VCS

VCS

1. VCS Flow Overview
2. Coverage
3. Partition Compile
4. FGP
5. X-Prop
6. Save & Restore

1. VCS Flow Overview

Phase	VCS - Verilog Only	VCS - VHDL/Verilog
Setup Map logical libraries		vi synopsys_sim.setup mkdir library
Analysis	vcs top.v	vlogan vhdlan
Elab, Compile		vcs top -o simv
Simulation	simv	simv

3-step – for Verilog+VHDL+SystemC

Map Logical Libraries
`synopsys_sim.setup`

Analyze Source Files

```
vlogan <files.v>
vhdlan <files.vhd>
syscan <files.cpp>
```

Elaborate/Compile Design

```
vcs <options> <design_top>
```

Simulate

```
simv <options>
```

2-step – for Verilog-only

Elaborate/Compile Design

```
vcs <options> <files.v>
```

Simulate

```
simv <options>
```

Verilog Analyzer-vlogan (3-step flow)

```
%> vlogan [-help] [+define+macro] [-f file] [-q] [-full64] [-sverilog]
[+incdir+idir] [-l logfile] [-v file] [-y libdir] [+libext+lext]
[-work logical_lib] [-resolve] [+nospecify] [-kdb]
[-timescale=1ns/1ns] [+notimingchecks] verilog_design_files
```

Hint! For faster analysis, specify multiple files per command— use a file list!

- Parses Verilog source into logical libraries
- Instantiated VHDL design units are resolved during elaboration
  : Resolves during analysis when “-resolve” used
- Common Verilog file parsing options are available (-y, -v, -f etc.)
- Example
  : %> vlogan and2.v –y /cad/library/tsmc65nm/Verilog +libext+.v

+define+<macro>	- Defines a macro in the Verilog source
-f file	- Specify files as well as command options
-l file.log	- Create log file
-q	- Quiet (no internal messages and banner)
-v <lib_file>	- Specify a Verilog library file
-y <libdir>	- Specify a directory of Verilog library files
+libext+<ext>	- Specify library file extensions (used with -y)
-work <libdir>	- Analyze into specified logical library
+nospecify	- Remove timing in specify blocks
+notimingchecks	- Remove timing checks
+v2k	- Enable Verilog 2001 constructs
-sverilog	- Enable SystemVerilog constructs
-timescale=1ns/1ps	- Specify default timescale
+incdir+<dir>	- Specifies search directory for included files
-full64	- Analyzes the design for 64-bit simulation
-kdb	- Generate KDB for Verdi

VHDL Analyzer-vhdlan (3-step flow)

```
%> vhdlan [-nc] [-4state] [-work library] [-vhdl87] [-no_opt] [-full64]
[-output outfile] [-f filename] [-xlrm] [-functional_vital] [-kdb]
[-help] VHDL_design_files
```

- Parses VHDL source into logical libraries
- Analyze VHDL blocks bottom up
- Partial elaboration during “configuration” analysis

-nc	- Suppresses copyright header
-work <library>	- Analyze into specified logical library
-vhdl87	- Enable VHDL-87 syntax (VHDL-93 is default)
-no_opt	- Enables boundary checking, slows simulation
-f optionsfile	- Specify source files and switches
-xlrm	- Allows relaxed/non-LRM compliant code
-4state	- Enable optimized 4 state simulation mode
-functional_vital	- Removing all timing from VITAL models
-kdb	- Generate KDB for Verdi

- Note: Library name for –work must be a valid logical library
  - Unix paths not accepted on vhdlan command line!

Common vcs options

VCS has many options. Use 'vcs –help' for listing

-o <simname>	- Rename simulation executable
-ucli	- Enable Tcl command-line interface
-debug_access(+option)	- Enable debug capabilities
-l <logfile>	- Create runtime log file
-kdb	- Generate KDB for Verdi
-R	- Run simulation immediately after compile
-P pli.tab	- Compile user-defined PLI table
<.c|.o files>	- Add C or object files to compile
-xlrm	- Allows relaxed/non-LRM compliant code
-cm <options>	- Enable coverage options
-full64	- Enable elaboration and simulation in 64-bit mode

Common simv options

-pv <param=value>	- Override runtime Verilog parameter
-gv <gen=value>	- Override runtime VHDL generics*
-ucli	- Stop at Tcl prompt upon start-up
-i <run.tcl>	- Execute specified Tcl script upon start-up
-l file.log	- Create runtime log file
-gui	- Start interactive GUI session
-cm <options>	- Enable coverage options

- Simulator executes "run; quit" automatically if no Tcl scripts provided

*Note: Only generics not effecting elaboration results may be overwritten.
Otherwise, specify during compilation.

3-step flow (normal VCS script)

Map Logical Libraries
synopsys_sim.setup

```
echo 'WORK > DEFAULT' > synopsys_sim.setup
echo 'DEFAULT : ./work' >> synopsys_sim.setup
echo 'LIB_VHDL : ./work.vhdl' >> synopsys_sim.setup
mkdir work.vhdl

vlogan \
        -kdb \
        -full64 \
        +define+LINK \
        -sverilog \
        -timescale=1ns/1ps \
        -f lib_vcs.f rtl_inc.f test.sv \
        -l ./vlogan.log
vhdlan \
        -kdb \
        -full64 \
        -f vhdl.f \
        -l ./vhdlan.log \
        -work LIB_VHDL
```

Need to make directory
before run for vhdlan
Not needed for vlogan

```
vcs \
        -kdb \
        -full64 \
        -reportstats \
        -l ./vcs.log \
        -debug_access \
        +vcs+lic+wait \
        +notimingcheck \
        -timescale=1ns/1ps \
        test
        #-top test, -top WORK.test

./simv -reportstats -l ./simv.log
```

Need to indicate
top module

2-step flow

```
2 step flow

vcs \
  -kdb \
  -sverilog \
  -full64 \
  -reportstats \
  -timescale=1ns/1ps \
  +define+MY_MACRO \
  +incdir+/path/to/verilog/include/dir \
  -y /path/to/verilog/library/dir \
  -v /path/to/verilog/library/file \
  one_verilog_file.v \
  -l vcs_comp.log \
  -f ./more_files_in_here.f \
  -o ./simv_2step

./simv_2step -reportstats -l vcs_sim.log
```

Generate kdb file for Verdi
(not needed to read design when invoke Verdi)

Enable System Verilog constructs

64bit simulation mode

Generate performance / memory usage report

Specify default timescale

create Log file

Can modify simulation executable file name with -o switch

```
*Invoke Verdi

1)
Simply load the FSDB, the KDB will be loaded
automatically
Use the -dbdir option
(When simv.daidir directory is existed)

%> verdi -dbdir <path to>/simv.daidir &

2)
If you want to open with existed fsdb file,
use the -ssf option to load it:

%> verdi -ssf xxx.fsdb -dbdir <path to>/simv.daidir &
```

```
-f <file>        : Specify files as well as command options
-v <lib_file>    : Specify a Verilog library file
-y <libdir>      : Specify a directory of Verilog library files
+incdir+<dir>    : Specifies search directory for included files
+define+<macro>: Defines a macro in the Verilog source
```

2-step flow

example "simv" command lines

```
# Batch/regression mode
./simv \
  -ucli -i script.tcl \
  -gv MYGENERIC=3

# Interactive mode: with Verdi simulation GUI
./simv -verdi
```

-pv <param=value>
Override runtime
Verilog parameter
-gv <gen=value>
Override runtime VHDL
generics*

```
%> cat script.tcl
fsdbDumpfile ./dump/test_env_clkrst.fsdb
fsdbDumpvars 0 test_env
call {readmemh
"/path/to/pram.hex",test.dut.ram_array)}
force test.dut.n1 1'b0
run
exit
```

Please refer to
$VCS_HOME/doc/UserGuide/pdf/ucli_ug.pdf

Gate-level Simulation

```
vcs \
    -full64 -kdb -debug_access \
    TB_TOP.v netlist.v \
    -sdf max:top.dut:sdf_file.sdf \
    +optconfigfile+sample.cfg \
    -v ./src/netlist_dir/lib.v \
    -y ./src/vendor_a_dir/ \
    -y ./memories/vendor/ \
    +libext+.v+.V \
    -o simv_timing \
    -negdelay \
    +neg_tchk \
    +maxdelays \
    -l comp.log

./simv_timing -l simv_timing.log
```

Enables the use of negative values in IOPATH and INTERCONNECT entries in SDF files.

Enables negative values in timing checks.

