution information (requires -line switch),</td>
</tr>
<tr>
<td>$vcdplustraceoff</td>
<td>Stop recording verilog statements execution information</td>
</tr>
<tr>
<td>$vcdplusautoflushon</td>
<td>flush for every interrupt ($stop)</td>
</tr>
<tr>
<td>$vcdplusautoflushoff</td>
<td></td>
</tr>
<tr>
<td>$vcdplusflush</td>
<td>Flushes the RAM buffer into the VCD+ file</td>
</tr>
<tr>
<td>$vcdplusevent</td>
<td>Displays a symbol on a waveform and maps it in the Logic Browser</td>
</tr>
<tr>
<td>$vcdplusfile(&quot;text&quot;)</td>
<td>Writes to specified .vpd file. Can open,write,close multiple vpd files</td>
</tr>
<tr>
<td>$vcdplusclose</td>
<td>Closes current .vpd file</td>
</tr>
</tbody>
</table>
Selective Capturing of a VCD+ File

• For better performance use target based dumping
  ▪ \$vcdpluson (<level>,scope,<signal>);
    ◦ specify levels of hierarchy of a specific module
    ◦ specify individual net or registers
    ◦ specify all the variables in a selected module instance
    ◦ default, all levels and all signals
  ▪ \$vcdplusoff (<level>,scope,<signal>);

• Capture time slices
  ▪ use \$vcdpluson in conjunction with \$vcdplusoff and #delay
  ◦ stop and resume recording anytime during simulation
  ◦ the gray represents unrecorded data in the waveform window

• Start by dumping only the first few levels and work down until the problem is isolated
Example of Capturing a VCD+ History File

module top;
  moduleA u1 (a,b,c);
  moduleB u2 (d,e,f,g);
  moduleC u3 (siga,sigb,sigc);
  ....

  // save all signal data in module u1 from time 100 to 300,
  // save all the variables in module u2 along with 5 levels of
  // hierarchy from time 200 to 500,
  // save two variables in module u3 starting at 600
  fork
    #100 $vcdpluson(top.u1);
    #200 $vcdpluson(5,top.u2);
    #300 $vcdplusoff(top.u1);
    #500 $vcdplusoff(5,top.u2);
    #600 $vcdpluson(top.u3.siga,top.u3.sigb);
  join
Running Simulation w/ VirSim

vcs des.v -f file.f -Mupdate -RI [-line +cli other_opts] +cfgfile+default.cfg

des.v  Verilog source code
-f file.f A file that contains a list of other Verilog files or VCS
-Mupdate use incremental compile and make sure to update the Makefile
-RI starts VirSim immediately after compilation
-RIG runs existing executable in VirSim (no recompilation)
-RPP +vpdfile+file.vpd Post Process Mode, opens file.vpd file

Enables debugging commands including value change breakpoints and force and release

OPTIONAL:
-line enable line stepping
+cli enable command line interface commands
+cfgfile+ load a VirSim Configuration file
Opening a VCD+ History File

- Post-Process viewing of vpd file:
  - Use `-RPP` (run Post Process mode) switch
    - `vcs -RPP <sources.v>`
  - from the Hierarchy Window, File => Open => *.vpd (select a vpd file)
- Post Process by opening a specific vpd file
  - compile using `-vpdfile+` switch
    - `vcs -RPP +vpdfile+design.vpd +cfgfile+default.cfg`
- To include source debugging
  - add source code on the command line
    - `vcs -RPP +vpdfile+design.vpd +cfgfile+default.cfg -f source_code`
    - required for Logic Browser
- Supports multiple history files
  - open multiple files either manually or from the command line
    - use the design icon to display a design hierarchy (VCD+ data)
Other VCD+ Tips and Suggestions

