CEVA-XM4™

RTL V1.1.3.F Simulation Reference Guide

Rev. 1.1.3.F

June 2016

DSP Division

Documentation Control

History Table

Version	Date	Description	Remarks
V1.0.0.A	15 February 2015	First CEVA-XM4 released version	
V1.0.0.F	12 April 2015	Updated test list	
V1.1.0.F	5 August 2015	Major update for V1.1.0.F release	
V1.1.1.F	18 January 2016	Updated version and the test lists	
V1.1.2.F	9 March 2016	Updated version and the test lists	
V1.1.3.F	9 June 2016	Updated version and the test lists	

Disclaimer and Proprietary Information Notice

The information contained in this document is subject to change without notice and does not represent a commitment on any part of CEVA®, Inc. CEVA®, Inc. and its subsidiaries make no warranty of any kind with regard to this material, including, but not limited to implied warranties of merchantability and fitness for a particular purpose whether arising out of law, custom, conduct or otherwise.

While the information contained herein is assumed to be accurate, CEVA®, Inc. assumes no responsibility for any errors or omissions contained herein, and assumes no liability for special, direct, indirect or consequential damage, losses, costs, charges, claims, demands, fees or expenses, of any nature or kind, which are incurred in connection with the furnishing, performance or use of this material.

CEVA®, CEVA-XC™, CEVA-XC5™, CEVA-XC8™, CEVA-XC321™, CEVA-XC323™, CEVA-Xtend™, CEVA-XC4000™, CEVA-XC4100™, CEVA-XC4200™, CEVA-XC4210™, CEVA-XC4400™, CEVA-XC4410™, CEVA-XC4500™, CEVA-XC4600™, CEVA-TeakLite™, CEVA-TeakLite-II™, CEVA-TeakLite-III™, CEVA-TL3210™, CEVA-TL3211™, CEVA-TeakLite-4™, CEVA-TL410™, CEVA-TL411™, CEVA-TL420™, CEVA-TL421™, CEVA-Quark™, CEVA-Teak™, CEVA-X™, CEVA-X1620™, CEVA-X1622™, CEVA-X1641™, CEVA-X1643™, Xpert-TeakLite-II™, Xpert-Teak™, CEVA-XS1100A™, CEVA-XS1200™, CEVA-XS1200A™, CEVA-TLS100™, Mobile-Media™, CEVA-MM1000™, CEVA-MM2000™, CEVA-SP™, CEVA-VP™, CEVA-MM3000™, CEVA-MM3100™, CEVA-MM3101™, CEVA-XM™, CEVA-XM4™, CEVA-X2™, CEVA-Audio™, CEVA-HD-Audio™, CEVA-VoP™, CEVA-Bluetooth™, CEVA-SATA™, CEVA-SAS™, CEVA-Toolbox™, SmartNcode™ are trademarks of CEVA, Inc.

All other product names are trademarks or registered trademarks of their respective owners.

Support

CEVA® makes great efforts to provide a user-friendly software and hardware development environment. Along with this, CEVA provides comprehensive documentation, enabling users to learn and develop applications on their own. Due to the complexities involved in the development of DSP applications that might be beyond the scope of the documentation, an online Technical Support Service has been established. This service includes useful tips and provides fast and efficient help, assisting users to quickly resolve development problems.

How to Get Technical Support:

- **FAQs**: Visit our website http://www.ceva-dsp.com or your company's protected page on the CEVA website for the latest answers to frequently asked questions.
- **Application Notes**: Visit our website http://www.ceva-dsp.com or your company's protected page on the CEVA website for the latest application notes.
- **Email**: Use the CEVA central support email address ceva-support@ceva-dsp.com. Your email will be forwarded automatically to the relevant support engineers and tools developers who will provide you with the most professional support to help you resolve any problem.
- **License Keys**: Refer any license key requests or problems to sdtkeys@ceva-dsp.com. For SDT license keys installation information, see the *SDT Installation and Licensing Scheme Guide*.