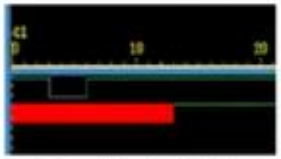
Specifies using the maximum delays in min:typ:max delay triplets in module path delays and timing checks

```
%> cat simple.cfg
module {FULL_ADDER} {noTiming};
instance {TB.INST1.fa0.temp1} {noTiming};
instance {TB.INST1.fa1.temp2} {noSpecify};
instance {TB.INST1.fa0.temp2} {noIopath};
```

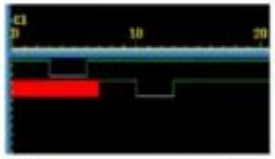
Transport Delays
*VCS defaults to inertial delay mode

+transport_path_delays :to enable module path delays
+transport_int_delays :to enable net delays

IOPATH delay : 7ns



Inertial Mode



Transport Mode

+nospecify :Remove timing in specify blocks
+notimingchecks :Remove timing checks

+sdfverbose :
This option enables the display of all back-annotation warning and error messages.(Default 10)

-diag=sdf:verbose :
VCS provides additional diagnostics.VCS generates a file sdfAnnotateInfo.

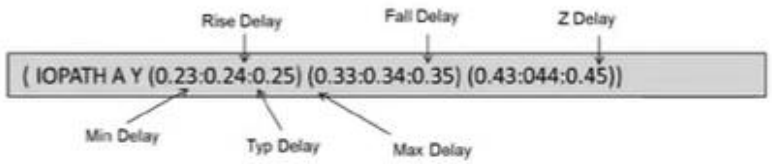
SDF(Standard Delay Format) Back-Annotation

```
( SDFVERSION "OVI 3.0" )
( DESIGN "F76A890" )
( VERSION "4.2.10" )
( VOLTAGE 1.32::1.32 )
( TEMPERATURE -40::-40 )
( TIMESCALE 1ns )

( CELL
 ( CELLTYPE "SDFF" )
 ( INSTANCE U0_FF)
 (DELAY
  ( ABSOLUTE
  ( IOPATH CLK Q ( 0.046::0.046 )( 0.04::0.04 ))
  ( IOPATH PREZ Q ( 0.088::0.088 ))
)
 )
 ( TIMINGCHECK
  ( WIDTH ( negedge CLK ) ( 0.036::0.036 ) )
  ( WIDTH ( posedge CLK ) ( 0.035::0.035 ) )
  ( SETUPHOLD SD ( posedge CLK ) ( 0.06::0.06 ) ( 0.026::0.026 ) )
  ( SETUPHOLD SCAN ( posedge CLK ) ( 0.066::0.066 ) ( 0::0 ) )
  ( SETUPHOLD D ( posedge CLK ) ( 0.029::0.03 ) ( 0.002::0.002 ) )
 )
```

SDF Snippet

```
[Sdf annotation in Verilog]
$sdf_annotate("sdf_file.sdf","top.dut","","","maxinum");
[VCS command line]
-sdf max:top.dut:sdf_file.sdf
```

Delay representation in SDF



- Min/Typ/Max Selection
- Transition Delay
  - 12 Values: 0→1, 1→0, 0→Z, Z→1, 1→Z, Z→0, 0→X, X→1, 1→X, X→0, X→Z, Z→X
  - 6 Values: 0→1, 1→0, 0→Z, Z→1, 1→Z, Z→0
  - 3 Values: 0→1, 1→0, ?→Z
  - 2 Values: 0→1, 1→0

```
specify
  (CLK => Q ) = (0.1,0.1);
  (PREZ => Q ) = (0.1, 0.1);
c
  $setuphold(posedge CLK, D, 0.1, 0.1, notifier,,, CLK_DEL, D_DEL );
  $setuphold(posedge CLK, SD, 0.1, 0.1, notifier,,, CLK_DEL, SD_DEL );
  $setuphold(posedge CLK, SCAN, 0.1, 0.1, notifier,,, CLK_DEL, SCAN_DEL );
endspecify
```

After sdf annotation

```
specify
  (CLK *> Q ) = (0.046,0.04);
  (PREZ *> Q ) = (0.088, 0.088);
  $setuphold(posedge CLK, D, 0.06, 0.026, notifier,,, CLK_DEL, D_DEL );
  $setuphold(posedge CLK, SD, 0.066, 0, notifier,,, CLK_DEL, SD_DEL );
  $setuphold(posedge CLK, SCAN, 0.03, 0.002, notifier,,, CLK_DEL, SCAN_DEL );
endspecify
```

-debug_access options

Option	Description
drivers	The -debug_access+drivers option enables driver debugging capability.
r	The -debug_access+r option enables the read capability for the entire design.
w	The -debug_access+w option applies write (deposit) capability to the registers and variables for the entire design.
wn	The -debug_access+wn option applies write (deposit) capability to the nets for the entire design.
f	The -debug_access+f option enables the following: • Write (deposit) capability on registers and variables • Force capability on registers, variables, and nets This option is equivalent to -debug_access+w+fn
fn	The -debug_access+fn option applies force capability to the nets for the entire design.
fwn	The -debug_access+fwn option applies write (deposit) and force capability to all nets in the design.
line	The -debug_access+line option enables line debugging. It allows you to use the commands for step/next and line breakpoints. This option is equivalent to -debug_access+pp -line
nomemcbk	The -debug_access+nomemcbk option disables callbacks for memories and multidimensional arrays (MDAs). By default, -debug_access enables callbacks for memories and MDAs.
all	The -debug_access+all option is equivalent to the following commands: -debug_access+line+class+wn+driver+r+w+cbk+f+fn+thread+cbkd The -debug_access+all option enables debug capabilities equal to -debug_all (except it does not apply capability inside cells and encrypted modules).

Is "-debug_access" enough for dump
Yes, absolutely

Other options are for more debuggability.
+r : read value in ucli command line
+w : write(deposit) capability
+f : force capability
+line : breakpoint

If you not want dump memory (MDAs)
please this option
+nomemcbk
By default, -debug_access will dump memory

Requirement: Default Dump option (Function/Task, Memory, Strength)
=> -debug_access ($fsdbDumpfile, $fsdbDumpvars recognition)

Simulation modes

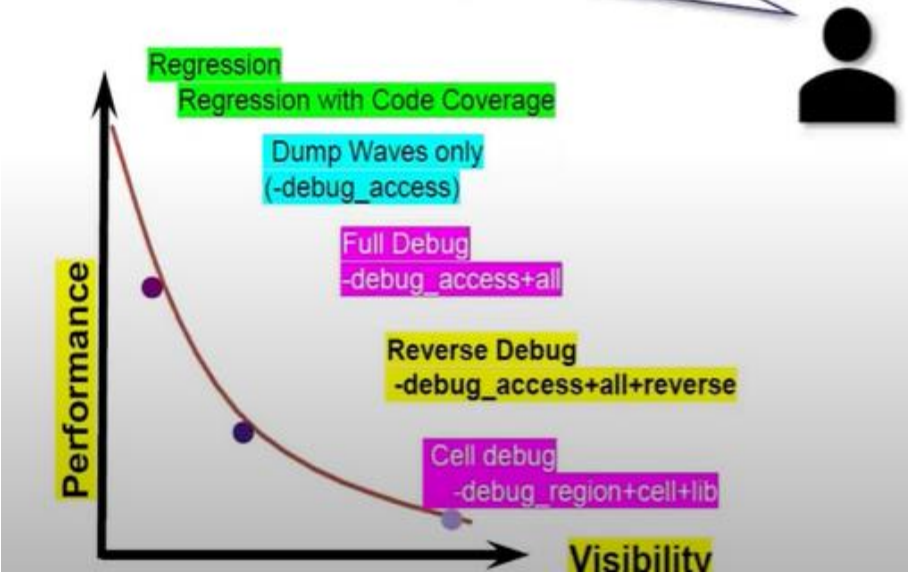
simv	• Batch mode • No user interaction
simv -ucli -i	• Unified Command Line Interface • UCLI TCL commands • Command line debug - step/next/run • Needs -debug_access
simv -gui=verdi OR simv -verdi	• Verdi based debug • Needs -kdb • Needs -debug_access

-debug_region options

requirement :
You must use the -debug_region option along with the -debug_access option at compile time