- For maximum run time performance try this:
  - use compiler directives `ifdef` and `endif
    ```
    `ifdef dumpme
      $vcdpluson();
    `endif
    ```
  - dumping is controlled by a compile time switch
    `+define+dumpme`
  - Its not recommended to use $test$plusargs for example:
    ```
    initial begin:enable_dumping
      if ($test$plusargs("dumpall")
        $vcdpluson();
      else if ($test$plusargs("dump+moduleA")
        $vcdpluson(1,moduleA);
      end
    ```
  - Even if dumping is disabled at run-time, the fact that
    `$vcdpluson` is enable at compile time means significantly
    slow simulation.
VCD+ Tips and Suggestions

• For better performance
  ▪ add \texttt{+vpdbufsize+nn}
    • increases internal RAM buffer
    • rule-of-thumb 1M for every 5K gates
    • bigger the buffer the faster simulation runs
  ▪ avoid recording Verilog statements execution
    • \texttt{$vcdplustraceon}$
    • big performance hit (by 6-8x)
    • selectively dump only small modules
  ▪ watch out for dumping in -RI mode (interactive debug)

• For Gate level design use
  ▪ \texttt{+nocelldefinedpli+1}
    • disables PLI access to the internals of cells
    • still traces all cell I/O pins
    • reduces internal cell activity (25% improvement)
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
  - Advanced Features
- Summary
Usability Features

• VCS has built-in Profiler which is easier to read versus UNIX profiler
  ▪ Compile with +prof switch
  ▪ `vcs.prof` is the report that is generated during simulation

• VCS has a race detection tool to detect common races during simulation.
  ▪ Compile with +race switch
  ▪ `race.out` file is created during simulation that reports races found. Perl script in `$VCS_HOME/bin` to filter out duplicates

☞ Recommend to do these occasionally since options carry overhead during compile and runtime.
Profiling Goals

• Make sure there aren’t any hidden bottlenecks.
• Get big performance gains for small design style improvements (e.g. better latch model)
• Run a profile regularly
• Use information to see if you can modify code to achieve performance
• Built-in profiler that aids in identifying
  ▪ Module instances in the hierarchy that use the most CPU time
  ▪ Module definitions whose instances use the most CPU time
  ▪ Verilog constructs in those instances that use the most CPU time
Profiling is easy with +prof

• Add +prof to the compilation command line.
• Simulation produces a “vcs.prof” file that is organized into “views”
  ▪ Top Level View
  ▪ Module View
  ▪ Instance View
  ▪ Module Construct Mapping View
  ▪ Top Level Construct View
  ▪ Construct View Across Design
• Provides information on
  ▪ Design
  ▪ VCS Kernel
  ▪ PLI
  ▪ VCD (dumping)
Race Detection

- Compile with `+race` option to detect races during runtime.
- Output file `race.out` is generated after simulation completes.
- Perl script is provided to post process `race.out`.
- VCS invokes perl script by default. Perl script by default finds Perl executable at `/usr/local/bin`.
- Change PostRace.pl in `$VCS_HOME/bin` to specify proper path to Perl5> to execute script.
PostRace.pl

- PostRace.pl offers some Post Processing Capability:
  -hier module_instance
    - The new report lists only the race conditions found in this instance and all module instances hierarchically under this instance.
  -sig signal
    - Specifies the signal that you want to examine for race conditions. You can only specify one signal and do not include a hierarchical name for the signal. If two signals in different module instances have the same identifier, the report lists race conditions for both signals.
  -minmax min max
    - Specifies the minimum, or earliest, simulation time and the maximum, or latest, simulation time in the report
  -nozero
    - Omits race conditions that occur at simulation time 0.
  -uniq
    - Omits race conditions that also occurred earlier in the simulation. The output is the same as the contents of the race.unique.out file.
race.out

1. module test;
2. reg a,b,c,d;
3. always @(a or b)
4. \text{c} \gets a \& b;
5. always
6. begin
7. a = 1;
8. #1 a = 0;
9. #2;
10. end
11. always
12. begin
13. #1 a = 1;
14. \text{d} \gets b \mid c;
15. #2;
16. end
17. initial
18. begin
19. $\text{display("}\%m \text{c} = \%b",\text{c});$
20. #2 $\text{finish};$
21. end
22. endmodule

