SYNOPSYS®

Synopsys Memory Compilers Verilog User Guide (APNT-25)

Memory Compilers

October 2020
Version 8.6

Copyright Notice and Proprietary Information

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

http://www.synopsys.com

Revision History

Version	Date	Note
1.0	May 2005	Initial release
2.0	May 2005	Revision release
3.0	September 2011	Major revision
4.0	March 2012	Added simulation with ikos model option
5.0	August 2012	Updated MEMFAULTINJ and verifault
6.0	October 2012	Update VIRAGE_IGNORE_RESET. Added Recovery check between read/write clock on 2PRF
7.0	June 2014	Minor updates
8.0	October 2015	Added ROM hex files related description
8.1	January 2017	Added functionality of enable_xfeature_ff flag in section 4 Added how to inject fault at a particular address and bit in section 6
8.2	September 2017	Added SNPS_FAST_SIM_FFV, SNPS_SIM_SPEEDUP,and SNPS_PON_RECONFIG_RESET to section Verilog Simulator Directives
8.3	March 2019	Added supported Synopsys emulator for IKOS model in section 2
8.4	February 2020	Modified IKOS note to remove reference of FPGA. Added note regarding TCSEP when running with timescale other than “1ns/1ps”
8.5	October 2020	Addition of message control options Re-ordered ‘Memory initialization’ section, added sub-headings and added table to initialize memory without init file
8.6	October 2020	Added missing *flt_type* argument to *task fault_inject* argument list

Table of Contents

1 Introduction

This Application Note describes the various Verilog models and Verilog simulator directives supported by Synopsys Memory Compilers. Different design stages require different Verilog models and ***+define+*** simulation arguments. The details of design stages and usage of these arguments is also described. The application note also describes the file setup for running Verilog simulations.

2 EDA Model Generation

Each Synopsys memory compiler library contains a template for each EDA view to be generated by the compiler. The Embed-It! Integrator software uses these templates to generate the corresponding Front-End or Back-End models. A number of Verilog models are thus generated. There are different stages in the chip design cycle and the generated models are of specific use at each stage when combined with ***+define+*** simulation arguments.

Synopsys' memory compilers generate following Verilog models:

<instance_name>_func.v	A switch-level Verilog netlist file. The same switch-level or transistor-level netlist that the compiler generates in SPICE format can also be generated in Verilog. The difference is that the Verilog netlist can be used for logic simulation and not for LVS. Please note one should not be using this model for chip level simulations. The simulation speed is also slowest with this model.
<instance_name>_fast_func.v	Verilog model with **limited functionality** to speed up the simulation time. This model supports only basic read/write operations LS, DS, SD and PIPEME functionality.
<instance_name>.v	Verilog model that can be used for behavioral, RTL and gate-netlist simulations.
<instance_name>.vhd	VHDL model that can be used for behavioral, RTL and gate-netlist simulations. For 90nm and below nodes Synopsys supports only VHDL RTL model;, behavioral model is not supported.
<instance_name>_stim.v	Verilog Test bench (Functional)
<instance_name>_verilog_vhd_stim.v	Verilog behavioral model and VHDL behavioral model Test bench. It is used for functional and hazard checks.
<instance_name>_ bus_wrapper.v	Verilog file used to test the basic functionality of the memory without consideration for BIST or ATPG. Test pins are tied off.
std_cells.v	Verilog model for the dummy standard cells used in Verilog netlist for ATPG.
<instance_name>_ikos.v	Synthesizable models used by Emulators. Synopsys Zebu, Mentor Velocity and Cadence Palladium use this view. This view models read/write functionality, BIST muxes, scan chain and power management pins. It does not model 'X' handling and timing. Redundancy can be also supported in this model by making "repair_in_ikos" to true in *.cfg or *custom.glb file. This is for 28nm and below nodes only.