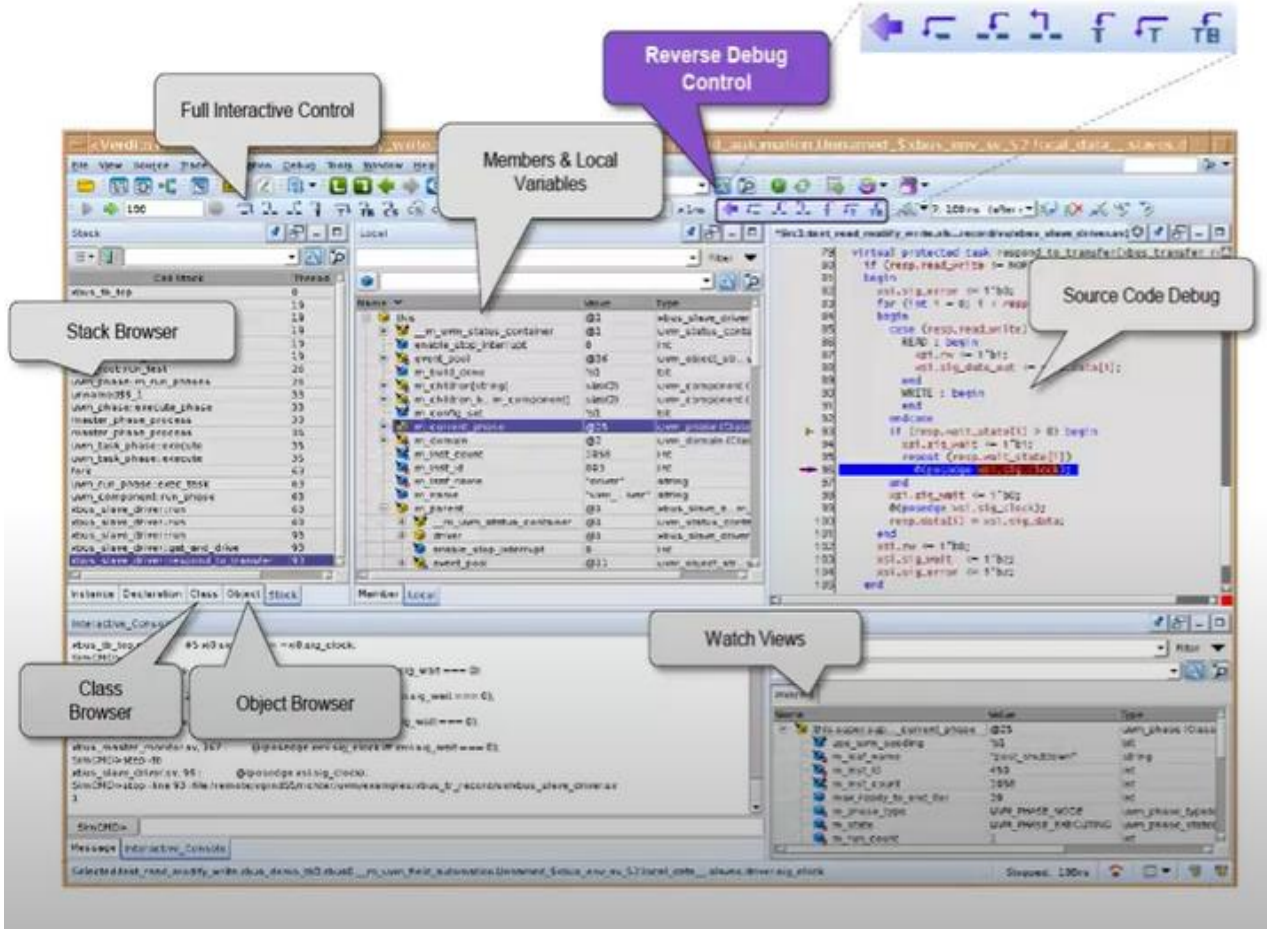
➔ -debug_region=(<option>)(+<option>)*

To dump cell and library
-debug_region=cell+lib



Option	Description	Default Functionality if -debug_region is not specified
lib	Applies debug capabilities to the cells inside libraries.	Debug capability is not applied to the libraries.
cell	Applies debug capabilities to the cells.	Debug capability is not applied to the cells.
encrypt	Applies debug capabilities to the fully-encrypted instances (modules, programs, packages, and interfaces).	Debug capability is not applied to the fully-encrypted instances.
tb	Applies debug capabilities only to the testbench, but does not apply debug capabilities to the standard packages. For more information on what defines testbench, see "Testbench Definition". It does not apply debug capability to the standard packages. The VPD/FSDB dumping of the DUT is not affected by this option.	Debug capability is applied to testbench and DUT.
dut	Applies debug capabilities only to the non-testbench objects. For more information on what defines testbench, see "Testbench Definition".	Debug capability is applied to testbench and DUT.
stdpkg	Applies debug capabilities to the standard packages. You must use the stdpkg option in combination with the tb option. VCS issues a warning message if you use -debug_region=stdpkg only. The -debug_region=tb+stdpkg option applies debug capabilities to both testbench and standard packages	Debug capability is applied to the standard packages.

Using Reverse Debug Feature



- Enables going **back in time** **without** **setting checkpoints**
- **Reverse stepping**
- **Reverse running** with breakpoints
- **Full visibility** on all objects back in time (same as in forward execution)
- Full **What-If** analysis
- **No huge dump files**
- *Debug backwards from an error?*
  ➔ Go back in time

```
%> vcs +debug_access+all+reverse
%> simv -verdi

Verdi> config reversedebug on
Verdi> config keepfuture off
```

- Tools > Preferences > Interactive Debug > Reverse Debug
- Path : $VERDI_HOME/demo/Verilog (Examples)

2. Coverage

Technology	Tool	Associated File Formats
Coverage Collection	VCS	VDB
Coverage Analysis	Verdi-Coverage	VDB
Coverage Merging	URG	VDB
Report Generation	URG/Verdi-Coverage	HTML, Text, Word, Excel
Verification Planning	Verdi-Coverage	HVP, PDF
Coverage Exclusion	URG/Verdi-Cov	Elfile (*.el)

Generating Coverage

- Functional coverage is enabled by default
- Code coverage can be enabled as shown below:

```
➢vcs -cm line+tgl+fsm+branch+cond \
      [-cm_dir comp_covdir.vdb] \

➢simv -cm line+tgl+fsm+branch+cond \
      [-cm_dir run_covdir.vdb]
      [-cm_name test1]
```

```
vcs \
    -kdb -sverilog -full64 \
    -reportstats \
    -timescale=1ns/1ps \
    +define+MY_MACRO \
    +incdir+/path/to/verilog/include/dir \
    -y /path/to/verilog/library/dir \
    -v /path/to/verilog/library/file \
    -f ./more_files_in_here.f \
    one_verilog_file.v \
    -cm line+tgl+fsm+branch+cond \
    -l vcs_comp.log \
    -o ./simv_2step

./simv_2step -reportstats -l vcs_sim.log \
    -cm line+tgl+fsm+branch+cond
```

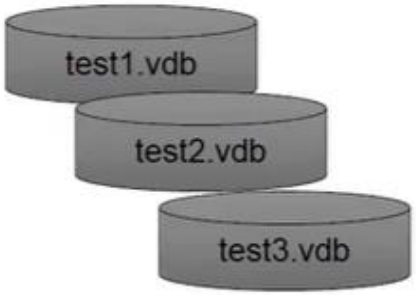
Supported Code Coverage Metrics

- Line Coverage *(-cm line)*
  - Monitors which line in the designs were executed
- Condition Coverage *(-cm cond)*
  - Detailed coverage on conditions
- FSM Coverage *(-cm fsm)*
  - Monitors states, transitions and sequences
- Toggle Coverage *(-cm tgl)*
  - Monitors value changes on signal bits
- Branch Coverage *(-cm branch)*
  - Tracks which branches are taken during simulation (if-then-else branches, case statement items, and ternary operator choices)
- Assertion Coverage *(-cm assert)*
  - Tracks assert statements

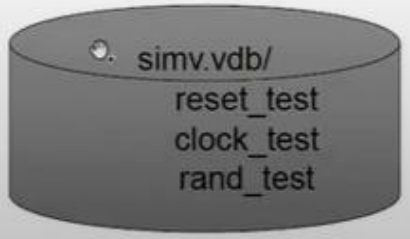
* condition : assign ? (if-else)
branch : if-else if-else, case

Managing Coverage Data Files

- **Compile once, run once**
  - vcs … test1.v -cm line+cond+fsm -o test1
  - test1 -cm line+cond+fsm
  - vcs … test2.v -cm line+cond+fsm -o test2
  - test2 -cm line+cond+fsm



- **Compile once, run many**
  - vcs … top.v -cm line+cond+fsm
  - simv +UVM_TESTNAME=reset_test -cm_name reset_test
  - simv +UVM_TESTNAME=clock_test -cm_name clock_test
  - simv +UVM_TESTNAME=rand_test -cm_name rand_test



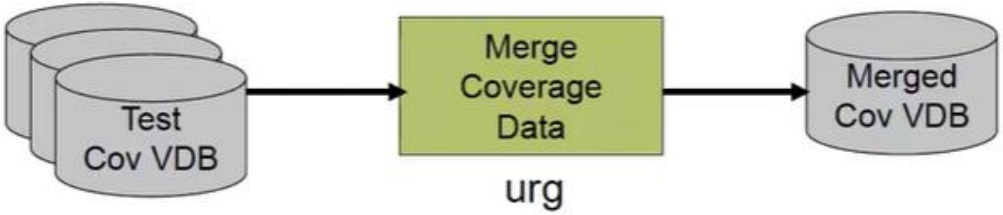
Collecting Partial Coverage

- Enable selective *code* coverage (except assert)
  - vcs … -cm_hier <config_file>
  - Config file example:

    +tree top.dut.core3 // *collect code coverage only on core3*

- Functional Coverage
  - Enabled by default (covergroups and cover properties)
  - Disable if not required (to speed up runtime)
    - To disable covergroups, add *-covg_disable_cg* at runtime
    - To disable cover properties add *-assert nocovdb* at runtime
  - Not affected by *-cm_hier*

URG-Coverage Merging & Reporting



Merged coverage database location	-dbname	• Merge all coverage databases into one
Test coverage database location	-dir	➢`urg -dir test1.vdb test2.vdb test3.vdb -dbname merged`
Metrics	-metric	➢`urg -dir *.vdb -dbname merged`
No report generation	-noreport	• Create a merged report (without database merging)
Parallel	-parallel	➢`urg -dir test1.vdb test2.vdb test3.vdb`
		➢`urg -dir *.vdb`

```
merge_db :
urg -metric line+cond+fsm+assert \
    -dir ../coverage/regression/wb_dma*.vdb \
    -dbname ../coverage/merge/rg1/rg1 \
    -noreport
```

Merging Coverage Databases

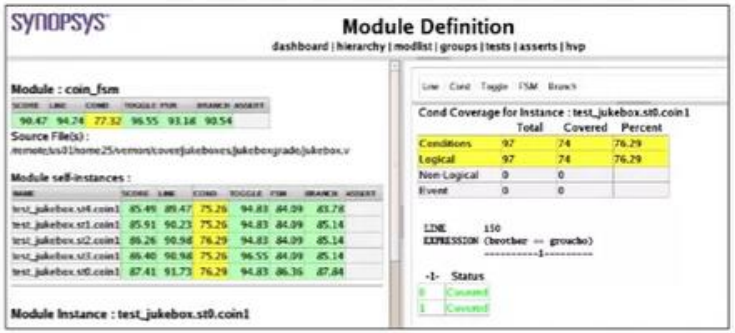
```
RG_DIR = ../coverage  ## Common directory

merge_db:
urg -metric line+cond+fsm+assert \

    -dir ${RG_DIR}/regression/wb_dma*.vdb \

    -dbname ${RG_DIR}/merge/rg1/rg1  \

    -noreport
```

Merge all type of coverage

Coverage source

Merged Coverage name

Don't create report (only merge)



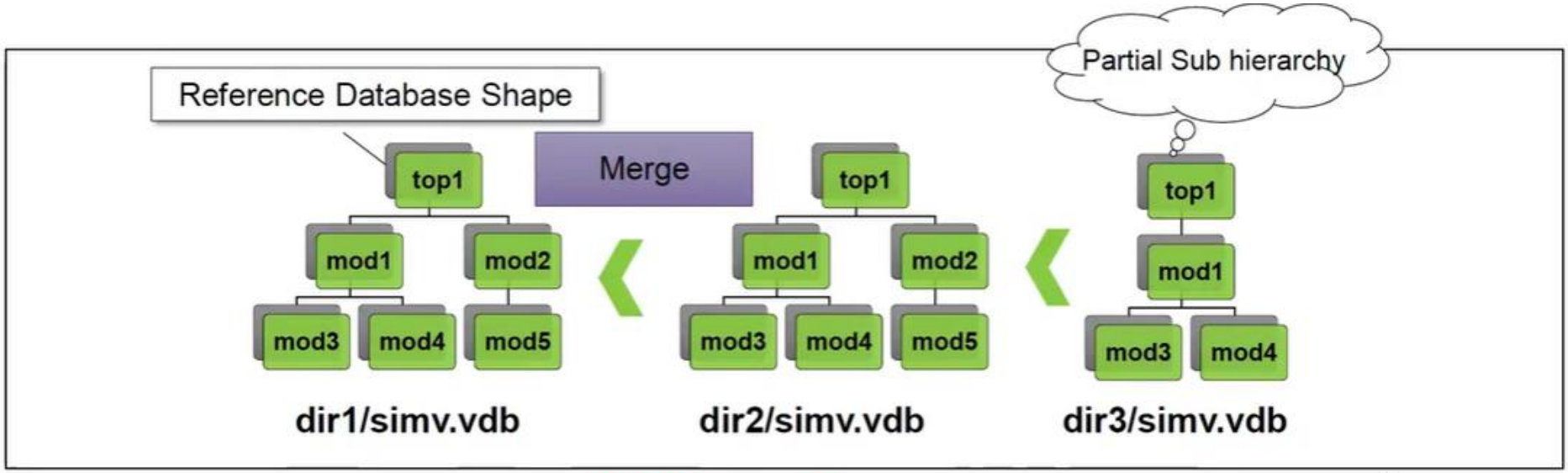
- Merge all coverage databases into one

```
➢urg -dir test1.vdb test2.vdb test3.vdb -dbname merged
➢urg -dir *.vdb -dbname merged
```

- Create a merged report (without database merging)

```
➢urg -dir test1.vdb test2.vdb test3.vdb
➢urg -dir *.vdb
```

Merging Code Coverage




```
%> urg -report urgReport -dir dir1/simv.vdb dir2/simv.vdb dir3/simv.vdb
```

Does this merge?

YES

Works for all metrics

```
%> urg -report urgReport -dir dir3/simv.vdb dir2/simv.vdb \
   dir1/simv.vdb
```

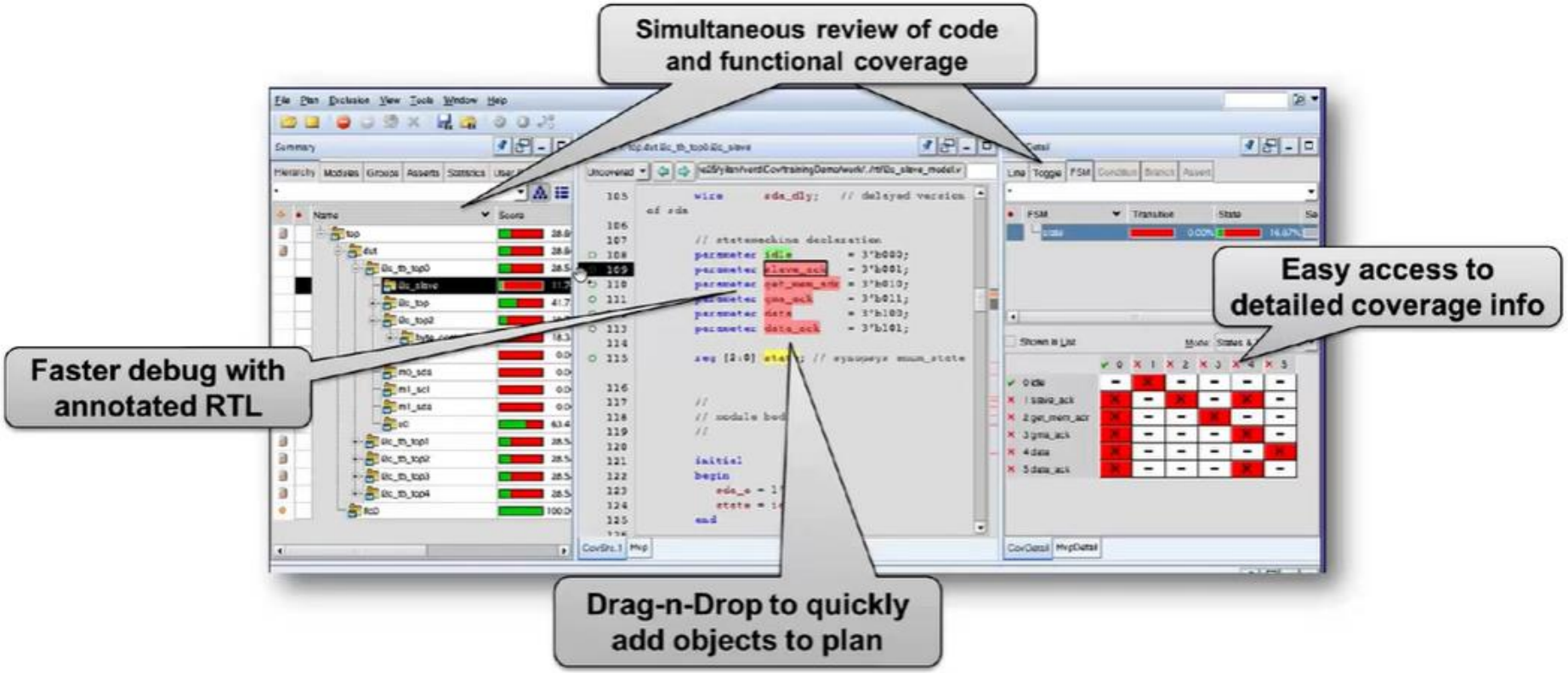
Does this merge?

