CYPRESS SEMICONDUCTOR
198 Champion Court
San Jose, CA 95134
Tel: 1-408-943 2600
Web: www.cypress.com

CYPRESS Model Manual

CYPRESS Model Manual

Version *C

Author: CYPRESS **Department:** Hardware Systems
SEMICONDUCTOR Engineering

Project: CYPRESS Model Manual

Document Name: Cypress Model Manual.docx

Last Changed:06/19/20

Distribution: General Distribution

Preface: This document covers Verilog and VHDL model simulation package

This document contains CYPRESS's technical specifications regarding the products described herein. This document may be revised by subsequent versions or modifications due to changes in technical specifications.

Table of Contents

1 Document History

Version/Date	Modification
**: 01/23/04	Initial version
*A: 05/20/15	Changed document template from Spansion to Cypress. Also, updated the contents for new files added in the package.
*B: 09/16/2015	Included information to assign timing model
*C: 11/05/2019	Updated for Cypress Semper Ultra

2 Overview

The self-extracting archive contains model files for Cypress Flash Memory (Figure 1: Downloadable Package Contents).

These models support:

- High/Low or Top/Bottom boot option (model dependent, see data sheet)
- Verilog / VHDL behavioral simulation
- Timing-accurate simulation (including support for interconnect path delays)
- Built-in Timing checks
- Pre-loading each instance of the model in the top-level net list with
    - Protection Mode for all sectors (if applicable)
    - Contents of main memory array
    - Contents of Secure Silicon (if applicable)
    - Contents for Top/Bottom boot sectors (if applicable)

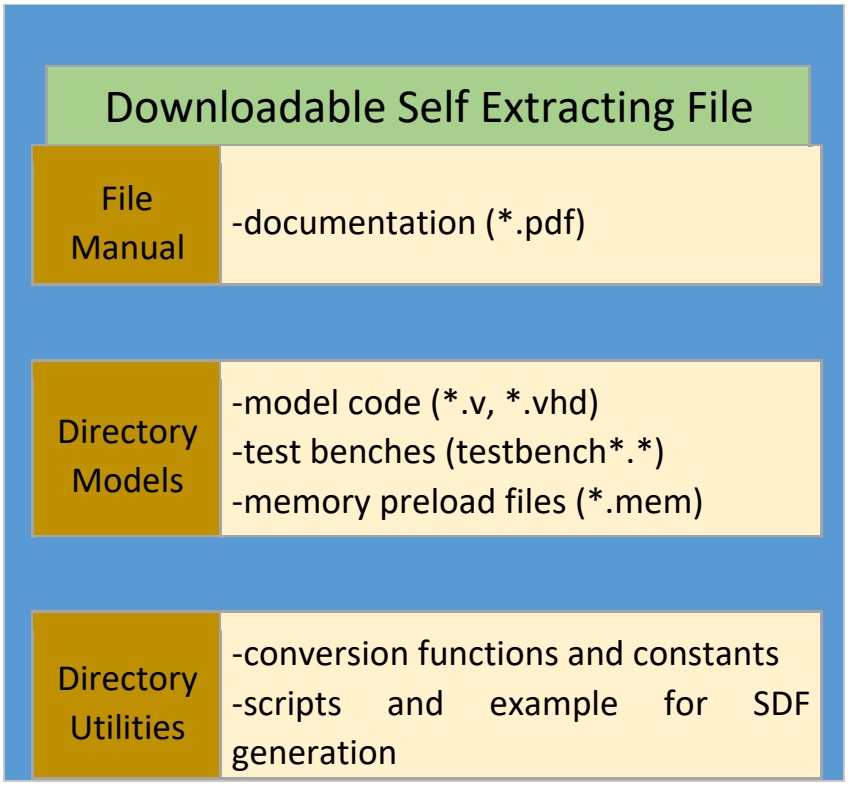

Figure 1: Downloadable Package Contents

The package contains all the necessary preload files, test benches and timing files to get started in your simulation environment.