<instance_name>_core.v	This file is used by Synthesizable RTL Model for IKOS. Memory core in the IKOS model will be a synchronous memory core if synch_mem_core=true. Setting synch_mem_core=false will result in asynchronous memory core and is the default behavior. This view has memory array and read/write operation defined.
<instance_name>_ase.hex	Available for ROM memories only. Contains anti-signature data for BIST verification. Calculated based on ROM content. ROM content is stored in <instance_name>.hex file. During simulation the user must ensure the proper <instance_name>_ase.hex and <instance_name>.hex are sourced into the simulator. The common mistake is to provide hex files from different ROM memories or generated with different content. <instance_name>.hex and <instance_name>_ase.hex should be generated from the same source, so these are associated to each other.
<instance_name>.hex	Represents ROM content. Anti-signature is generated based on this data. For successful BIST simulation *_ase.hex and *.hex files should be associated to each other which means taken for the same memory instance.

3 Differences between fast_func.v and behavioral models

BEHAVIORAL MODEL(*.v)	**FASTFUNC MODEL(*_*fast_func.v)***
This is the model with full functionality including scan chains and timing information	Verilog model with limited functionality to speed up the simulation time. This model supports only basic read/write operations, LS, DS, SD, and PIPEME functionality.
Initialize memory from a pre-loaded file (load_mem) and store memory contents into an array when redundancy option is used. Available for 28nm compilers.	No such initialization*

* For memories without redundancy, one can use $readmemh task to initialize using both behavioral or FASTFUNC model.

4 Limitations of fast_func.v model

1. Zero hold is not supported in FastFunc model. Input signals may not be properly captured when toggling at posedge of clock.
2. The fast_func.v (FFV) model by default supports timescale of “*1ns/1ps*”, which means that the precision value is in “ps”. In this model, "TCSEP" is used to decide the simultaneous CLK handling, and this value is defaulted to 0.001 (=1ns/1ps).

If simulations are required to be run with timescale other than “*1ns/1ps*”, please follow the below example.

If timescale is defined as "1ns/1fs", it would mean a precision of "fs" or "0.000001ns".
User would need to pass the same information to the simulation tool to apply for the FastFunc model.
This can be done by passing the following argument in the vcs simulation run command.
“-pvalue+TCSEP=0.000001”.

5 Additional functionality which can be included in fast_func.v model

(This can be selectively disabled/ brought-in by suitably defining bus wrapper and/or ifdef compile time options).

1. When bist_enable is set to TRUE, BIST MUXES automatically come in 45nm/40nm/28nm in fast func model. If user needs to use the test pins, then bus wrapper <instance_name>_bus_wrapper.v has to be generated by enabling “enable_define_wrapper” flag as TRUE in custom.glb.
   a. Additionally, for these pins to be functional -- `ifdef VIRAGE_SUPPORT_TESTPINS_FFV – compile option has to be enabled.

2. Reconfig register pins - for only load-shift-reset operation without repair -- (all ST instances).
   a. These pins are functional with a Verilog bus wrapper generated using flag defined in custom.glb **enable_define_wrapper** = true else tied to 0 (disabled) during run time.
   b. Additionally, for these pins to be functional -- `ifdef VIRAGE_SUPPORT_RED_FFV – compile option has to be enabled.
   c. For all ST instances except 8M/16M – additionally, `ifdef VIRAGE_SUPPORT_TESTPINS_FFV – compile option has to be enabled.

3. When verilog bus wrapper is generated with flag enable_define_wrapper = false, all redundancy pins and test pins are tied to 0 (disabled) in ST/IT instances.

4. When enable_xfeature_ff flag is set to TRUE, power management pins and PIPEME modeling are included in fast_func models. If this flag is set to FALSE, power management pins and PIPEME modeling are excluded in fast_func models.