Chronologic Simulation VCS RACE REPORT
0 "c": write test (exp1.v: 4) && read test (exp1.v:19)
1 "a": write test (exp1.v: 13) && write test (exp1.v:8)
1 "c": write test (exp1.v: 4) && read test (exp1.v:14)

Write-Read Race: Does \text{d} get evaluated before \text{c} gets assigned?
Radiant Optimizations Goals

• Radiant optimizations are enabled with a `+rad` compile-time switch
• Designs are never written for simulation performance.
  ▪ written to be very close to the hardware they model
  ▪ they have a lot of redundant logic and low-level descriptions, which slows down simulation
• Great performance gains for functional simulation for both RTL and gate-level designs.
• Eliminates inefficient code for the VCS back end (e.g. trans, xmr’s, case statements)
Examples

• Input Verilog

```verilog
wire [7:0] a;
integer b;
assign a[0] = (b == 0);
assign a[1] = (b == 1);
assign a[2] = (b == 2);
assign a[3] = (b == 3);
assign a[4] = (b == 4);
assign a[5] = (b == 5);
assign a[6] = (b == 6);
assign a[7] = (b == 7);
```

• Optimized Verilog

```verilog
wire [7:0] a;
integer b;
assign a = (1 << b);
```
Human Sources for Information and Help

- Contact the Support Center for technology support:
- Before you contact Synopsys, please make sure:
  1. You have checked the problem using Acroread to browse the SOLD CD (SOLD: Synopsys On Line Documentation)
  2. You have checked the problem using SolvNET (http://solvnet.synopsys.com/login/designsphere)
  3. Or browsing on SNUG (http://www.snug-universal.org)

- E-mail: twhelp@synopsys.com
- Phone: 0800-079595

- Application Consultants:
  - Process and tool expertise available worldwide

- Consultants:
  - Available for in-depth, on-site, dedicated, custom consulting
CheatSheet

- **vcs -RI**
  - Compile and then run interactively with VirSim GUI immediately after compile
- **vcs -RIG**
  - Run interactively with VirSim GUI w/o compiling, use current executable
- **vcs -R -I**
  - Compile and then run simulation executable w/o GUI
- **vcs -RPP +vpdfile+ file.vpd +cfgfile+ file.cfg sources.v**
  - View waveform file using GUI in post process mode
- Compile switches that slow you down:
  - -I, -P, +cli -line, +oldsdf, +neg_tchck
- Compile switches that speed you up:
  - -nospecify, -notimingchecks, +rad, +nbaopt
- To enter interactive mode either from executable:
  - Use the -s flag at run time, e.g. simv -s - this will halt the simulation at time zero and enter interactive mode
  - ^c will halt the simulation while it is running
  - Use $stop in the Verilog source
  - In interactive mode VCS prompts you for interactive commands, help to list commands
Agenda

- Overview
- VCS Coverage Metrics (VCM)
  - Basic Commands
  - Coverage Metrics, DirectC & NIV
  - PLI Usage
  - SDF Backannotation
  - Graphical Debugging
- Advanced Features

- Summary
Summary

• VCS is superior to 3rd-party simulators in terms of performance and integration
• VCS’ performance is continuing improved without extra buy-in charges
  • RTL/Gate-level performance, Direct-C
• Integration with Synopsys’ tools are rapidly provided
  • Coverage, testbench automation: CM, VERA
  • Mixed-HDL: Scirocco
  • Mixed-Signal: Nano-SIM
  • Models: SmartModel, FlexModel
  • Synthesis & STA: DC & PrimeTime
Synopsys Verification Solution
Faster & Smarter

- TestBench Automation
- Code Coverage
- Mixed-HDL
- LEDA® Checking
- HDL Checking
- Mixed-Signal
- Circuit Simulation
- Power Analysis
- NanoSim™
- PrimePower
- Formality®
- PrimeTime® PathMill®
- Mixed-HDL
- DesignWare® SmartModel®
- Verification IP & Models
- Formal Verification
- Static Timing Analysis
- Vera®
- CoverMeter
- VCS Scirocco