No

Error-[CMR-VCINF] Version Check Error:Instance not found
Database mismatch: Instance name "top1.mod2" in test file des2/coverage/verilog/test.line is not found in base design. Coverage data of this instance will not be merged.

Verdi Coverage

```
%> verdi -cov -covdir ./simv.vdb -elfile my.el
```



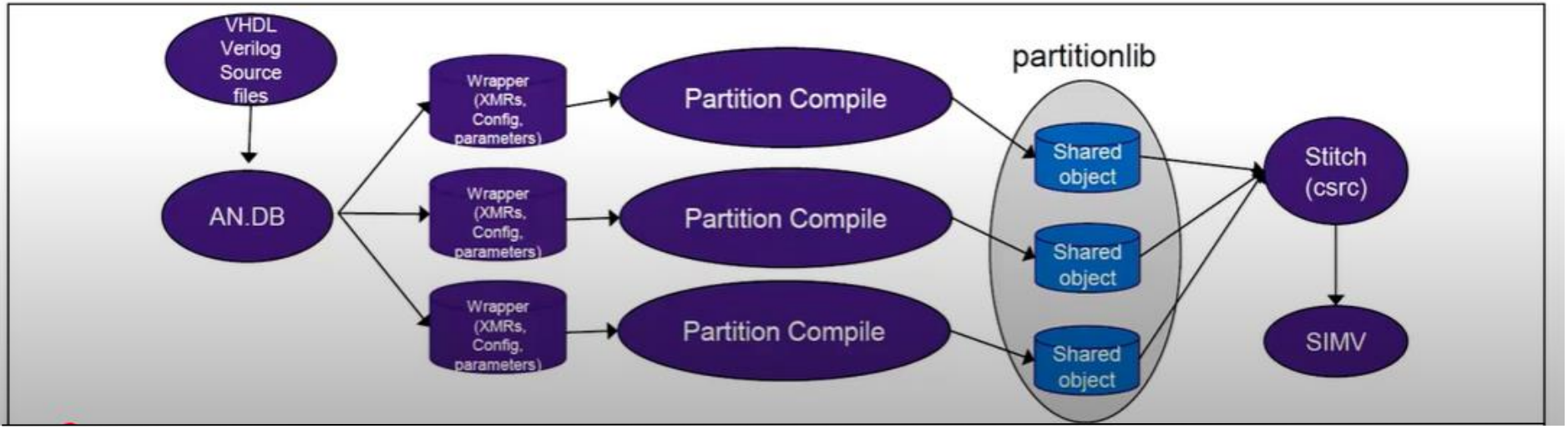
3. Partition Compile

Partition compile reduces compile times and memory utilization by only compiling portions of the design that have changed

- Improve overall turn-around-time (TAT)
  - Avoid recompiling full design for small changes
  - Take advantage of multi-core machines
- On recompile,VCS determines what changed
  - Recompiles only required partitions
- Design is partitioned by user OR tool can do auto-partitioning
- Allows multiple tests to share common compilation data
  - Reduces disk space across multiple tests/users

Partition Compile Flow (1)

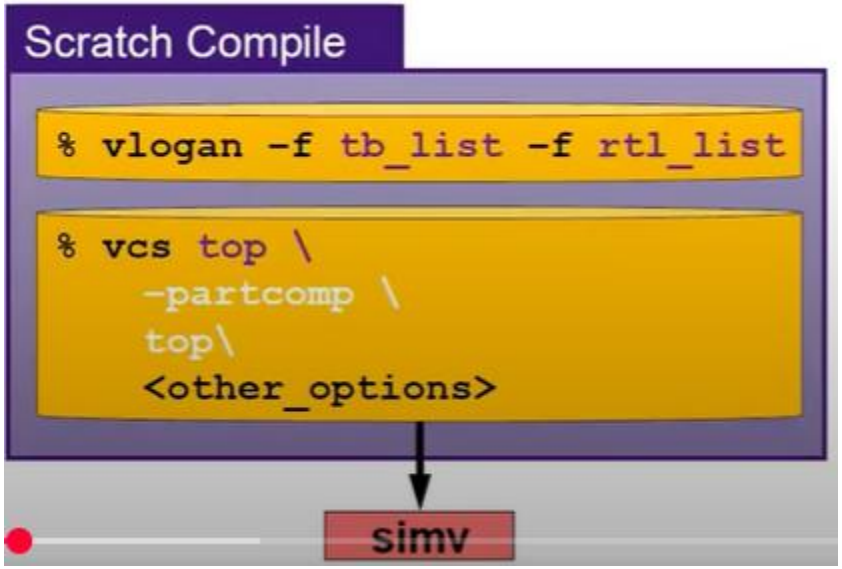
- Each partition is compiled without GLOBAL visibility.
  - Global DR provides information regarding XMRs, parameterization so that each partition can be compiled independently.
  - Partition compile step generates code (shared object) for each partition.
  - Stitch step (vcs-elab) stitches the design together.

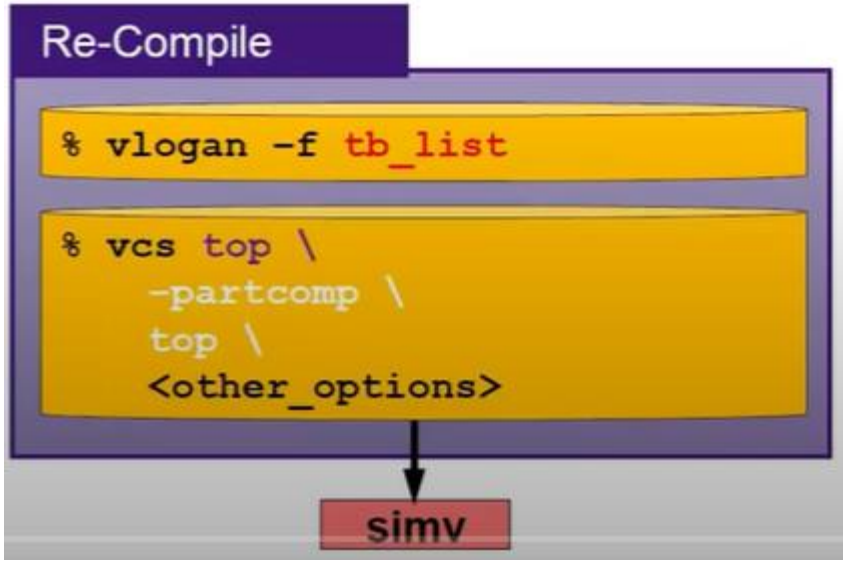


Partition Compile Flow (2)

1. Analyze HDL source files using 'vlogan'/'vhdlan'
2. Compile with **-partcomp** option(s) using 'vcs'
3. Re-analyze changed HDL source files
   - For example, testbench files
4. Re-compile
   - Recompilation options should be the same

- 3 step compile





Partition Compile Message in Log

Scratch Compile

```
Note-[PC_GEN_PARTITION] Generating partition
  Generating new partition 'PACK1' at
'./partitionlib/PACK1_0Qfowb'.

Note-[PC_GEN_PARTITION] Generating partition
  Generating new partition 'DUT' at './partitionlib/DUT_JQkWXc'.

Note-[PC_GEN_PARTITION] Generating partition
  Generating new partition 'TEST' at './partitionlib/TEST_ykyZ3b'.
```

Check the generated partitions

Re-Compile

```
Note-[PC_GUTS_RECOMPILE] Recompiling partition
  Recompiling partition 'TEST' since there are changes in design
units that
  constitute this partition.

  Modified Design Units :
        TEST : "test/test_modify.sv", 11
```

Check if there is unexpected partition that is re-compiled, reasons?

Sharing the partition library

```
%cd test1
%vlogan -work dut_lib  dut.v
%vlogan -work test_lib   test1.sv -sverilog
%vcs top -partcomp  -partcomp_dir=GLOBAL_PCOMP_DIR
```

```
%cd test2
%vlogan -work test_lib  test2.sv -sverilog
%vcs top -partcomp  -partcomp_sharedlib=GLOBAL_PCOMP_DIR
   –partcomp_dir=test2_plib
```

Control Auto-partitioning

Use-model:

- -partcomp=autopart_high
  - Modules and packages with a high threshold
  - Results in larger and fewer partitions.

- -partcomp=autopart_low
  - Modules and Packages with a low threshold
  - Results in a smaller and more numerous partitions.