6 Features common in fast_func.v and behavioral (.v) model

a. Basic read/write operation, X-handling of input pins & exceptional handling & corruption due to invalid sequencing of BS pins (8M)

b. Power management pins functionality and SO* clamping in Deep Sleep/Shut Down mode (**28nm only**)

7 Verilog Simulator Directives

The following simulator directives are supported by Verilog models. These directives provide flexibility and extra features to the customers and can be called by the simulator command line argument ***+define+*** or ***-define***.

-define is used with NC Sim tools from Cadence Design systems.

+define+ is with used with Mentor Graphics and Synopsys simulation tools.

- Message control options

• ***MEM_CHECK_OFF*** - This flag turns off most display messages generated by the Verilog model. These display messages are mostly warnings about hazardous conditions.

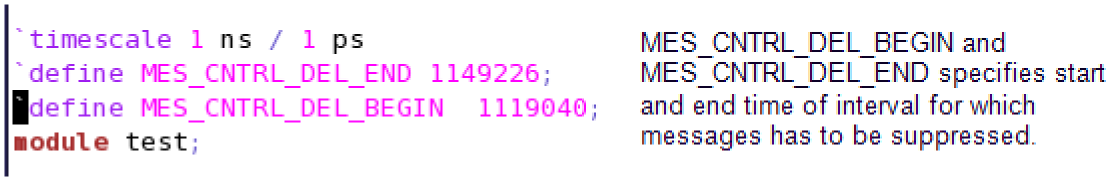
Command line argument: ***+define+MEM_CHECK_OFF***

<u>Delay based Message Control:</u>

Using this method, messages can be suppressed during a specified time.

a. Time limits are defined using MES_CNTRL_DEL_BEGIN and MES_CNTRL_DEL_END define directives. Default values of these are “0”. If these directives are not defined then it takes them as “0”. For example, if messages are not required during initial 100ns, then only MES_CNTRL_DEL_END needs to be defined as MES_CNTRL_DEL_BEGIN would be set to “0” by default.

Define directives should be defined in topmost level in hierarchy of design as shown below.



```
`timescale 1 ns / 1 ps
`define MES_CNTRL_DEL_END 1149226;
`define MES_CNTRL_DEL_BEGIN  1119040;
module test;
```

MES_CNTRL_DEL_BEGIN and MES_CNTRL_DEL_END specifies start and end time of interval for which messages has to be suppressed.

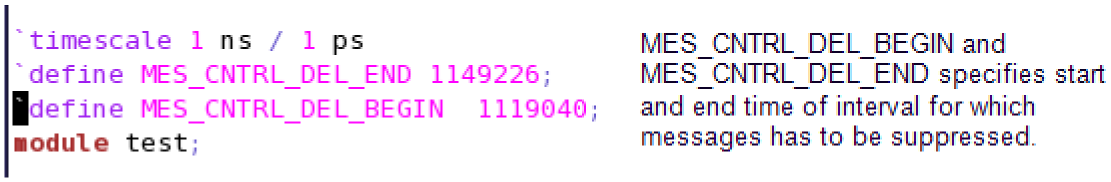
Messages are suppressed whenever simulation time lies in interval specified by “MES_CNTRL_DEL_BEGIN” and “MES_CNTRL_DEL_END”.

b. Time limits can be specified using parameters “MesCntrl_Begin ” and “MesCntrl_End”. Default values of these are “0”. Implementation would be similar to that described above (a). Parameter can be passed through command line as shown below:

“-pvalue+test.top_behav.MesCntrl_End=1149226”

Pin/Value based Message Control:

Using this method, messages can be suppressed based on a value on a particular Input/Output pin. Please note that this method will be applicable only when both MES_CNTRL_DEL_BEGIN and MES_CNTRL_DEL_END, are either not defined or defined “0” and both “MesCntrl_Begin” & “MesCntrl_End” are “0”. .
Pin and its value are defined using “MES_CNTRL_PIN” and “MES_CNTRL_PIN_VAL” define directives.
Define directives should be defined in topmost level in hierarchy of design.



```
`timescale 1 ns / 1 ps
`define MES_CNTRL_DEL_END 1149226;
`define MES_CNTRL_DEL_BEGIN  1119040;
module test;
```

MES_CNTRL_DEL_BEGIN and MES_CNTRL_DEL_END specifies start and end time of interval for which messages has to be suppressed.