3 VHDL/Verilog – Model

3.1 Required Files

3.1.1 Files for a Simple VHDL Behavioral Simulation

The model file model.vhd is located in the model directory. It relies on a set of functions defined in the files gen_utils.vhd and conversions.vhd. These 3 files comprise the minimum set of files for a behavioral VHDL simulation.

3.1.2 Files for a Simple System Verilog Behavioral Simulation

The model file model.sv and testbench.sv are located in the model directory. It does not depend on other files and can be run as is in a behavioral System Verilog simulation.

3.1.3 Pre-loading Flash Memory Model in Behavioral Simulations

In order to reduce simulation time and simplify the simulation, each instance of the model can be pre-loaded, i.e. the simulation starts up as if the memory had been programmed before and enters the simulation in a certain mode and contains certain data. The pre-loading feature is controlled by 2 attributes

- UserPreload
- preload_file_name(s)

These attributes should be assigned to each instantiation of the model in the top level net list.
If these attributes are not assigned, the model will default to not use pre-loading.

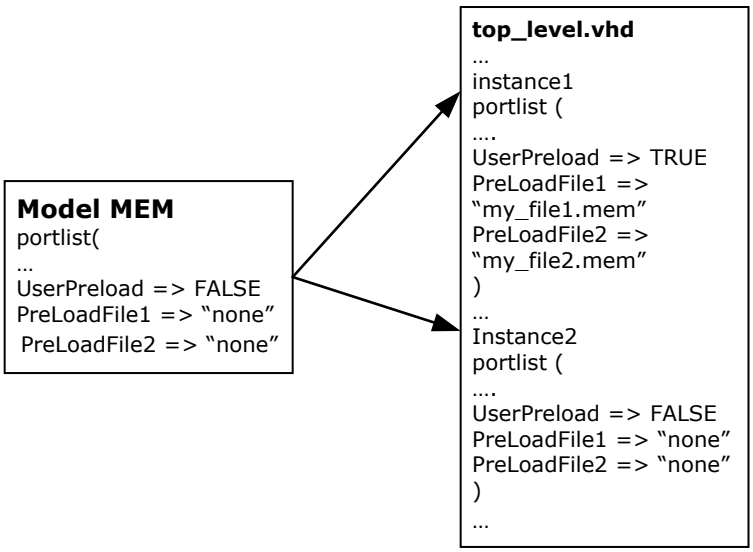

Figure 2: Instantiations of models with different pre-load configurations

3.1.4 Writing Pre-load Files

The test bench process loads default values into memory array, CFI field and protection bits. The Memory array is initialized to all 0xFFFF, the CFI space is preloaded according to the data sheet. Memory Array is unprotected by default. The values can be overwritten by defining pre-load files for the model instantiation in the test bench:

mem_file_name => loads memory array with data prot_file_name
=> lists protected memory blocks of array
Secsi_file_name => loads Secure Silicon Sector (if available)

Your device may not support all the types of preload files. See your datasheet to ensure you have a Secure Silicon Region and a protection method that allows protection of all the sectors individually

**** NOTE: *Preload files should not have empty lines* ****
NOTE: *Provide leading zeros for addresses*

Example for memory/secsi pre-load file

```
// select Addr0: ADDR_0= AA, ADDR_1= 55, ADDR_2= 11
// select AddrA8: ADDR_8= 01, ADDR_9= 02, ADDR_10= 03
@00000
AA
55
11
@000A8
01
02
03
```

Example for protected memory block array pre-load file

```
// select memory block 01, 19, EE: set 1 to protect
@01
1
@19
1
@EE
1
```

3.2 Compiling Model Files

3.2.1 Compilation of VHDL Model files

The following list shows which files need to be compiled to which library:

- conversions.vhd : compile to library FMF
- gen_utils.vhd: compile to library FMF
- model.vhd: compile to library work

The file model.vhd depends on conversion.vhd and gen_utils.vhd and therefore needs to be compiled only after these two files.