- -partcomp=autopartdbg
  - Creates the vcs_partition_config.file which contains the design partitioning information.
  - User can use this config file to add/delete any partitions based on user requirements .
  - Pass this config file at vcs to pick the modified partitioning.

- vcs –partcomp +optconfigfile+vcs_partition_config.file <..... >
  - Using +optconfigfile+<pcomp.cfg>

```
partition cell        module/Program;
partition instance    instance_name;
partition package     package_name;
```

Partition Compile Profiling

-***pcmakeprof*** : Enables profiling of time spent in each step of a compilation.

%vcs –partcomp -pcmakeprof

Time taken for each partition

Profile Output :

Activity	Real(s)	r(s)	Sys(s)
Global_analysis	1.20	0.15	0.10
Partition:_SNPS_VCS_intf_repository	0.94	0.27	0.15
Partition:_vcs_pc_package__6QYpze	1.00	0.27	0.17
Partition:TEST_LIB_env_pkg_GfheUb	10.19	8.62	0.40
Partition:DUT_LIB_dut_MfG2le	1.02	0.27	0.15
Partition:top_s9PCHd	1.49	0.76	0.19
Partition:TEST_LIB_envtop_IKIZrb	2.11	1.29	0.20
Stitching_&_Elaboration	1.52	0.44	0.22
All_partition_time	18.32	11.95	1.53
Total_time	22.00	12.56	1.88

3 step VCS-PC(Partition Compile) script (1) : Scratch

```
%>cat :run_pc_scratch
echo 'WORK > DEFAULT' > synopsys_sim.setup
echo 'DEFAULT : ./work' >> synopsys_sim.setup
echo 'LIB_VHDL : ./work.vhdl' >> synopsys_sim.setup

mkdir work.vhdl

vlogan \
          -kdb -lca \
          -full64 \
          -sverilog \
          -timescale=1ns/1ps \
          +define+LINK \
          -f lib_vcs.f -f rtl.f test.sv \
          -l ./vlog.log

vhdlan \
          -kdb -lca \
          -full64 \
          -f vhdl.f \
          -l ./vhdl.log \
          -work LIB_VHDL
(continue…)
```

```
(continue…)
vcs \
          -kdb -lca \
          -full64 \
          -reportstats \
          -l ./comp_1st_$1.log \
          -sverilog \
          -debug_access \
          -timescale=1ns/1ps \
          test \
          -partcomp \
          -pcmakeprof \
          -partcomp_dir=./scratch_lib \
          -Mdir=csrc.scratch \
          -o simv.scratch

# ./simv.scratch -reportstats -l ./sim_scratch.log
```

```
Execution)
%> :run_pc_scratch
```

* -Mdir : compile partition (default : csrc)

3 step VCS-PC(Partition Compile) script (2) : Recompile

```
%> cat :run_pc_recompile

echo 'LIB_$1 : ./work.$1' >> synopsys_sim.setup

vlogan \
          -kdb -lca \
          -full64 \
          -sverilog \
          -timescale=1ns/1ps \
          +define+LINK \
          test.sv \
          -l ./vlog_$1.log \
          -work work.$1

(continue…)
```

```
(continue…)
vcs \
          -kdb -lca \
          -full64 \
          -reportstats \
          -l ./comp_1st_$1.log \
          -sverilog \
          -debug_access \
          -timescale=1ns/1ps \
          -top work.$1.test \
          -partcomp \
          -pcmakeprof \
          -partcomp_sharedlib=./scratch_lib \
          -partcomp_dir=./recompile_$1_lib \
          -Mdir=csrc.recompile_$1 \
          -o simv.recompile_$1

# ./simv.recompile_$1 -reportstats -l ./sim_$1.log
```

```
(execution)
%> :run_pc_recompile sim_xxxxxxxxx
```

3 step VCS-PC(Partition Compile) run file example

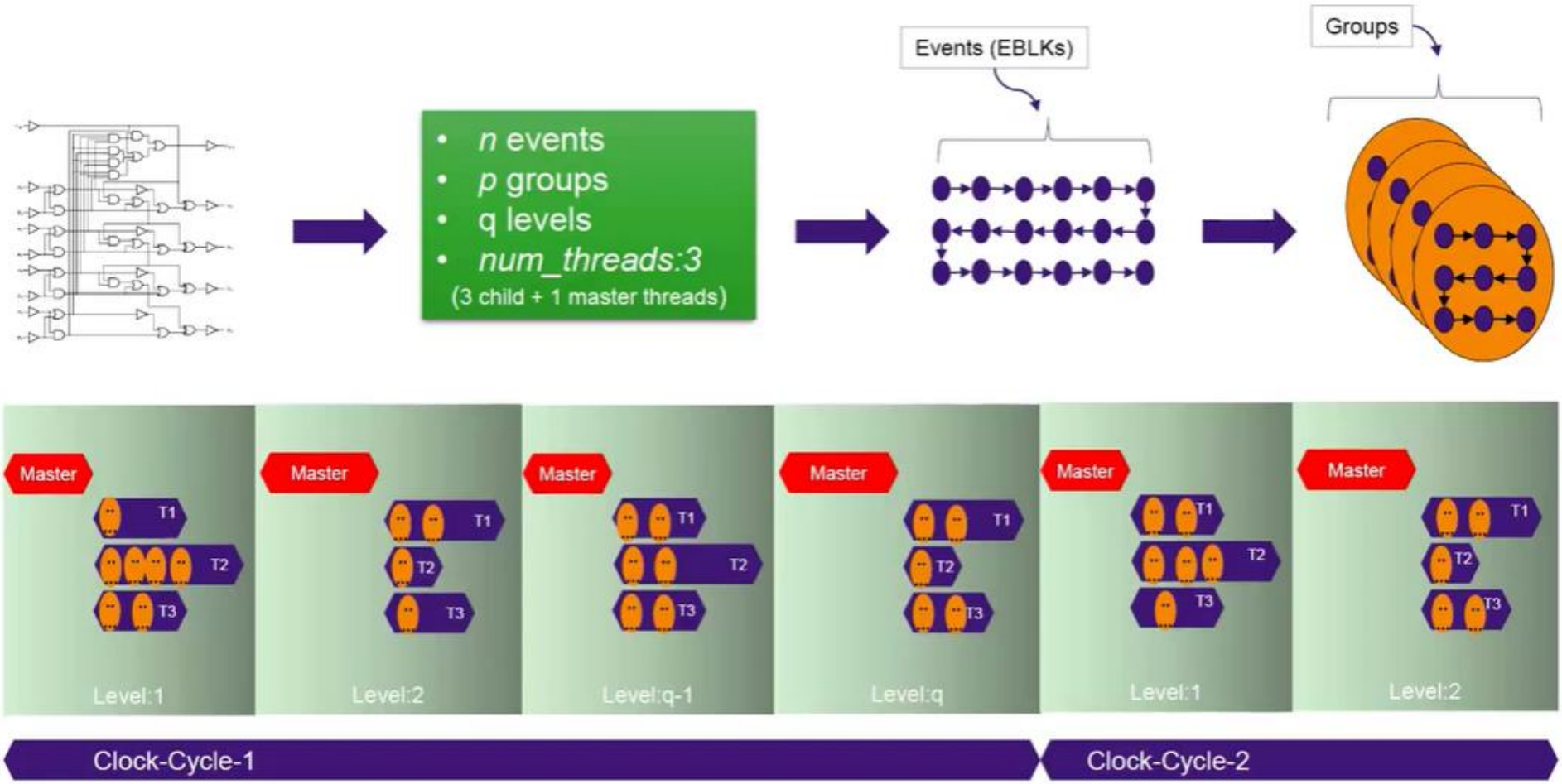
```
Usage example

%> cat run_vcs_pc

:run_pc_scratch
:run_pc_recompile testcase2 &
:run_pc_recompile testcase3 &
:run_pc_recompile testcase4 &
:run_pc_recompile testcase5 &
:run_pc_recompile testcase6
...
```