Note: User has to take care while defining these directives that Pin name should exactly match the name present in model at that level, and should be appended by suffix “_buf”. Also, pin value should be defined with pin dimension in absolute format as shown in example; otherwise will not serve the desired purpose. Set of values supported for MES_CNTRL_PIN_VAL are “0/1/X”.

Available to: behavioral and fast_func.v models

- Other options

• ***VIRAGE_FAST_VERILOG*** - Usage of this flag will make the model run in RTL mode and no timing information will be present.

Command line argument: ***+define+VIRAGE_FAST_VERILOG***

Available to: behavioral model

• ***MEMFAULTINJ*** - This macro enables the fault injection mechanism if this is available in memory model with the corresponding run-time simulation arguments.

Command line argument should be used: ***+define+MEMFAULTINJ +argument***

The following arguments are available:

o ***star_error_fail*** – this option comes with in STAR memory. If this option is used, faults will be more than repair capability of the memory, i.e. injects non-repairable faults into memory.

o ***star_error*** - this option comes with in STAR memory. The fault injected with this option are repairable with existing redundant columns and or rows, i.e. injects repairable faults into memory.

o ***asap_error*** - to inject faults in ASAP (non-STAR) memory and test it, this argument injects a single fault into memory.

Available to: behavioral model

- To inject fault on a particular address, you can follow the instructions below. Below task would enable user to inject fault at given address and bit:
    - Logical address
    - Logical bit
    - Type of fault (0 or1)

  Support Fault Injection through a Task:

  The task defined in the Verilog Behavioral model can be accessed hierarchically from the testbench or from any higher level module. This task can be used only when "MEMFAULTINJ" is not defined. With the help of it, user can inject fault in a specific memory. At the time of task call, user will pass the address and bit location at which the user wants to inject the fault.
  Please note that for injecting faults into multiple locations, task needs to be called multiple times.

```
`ifndef MEMFAULTINJ
task fault_inject;         // User will access this task hierarchically to inject fault at
                                location of their choice
        input [5:0] address;    // Logical Address (decimal/binary/hex) at which user
                                        wants to inject the fault
        input [3:0] bit_pos;    // Bit in the word at above location which user wants to
                                        define faulty
        input flt_type;         // Type of stuck-at fault – 0 or 1
        begin
        /**** Task Body, it will enable fault injection at user defined location ****/
        end
endtask
`endif
```

  This task can be called in the top level module/testbench as shown below:
  <memory_instantiation_name>.fault_inject(10,4,1);

- ***VIRAGE_IGNORE_RESET*** - This flag ignores the initialization on RSCRST required by the memory verilog model when bist_enable = true and redundancy_enable = true.

  Command line argument: ***+define+ VIRAGE_IGNORE_RESET***

  Available to: behavioral model

- ***VIRAGE_IGNORE_SAME_ADDR_MSG*** – this flag turns off the warning on concurrent read and write operation on the memory same address. The violation looks:

  <<Simultaneous access to the same address on B-port(READ) and A-port(WRITE)>>

  Command line argument: ***+define+VIRAGE_IGNORE_SAME_ADDR_MSG***

  Available to: behavioral model

- ***virage*_ignore_*read_addx*** - This flag, if used, will make sure that the memory is not corrupted during an unknown address read cycle. Usually when the address is unknown during a read or a write cycle, memory gets corrupted.
  If your design is such that you are 100% confident that setup and hold time is not violated - i.e., even though Address is X, this only means that it is either 0 or 1 but you

do not know which, and due to the latching within your design,the address is stable near the clock edge and meets setup and hold times – then you may take a less-pessimistic approach to modeling the memory.