3.2.2 Compilation of Verilog Model Files

The files model.sv and testbench.sv can be compiled as is. No further libraries need to be provided.

3.3 Simulating Model Files

3.3.1 Basic and Timing Simulation

For a basic simulation no files beyond the files used for compilation are required.

For performing timing simulation, the timing information needs to be provided as an SDF file. This allows for a complete back-annotation including interconnect path delays (e.g. for high speed boards) on system level. This approach is identical to typical ASIC simulation flows.

The model must be configured user must assign "TimingModel" variable with appropriate OPN code listed in the included *.ftm files. Verilog model simulation must include the OPN code from the model_ver.ftm file and VHDL simulation must include the OPN code from the model_vhd.ftm file.

NOTE: Not updating the "TimingModel" variable appropriately will the correct OPN number will lead to wrong or unexpected simulation results.

3.3.2 Generating SDF files

The SDF information for all speed grades of a model is provided in the FTM file. The FTM files for the Verilog and VHDL version of the model are located in the model directory. From the FTM file, an SDF file can be generated by 2 different methods:

Method 1

Select the section with the desired speed grade (i.e. the OPN) in the FTM file. Copy and paste this section into the final SDF file for the overall simulation.

Method 2

For VHDL-only simulations a global SDF file can be generated automatically if an FTM files exist for each component of the overall simulation environment. The global SDF file can be created by executing the Perl script provided in the utilities directory. The script parses the top-level netlist for instances of components. Each instance has a timing attribute (TimingModel) with a value that is equivalent to its OPN in the data sheet (Figure 3: Instantiations of models with different speed grades).

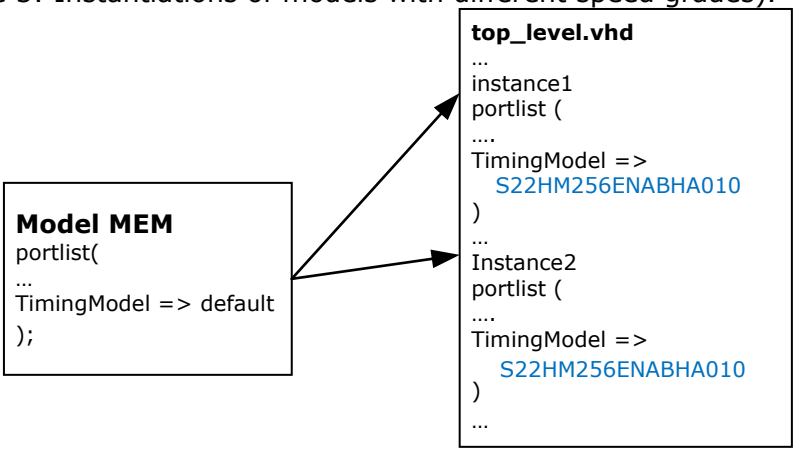

Figure 3: Instantiations of models with different speed grades

Based on these attributes the Perl script selects the appropriate timing information from the FTM files for each component and assembles the overall SDF file. The Perl script is executed by entering:

```
perl mk_sdf_204.pl top_level.vhd // Unix environment
```

Note that this script is applicable to VHDL-only simulations (no mixed Verilog/VHDL modules) and requires the command script mk_sdf.cmd to reside in the same directory as the perl script. Both, mk_sdf_204.pl and mk_sdf.cmd as well as an example of a testbench, the corresponding timing file and the resulting SDF file are provided in the directory utilities/CreateSDF.

Note: In command script mk_sdf.cmd the directory setting for the timingfile_dir has to be set to where mk_sdf_204.pl and mk_sdf.cmd are as well as testbench and timing file are residing e.g. to /user/USERNAME/CreateSDF.