First scratch compile step generate shared lib

Recompile step can be run parallelly

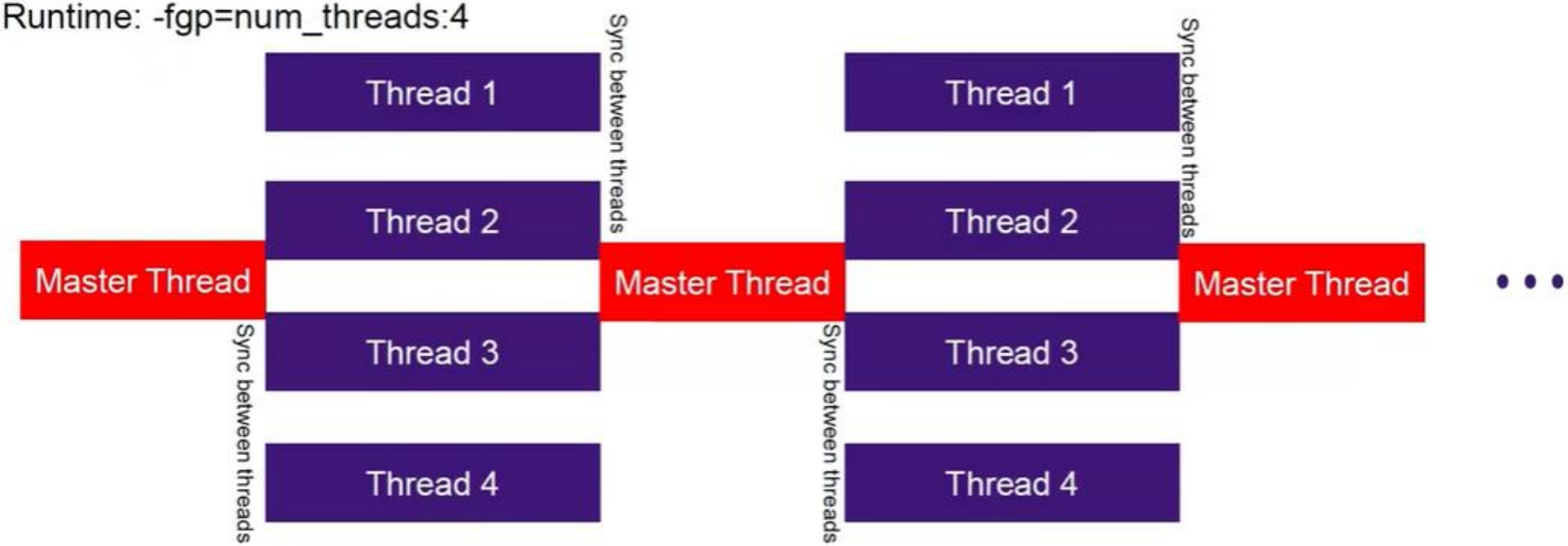
4. FGP (Fine Grained Parallelism)

FGP – Step by step flow



FGP(Fine Grained Parallelism) Basics

- Compile: -fgp
- Runtime: -fgp=num_threads:4



Master thread do all the serial/exclusive work. All other threads do parallel work

- CPU 코어 수 확인 방법
  cat /proc/cpuinfo
  CPU 당 물리 코어 개수
  grep “cpu cores” /proc/cpuinfo

VCS-FGP switch explanation

- Compile step
  - need only "-fgp"

- Simulation step
  - num_threads : CPU core number to use – 1
    - When use 8 CPU core - num_threads:7
    - When use 4 CPU core - num_threads:3
  - num_fsdb_threads : CPU core number to use for fsdb dump (default : 4)
  - sync:busywait : default, simulation performance optimize
  - auto_affinity :
    - If there is no enough CPU number after set num_threads, fgp is running within available CPU resource automatically (recommend).
    - cpu_affinity is OK when server is dedicated for simulation.
  - -fgp=diag:ruse :
    - Generation diagnostics report.

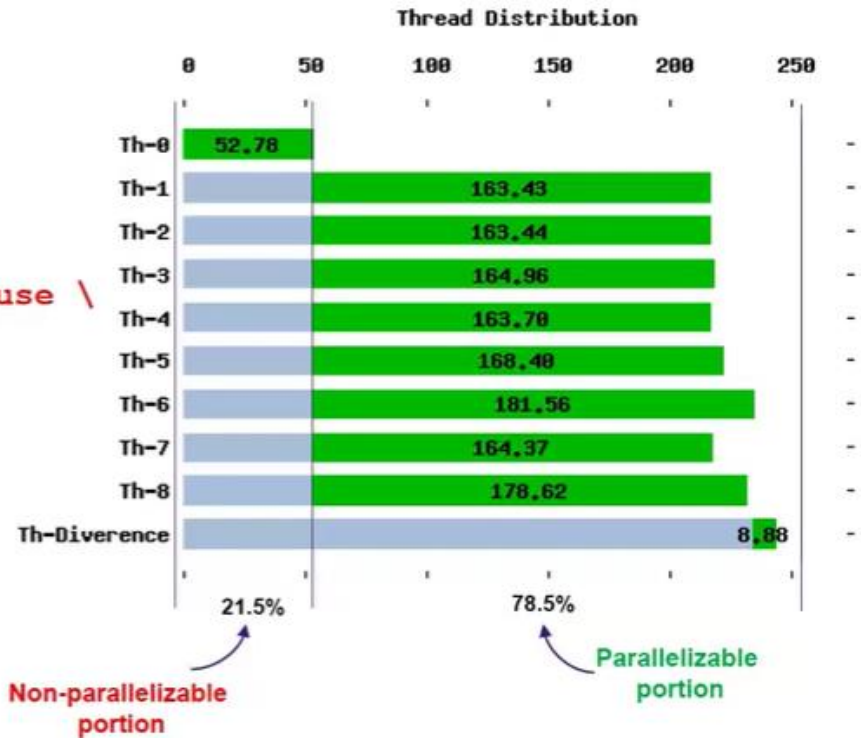
Performance Summary & Thread Distribution

```
vcs <other-options>
simv -Xdprof=timeline <other-options>
```

```
 events
EPC>=100000:          6324243508 (80.3%) // sim-cycles=38846
EPC[10000,100000):    1493170148 (18.9%) // sim-cycles=26885
EPC[1000,10000):        30494725 (0.4%) // sim-cycles=18903
EPC[100,1000):          28802824 (0.4%) // sim-cycles=55593
EPC[10,100):             3100350 (0.0%) // sim-cycles=77939
EPC<10:                   245876 (0.0%) // sim-cycles=80289
```

```
vcs -fgp -reportstats <other-options>
simv -reportstats -fgp=num_threads:7 -fgp=diag:ruse \
<other-options>
```

```
Simulation Performance Summary
==============================
Simulation started at :  Mon Feb 27 14:29:24 2017
Elapsed Time          :  245 sec
CPU Time              :  2178.0 sec
Virtual memory size   :  1455.0 MB
Resident set size     :  1173.9 MB
Shared memory size    :  1.5 MB
Private memory size   :  1172.4 MB
Major page faults     :  0
==============================
```



* EPC: Event Per Cycle
CPU Time/Elapsed Time = 2178/245 = 8.89

3 step VCS-FGP script

```
(File_name - :run_fgp)
echo 'WORK > DEFAULT' > synopsys_sim.setup
echo 'DEFAULT : ./work' >> synopsys_sim.setup
echo 'LIB_VHDL : ./work.vhdl' >> synopsys_sim.setup
mkdir work.vhdl

vlogan \
          -kdb -lca \
          -full64 \
          -sverilog \
          -timescale=1ns/1ps \
          +define+LINK_SPEED \
          -f lib_vcs.f test.sv \
          -l ./vlog_$1.log

vhdlan \
          -kdb -lca \
          -full64 \
          -f vhdl.f \
          -l ./vhdl_$1.log \
          -work LIB_VHDL
(continue…)
```

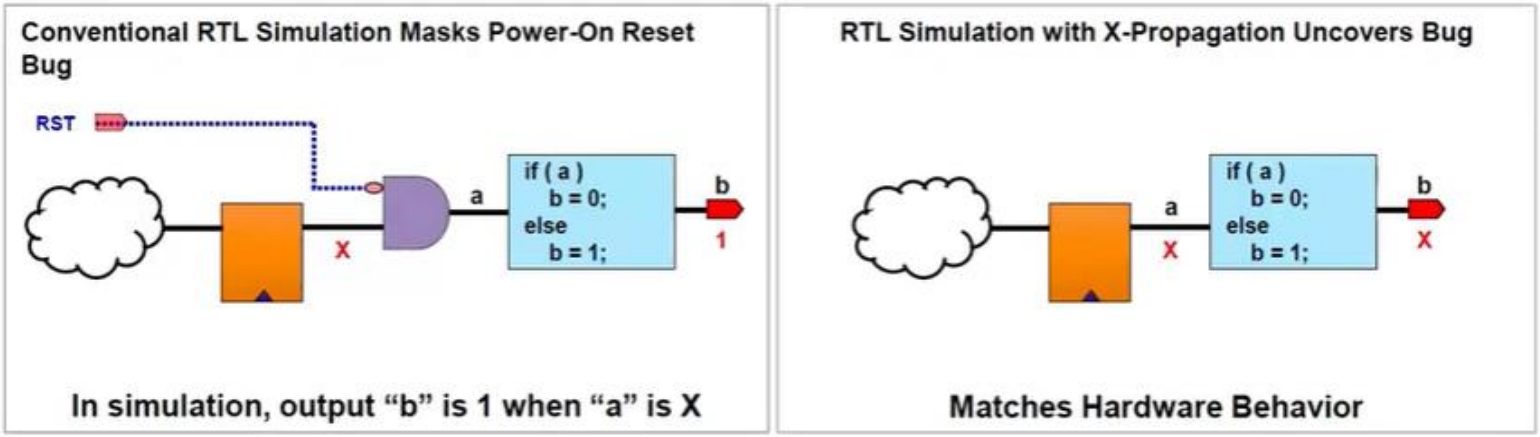
```
(continue…)
vcs \
          -fgp \
          -kdb -lca \
          -full64 \
          -reportstats \
          -l ./comp_$1.log \
          -sverilog \
          -debug_access \
          -timescale=1ns/1ps \
          test ref_design_stimuli

./simv    -reportstats -l ./sim_$1.log \
     -fgp=num_threads:7,sync:busywait,auto_affinity \
     -fgp=diag:ruse
```

```
Execution)

%> :run_fgp sim_cpuif_rw
```