Command line argument: ***+define+virage_ignore_read_addx***

Available to: behavioral model

- ***Verifault -*** This switch is used to control the fault injection on memory model for ports. Memory models are modeled in behavioral description, so it is not possible to inject faults on lot of logic portion. This option allows user to inject faults only for ports.

  Command line argument: ***+define+verifault***

  Available to: behavioral model

- **VIRAGE_IGNORE_SD_HAZARD** - This define argument controls the corruption of outputs when SD transitions during DS mode in Verilog model and FFV.
  This argument will also control the tests in the testbench. Default behavior is the new behavior wherein outputs are corrupted when SD transitions during DS mode. If DS goes back to '0' while SD is still '1', outputs continue to be 'X'. If SD transitions back to '0' when DS=0, outputs will be '0'(only Q;QP, SO* & RSCOUT continue to be 'X').
  If define argument is provided, then models will have old behavior. However following models/sections will not be affected by this define argument and will continue to have new behavior only.

  - Verilog model – Recovery between negedge of DS and posedge SD(Tfdsc); notifier block corrupting outputs in case of violation.
  - Liberty Model – Recovery Tfdsc(same as above);
  - Liberty Model – Leakage condition changes (conditions modified as per new requirement);
  - Datasheet Model – Truth table changes for SD mode hazard;
  - Datasheet Model – Recovery time Tfdsc for the recovery arc between negedge of DS and posedge SD.
  - VHDL RTL model – will have only new behavior.

  Note: VIRAGE_IGNORE_SD_HAZARD is only applicable for 45/40nm compilers.

  Available to: Verilog behavioral and FastFunc model

- To use <instance_name>.ikos.v and <instance_name>_core.v for simulation using <instance_name>_stim.v with <instance_name>.v as reference model, user needs to include command line option +define+synthrtlmodel".

- **SNPS_FAST_SIM_FFV** in fast func model: On defining, it will increase simulation performance by masking Hazards & power down assertion messages.

- **SNPS_SIM_SPEEDUP** is meant for parameterised std_cells and scan chains in Verilog model. This define macro should not be used for parallel ATPG simulation. It it is defined, parallel ATPG simulation will not run.

- **SNPS_PON_RECONFIG_RESET** with define macro **VIRAGE_IGNORE_RESET** is to reset the reconfig register when reconfig register supply set VDDF=1.

NOTE: VIRAGE_FAST_VERILOG, VIRAGE_IGNORE_RESET, virage_ignore_read_addx are not recommended for final simulation.

8 Design Cycle Stages

The stages of the design cycle can be categorized as follows:

a) **Memory and surrounding design rough estimation** –In this mode basic read/write operations, X-handling and exceptional handling, pipe-flop functionality for output with one clock latency can be checked. This model is mainly used to help customers speed up their simulations during initial phase of design. ***<instance_name>_fast_func.v*** is used in this flow steps.

+define+ arguments, which can be relevant under this mode are:

+define+MEM_CHECK_OFF

+define+VIRAGE_SUPPORT_TESTPINS_FFV

+define+virage_ignore_read_addx

b) **Functional mode only** – In this model, the SoC designer is concerned about exact behavior of the memory with all the diagnostic warning messages and hazard conditions. The memory behavioral model (***<instance_name>.v***) can be also used for timing simulations with SDF back annotated after synthesis.

<instance_name>_bus_wrapper.v should be used during this mode. The bus_wrapper is used in this case as during the functional simulations there is no need to handle test pins and that is why the memory behavioral model is easier to use with bus wrapper which wrapping the memory ties all test signals to safe values and propagates all functional signals to the top for user control.