4 Backdoor Memory Access

4.1.1 Overview

There is a separate TB developed to meet the needs of testing the backdoor memory access features of the memory model – semper_ultraLP4tb. The TB tests only these features and it doesn't support any checkers nor data checks.

The whole mechanism is developed using internal memory model signals.

4.1.2 Memory Model Dump signals and parameters

dump_mem – Force this signal to dump the whole memory array content. When using this feature for memory dump it requires the Banks to be consistent (either all set as SPI or LPDDR4), otherwise it will dump the data in byte format, and additional formatting might be required which is outside of this solution.

NOTE: If DumpAllInByteFromat is set to 0, then the model will dump all of the memory array content into one file. Otherwise the model will dump its contents into several files.

dmp_start_address – Force this signal to set the start address for dump_mem feature. Default value is 0. It is defined as a 32bit integer, so when assigning a value, it must use correct signal type.

dmp_end_address – Force this signal to set the end address for dump_mem feature. Default value is 0x1FFFFFF. It is defined as a 32bit integer, so when assigning a value, it must use correct signal type.

NOTE: If start and end addresses are not set then the whole memory array will be dumped. Also notice that dumping the memory with start and end address is possible only if DumpAllInByteFromat is set to 1.

dump_mem_ecc – Force this signal to initiate ECC memory dump. Output file is in byte format.

dump_mem_ssr – Force this signal to initiate SSR memory dump. Output file is in byte format.

4.1.3 Testing start and end address memory dump

In order to test start and end address of the memory dump, please use below lines. Also make sure to set the DumpAllInByteFromat prameter to 1.

```
// ------------------------------------------------------------------------
// Test star end address memory dump
//   $display("TB: dump mem content using start end addresses");
//   DUT.dmp_start_address = 0;
//   DUT.dmp_end_address   = 100;
//   DUT.dump_mem = 1'b1;
//   #LP4_Period DUT.dump_mem = 1'b0;
//   $system("cp s22hm256e_mem.dmp s22hm256e_mem.dmp-
SPI_with_mem_range");
// ------------------------------------------------------------------------
```

4.1.4 Peek and Poke Tasks

Peek task is provided in the testbench to backdoor read 8 bytes of data + 1 byte of ECC from the Memory Array. The task takes a byte address as the input and ignores the LSB 3 bits of the address to create an 8 byte aligned address. Thus, the peek task provides read data on an 8 byte aligned address.

Poke task is provided in the testbench to backdoor write 8 bytes of data + 1 byte of ECC to the Memory Array. The task takes a byte address as the input and ignores the LSB 3 bits of the address to create an 8 byte aligned address. Thus, the poke task writes to the Memory array on an 8 byte aligned address.

Note that the peek and poke tasks access the memory in zero simulation time. Hence care is needed when the task is invoked while transactions are ongoing on the LPDDR4 interface.

4.1.5 Test

The TB has one main initial block. Also, all memory reading is done in LPDDR mode.

Initially memory is preloaded by initial value of the preload file: "s22hm256e.mem". This is handled by the model itself and by setting the UserPreload parameter to 1 during the compile time.

Dump the memory content in SPI mode by forcing the dump_mem signal. This will dump all memory content into separate bank output files.

Force BNKAV and BNKNV registers to FF (moving the memory bank settings to LPDDR).

Invoke Poke task in a for loop to write data and ECC to the Array 8 bytes at a time.

Invoke Peek task in a for loop to read the same data and ECC from the array and visually ensure it is returning correct data in the waveforms.

Do a LPDDR4 read transaction to the same address and ensure correct data is received as set by poke task.

Dump the memory content in LPPDR mode by forcing the dump_mem signal. This will dump all memory content into separate bank output files.

Dump the ECC memory content by forcing dump_mem_ecc.

Dump the SSR memory content by forcing dump_mem_ssr.

5 Support

We appreciate your feedback or suggestions to ensure our models meet your needs. Please contact us at:

http://www.cypress.com/support