5. X-prop

Verify X-Propagation Issues at RTL

Find X-related bugs at RTL simulation – 10X faster than gate level



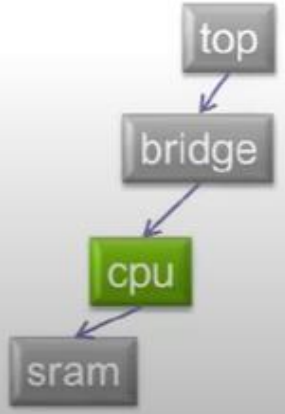
- Fits into existing RTL simulation flow
- Handles both combinational and sequential logic
- Switch to control degree of optimism/pessimism in simulation
- User-controllable scope

Compiling the design for X-propagation

- **`vcs -xprop ...`**
  - Entire design is instrumented for Xprop, using Tmerge (default)
- **`vcs -xprop=xprop.cfg ...`**
  - Xprop instrumentation is done adhering to config file
    - Default : no xprop instrumentation done
    - Order is important!

xprop.cfg

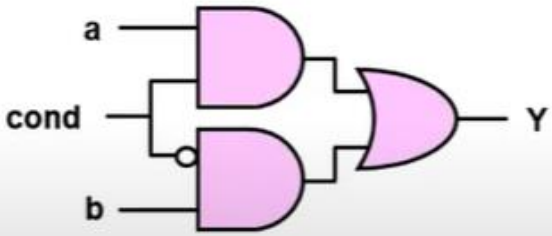
```
tree     { bridge }          { xpropOff } ;
instance { top.bridge.cpu } { xpropOn } ;
module   { sram, cache }     { xpropOff } ;
merge = tmerge;
```



Merge Function

- Several possibilities. Two are particularly interesting:
  - tmerge : Yields X when all values are different (like ternary op)
  - xmerge : Yields X always (unconditional)
    - More pessimistic

```
if( cond )
  Y = a;
else
  Y = b;
```



cond	a	b	T-merge	Y	X-merge
X	0	0	0	0	X
X	0	1	X	X	X
X	1	0	X	X	X
X	1	1	1	X	X

  - xmerge may better match gate-level simulation
    - Never more optimistic than any “gate” implementation

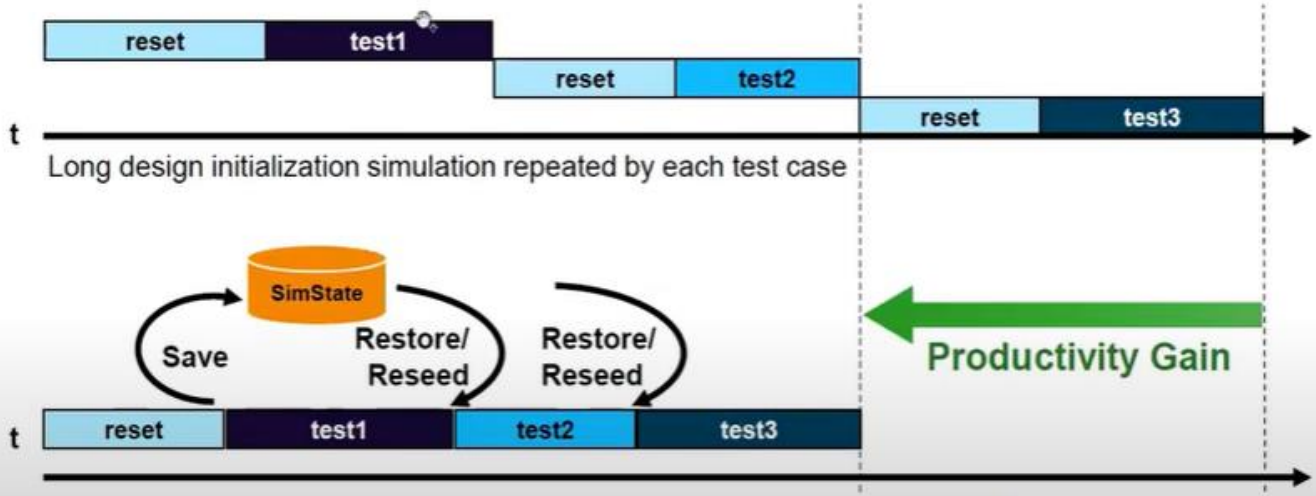
* xmerge preferred

6. Save & Restore

Save/Restore Common Reset Sequence

20-50% runtime improvement by eliminating redundant simulation cycles

- Save and Restore provides a mechanism to create a snapshot of a simulation at a specific point in time (Save) which can be replayed for subsequent simulations (Restore).



VCS 'save' command captures state at the end of first reset sim

Subsequent tests begin with 'restore' of previously saved state

Productivity gain depends on ratio of time spent in reset relative to average test time

Save and restore through UCLI command prompt

- Compile the design with any of the debug switches. It will work starting from -debug_access
- Load the simulations in UCLI and run till the desired time.
- Create a snapshot using the UCLI "save" command:

  %> ucli% save State_15
- Run and/or quit the simulations
- Relaunch the simulation in UCLI and restore the saved snapshot

  %> ucli% restore State_15

```
---------------test.v---------
module save_restore_ex1;
initial begin
#11 $display("one");
#10 $display("two");
#10 $display("three");
#10 $display("four");
end
endmodule
------------test.cmd----------
run 15      // stops at time15
save State_15
// saving snapshot at time 15
run 20
restore State_15
// restoring of saved snapshot
at time 15
run
quit
```

```
-----------------------------
Compile the above source file:
% vcs test.v -debug_access
Run the simulation:
% simv -ucli -i test.cmd
-----------------------------
ucli% run 15
one
15 s
ucli% save State_15
ucli% run 20
two
three
35 s
ucli% restore State_15
ucli% run
two
three
four
```

Using $save and $restart system tasks in source code

- Use $save system task to save the simulation till the desired time.
- Run and/or quit the simulation.
- Restore the saved snapshot using user-defined plusarg Options.
- NOTE: No need of using debug switches in this case

```
-----------test.v----------
module save_restore_ex2;
initial begin
#11 $display("one");
#10 $display("two");
#10 $save ("test.chk"); // saving snapshot at time 31
$display ("Saving simulation at %t", $time);
   $display("three");
#10 $display("four");
end
```

```
initial begin
if ($test$plusargs("save"))
begin
// restoring the saved
snapshot at time 31 using
user-defined plusarg options
$display ("Restoring saved
simulation");
$restart("test.chk");
end
end
endmodule
-----------------------------
Compile the above source
file:
% vcs test.v
Run the simulation:
% simv
```

```
VCS displays the following:
-----------------------------
     One
     two
     Saving simulation at 31
     three
$save: Creating test.chk from
current state of simv...
     four
-----------------------------
To restart the simulation from
the state saved in the check
file, enter the following
command:
% simv +save
VCS displays the following:
-----------------------------
Restoring saved simulation
Restart of a saved simulation
     four
```

Using $save system task in source code and passing saved check file at run time with the –r option

- Use $save system task to save the simulation till the desired time.
- Run and/or quit the simulation.
- Restore the saved snapshot using -r.
- NOTE: No need of using debug switches in this case.

```
-----------test.v-------------
module save_restore_ex3;
initial begin
#10 $display("one");
$save("test.chk");
// saving snapshot at time 10
$display ("Saving simulation
at %t", $time);
$display("two");
#0 $display("three");
#10 $display("four");
end
endmodule
-----------------------------
Compile the above source file:
% vcs test.v
Run the simulation:
% simv
```

```
VCS displays the following:
-----------------------------
one
Saving simulation at 10
two
save: Creating test.chk from
current state of simv...
three
four
-----------------------------
To restart the simulation from
the state saved in the check
file, enter the following
command:
% simv -r test.chk
VCS displays the following:
-----------------------------
Restart of a saved simulation

four
```