User has to use ***+define+VIRAGE_IGNORE_RESET***. Other ***+define+*** arguments, which can be relevant under this mode are:

+define+MEM_CHECK_OFF

+define+VIRAGE_FAST_VERILOG

+define+virage_ignore_read_addx

c) **SMS/BIST mode** - In this mode, the memory is used in test mode and is supposed to be controlled by BIST logic, like Synopsys SMS BIST solution. It is important to have memory exact behavioral model to be able to simulate the memory taking into account hazard conditions, timing and all available display messages from the memory, The memory behavioral model ***<instance_name>.v*** should be used.

The +define+ arguments relevant in this mode are:

+define+MEM_CHECK_OFF

+define+VIRAGE_FAST_VERILOG

+define+MEMFAULTINJ

+define+virage_ignore_read_addx

d) **ATPG mode** - It is assumed that the BIST and/or functional RTL verification was already performed and SoC designer is dealing with the scan inserted gate-netlist to perform ATPG tool test-bench simulations. This will also require memory full modeling which includes timing, hazard conditions and display messages. The model to be used is (***<instance_name>.v).***

The +define+ arguments relevant in this mode are:

+define+MEM_CHECK_OFF

+define+virage_ignore_read_addx

+define+MEMFAULTINJ

To summarize the above four points, user needs to note that for the b, c, and d modes the memory behavioral model ***<instance_name>.v*** is used.

9 Procedures for Initializing memories

9.1 Memory Initialization in LT & IT memories:

For the memory instances configured with *redundancy_enable*=0, user can directly read in a file into memory array by loading directly from the testbench or the behavioral model by using predefined “$readmemh” Verilog system task and using memory array through hierarchy. The syntax of the “$readmemh” Verilog system task is described below:

```
$readmemh(“File_name_to_load", inst0.u0.mem_core_array);
```

Where, *inst0* is the instance name of the memory in customer's netlist and *File_name_to_load* is the name of the file to initialize memory.

For Verilog behavioral model, modifications can be done through the following process by adding the INITIALIZE_MEM define argument to the model (this argument may be present for some memory compilers’ generated behavioral models).

```
`ifdef INITIALIZE_MEM
    initial begin
            $readmemh("preload.dat", <instance_name>.u0.mem_core_array);
    end
`endif
```

9.2 Memory Initialization Capability in LR & ST memories:

For the memory instances configured with *redundancy_enable*=1, the memory cannot be loaded directly from a data hex file using the predefined “$readmemh” Verilog system task. (This is because of the additional rows and columns, owing to redundancy.)

9.2.1 Initialize Memory using an external file

User may use the “load_mem” task defined in the behavioral model. The task “load_mem” loads the file defined by ‘parameter PreloadFilename = “init.file”’, into the memory array. Use “+define+INITIALIZE_MEM” to enable this feature.

9.2.2 Initialize memory without using any external file

Use define argument “INITIALIZE_MEM” and $plusarg option “init_mem_data” to enable this feature.

Initialize entire memory with all ‘0’, all ‘1’ or with a specific data. If user does not specify any value, then memory will be initialized with data = address. Table below shows the different possibilities.

Initialize entire memory with	**$plusargs option**	**Remark**
All 0	%simv +init_mem_data=0	Initialize all memory locations with “0”.
All 1	%simv +init_mem_data=1	Initialize all the bits in memory with “1”. E.g. if there are 8 bits in a word, then content of each word in memory will be initialized with data “FF”.
User specified data	%simv+init_mem_data=<num> e.g. +init_mem_data=12 Note: 12 is a decimal number.	Each memory location will have data “12”.
No option chosen by user	%simv +init_mem_data	Each memory location will have data = address; mem_array[adr] = “adr”

10 Running Verilog Simulations

For successfully running Verilog simulations the end user must be familiar with the location of the Verilog files, for example:

- The location of the Functional Verilog model for the particular memory instance. The naming convention for this file is ***<instance_name>_func.v***, where ***<instance_name>*** is the name of the instance generated.

  An example path to this file is:

  ../<work>/compout/views/<instance_name>/<instance name>_func.v

  Where ***<work>*** is the directory in which the compiler instance generation was run.

- The location of the Behavioral Verilog model for the particular memory instance. The naming convention for this file is ***<instance_name>.v***.

  An example path to this file is:

  ../<work>/compout/views/<instance_name>/<pvtcorner>/<instance_name>.v

  Where ***<pvtcorner>*** is the PVT corner of interest (for example, Best, Typical, Worst).

- The location of the Verilog Test Bench model for the particular memory instance. The naming convention for this file is ***<instance_name>_stim.v***.

  An example path to this file is:

../<work>/compout/views/<instance_name>/<pvtcorner>/<instance_name>_stim.v

- The location of the standard cells for the particular compiler. Std_cells.v is located under the compiler directory. Std_cells.v is also generated under the instance generation directory, i.e., under the "

  ../<work>/compout/views/<instance_name>/std_cells.v

- The location of the Verilog primitives for the particular compiler you are using. The naming convention for this file is ***<compiler_name>.v*** where ***<compiler_name>*** is the compiler name without the version numerals. For example, the Verilog primitive for the ***ts40npk42p22sadsl512sa05*** compiler is ***ts40npk42p22sadsl512sa.v*** (note the missing ***05*** suffix). An example path to this file is:

  ../<complib>/<compiler>/<compiler_name>.v

  Where ***<complib>*** is the directory, typically under system administrator control, where the compilers are installed, and ***<compiler>*** is the compiler name including the numeric suffix.

Setting up Files for Simulation

To run a Verilog simulation, the user will need to make the following edits to the Test Bench model (***<instance_name>_stim.v*** file) as generated by the compiler:

i. Remove the suffix _***behav*** from the instance name only. Do not remove the suffix _***behav*** from the ***top_behav*** instantiated name. Listed below is a typical example that shows what to change.

   sadslspk42p128x64m4b1w0c1p0d0r1_behav top_behav (.QA(QA_BEHAV), .QB(QB_BEHAV), .VDD(VDD), .VSS(VSS), .ADRA(ADRA), .DA(DA), .WEA(WEA), .MEA(MEA), .CLKA(CLKA), .TEST1A(TEST1A), .RMEA(RMEA), .RMA(RMA), .LS(LS), .ADRB(ADRB), .DB(DB), .WEB(WEB), .MEB(MEB), .CLKB(CLKB), .TEST1B(TEST1B), .RMEB(RMEB), .RMB(RMB));

   Change it to:

   sadslspk42p128x64m4b1w0c1p0d0r1 top_behav (.QA(QA_BEHAV), .QB(QB_BEHAV), .VDD(VDD), .VSS(VSS), .ADRA(ADRA), .DA(DA), .WEA(WEA), .MEA(MEA), .CLKA(CLKA), .TEST1A(TEST1A), .RMEA(RMEA), .RMA(RMA), .LS(LS), .ADRB(ADRB), .DB(DB), .WEB(WEB), .MEB(MEB), .CLKB(CLKB), .TEST1B(TEST1B), .RMEB(RMEB), .RMB(RMB));

   The instance name in the example above is ***sadslspk42p128x64m4b1w0c1p0d0r1.***

ii. Remove the comment character on the lines that contain ***dumpfile*** and ***dumpvars.***. The comment character is a set of double forward slash ***//***.

Running the Simulation

The following command can be used to run VCS simulations. Please note that hard coded paths are shown in this example. Relative paths may also be used.

vcs –debug_all +v2k –R \

<directory path to Verilog Test Bench Model>/<instance_name>_stim.v \

<directory path to Behavioral Verilog model>/<instance_name>.v

11 Running a Co-simulation between Verilog Behavioral and Functional Models

Use the following procedure to run a sanity check between the Verilog Behavioral and Functional models. Please note that this may take more time when compared to running simulations on standalone behavioral model.

Setting up Files for Simulation

The user will need to make the following edits to the Behavioral (***<instance_name>.v*** file) and Test Bench (***<instance_name>_stim.v*** file) models as generated by the compiler:

a) Edit the Behavioral Verilog module file by adding the suffix ***_behav*** to the module name. The module name can be found on the line that starts with ***module***. The name following the construct ***module*** is the instance name, which is followed by a list of inputs, outputs, and bi-directional pins.

b) Edit the Verilog stimulus file by removing the comment characters on the lines that contain ***dumpfile*** and ***dumpvars.*** The comment character is a double forward slash **//**.

NOTE: Do not make the changes to the stimulus file described in Step i) of the previous section.

Running the Simulation

VCS command line:

***vcs –debug_all +v2k –R ***

***/<directory path to Verilog Test Bench Model>/<instance_name>_stim.v ***

***/<directory path to Functional Verilog Model>/<instance_name>_func.v ***

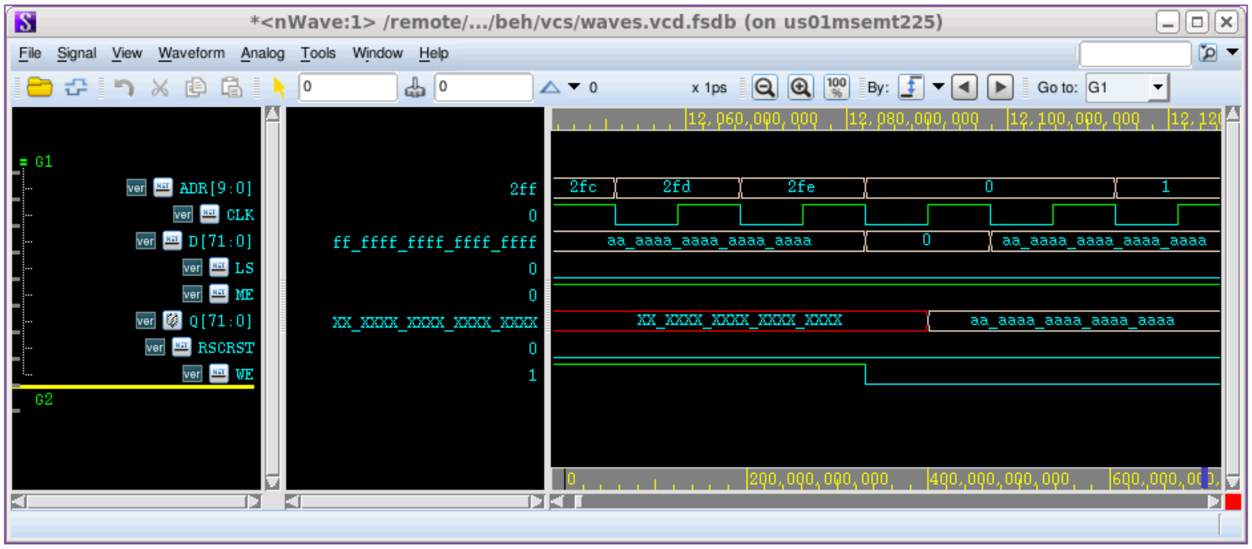
***/<directory path to Behavioral Verilog Model>/<instance_name>.v ***

/< path to the project directory>/std_cells.v \ /<path to the memory compiler directory>/<compiler_name>.v

12 Viewing Simulation Waveforms

Running the Verilog simulation generates a ***waves.vcd*** signal waveforms directory which can be viewed with ***nWave/Simvision*** and other compatible waveform viewers.

Following is a sample **nWave** screenshot showing signal waveforms.



13 Recovery check on Two Port Register File

The delay requirement between the write clock and the read clock is modeled as recovery check. If read and write clocks of the 2-port Register File memory are from the same source at the chip level, the following set_multicycle_path should be used. In this example, CKA is read clock and CKB is write clock.

```
set_multicycle_path -setup 0 -from [get_clocks CKA] -to [get_clocks CKB]
```