Email: ceva-support@ceva-dsp.com

Visit us at: www.ceva-dsp.com

List of Sales and Support Centers

Israel	USA	Ireland	Sweden
2 Maskit Street P.O. Box 2068 Herzelia 46120 Israel **Tel**: +972 9 961 3700 **Fax**: +972 9 961 3800	1174 Castro Street Suite 210 Mountain View, CA 94040 USA **Tel**: +1-650-417-7923 **Fax**: +1-650-417-7924	Segrave House 19/20 Earlsfort Terrace 3rd Floor Dublin 2 Ireland **Tel**: +353 1 237 3900 **Fax**: +353 1 237 3923	Klarabergsviadukten 70 Box 70396 107 24 Stockholm Sweden **Tel**: +46(0)8 506 362 24 **Fax**: +46(0)8 506 362 20
China (Shanghai)	**China (Beijing)**	**China (Shenzhen)**	**Hong Kong**
Unit 1203, Building E Chamtime Plaza Office Lane 2889, Jinke Road Pudong New District Shanghai, 201203 China **Tel**: +86-21-20577000 **Fax**: +86-21-20577111	Rm 503, Tower C Raycom InfoTech Park No.2, Kexueyuan South Road Haidian District Beijing 100190 China **Tel**: +86-10 5982 2285 **Fax**: +86-10 5982 2284	Rm 709, Tower A SCC Financial Centre No. 88 First Haide Avenue Nanshan District Shenzhen 518064 China **Tel**: +86-755-8435 6038 **Fax**: +86-755-8435 6077	Level 43, AIA Tower 183 Electric Road North Point Hong Kong **Tel**: +852-39751264
South Korea	**Taiwan**	**Japan**	**France**
#478, Hyundai Arion 147, Gumgok-Dong Bundang-Gu Sungnam-Si Kyunggi-Do, 463-853 South Korea **Tel**: +82-31-704-4471 **Fax**:+82-31-704-4479	Room 621 No.1, Industry E, 2nd Rd Hsinchu, Science Park Hsinchu 300 Taiwan R.O.C **Tel**: +886 3 5798750 **Fax**: +886 3 5798750	1-6-5 Shibuya SK Aoyama Bldg. 3F Shibuya-ku, Tokyo 150-0002 Japan **Tel**: +81-3-5774-8250	RivieraWaves S.A.S 400, avenue Roumanille Les Bureaux Green Side 5, Bât 6 06410 Biot - Sophia Antipolis France **Tel**: +33 4 83 76 06 00 **Fax**: +33 4 83 76 06 01

Table of Contents

List of Examples

List of Figures

List of Tables

1. Introduction

1.1 Scope

This document describes the CEVA-XM4™ DSP core soft IP simulation environment and guidelines.

Important: *The ETM/RTT modules referred to in this document are an add-on feature separately licensed.*

1.2 Audience

This document is intended for ASIC designers who are implementing and embedding the CEVA-XM4 into their design.

1.3 Related Documents

The following documents are related to the information in this document:

1. *CEVA-XM4 Release Notes*
2. *CEVA-XM4 Database Reference Guide*
3. *CEVA-XM4 Integration Reference Guide*
4. *CEVA-XM4 Backend Reference Guide*

1.4 File Types

Table 1-1 describes the file types referred to in this document.

Table 1-1: CEVA-XM4 Simulation Environment File Types

File Type	Description
no extension	Binary executable
*.v	Verilog file
*.pl	PERL script file
*.csh	UNIX C Shell script file
*.asm	Assembly source test files
*.lst	Listing file Includes the program source code, the instruction encoding, and instruction packet addresses
*.mem	Verilog ASCII simulation test file Includes the opcodes that are loaded to the program memory
*.a	Compiled assembly files
*.log	Log files

2. Simulation Environment

The CEVA-XM4 is a re-usable Silicon Intellectual Property (SIP) DSP core. The IP source is an RTL Verilog code, which is technology-process independent and can be easily implemented in various technologies.

The simulation environment supports the following main features:

- Running Verilog simulation in RTL or gate-level with timing
- Running Verilog simulation using leading EDA vendor Verilog simulators
- Self-checking test suite, including system Verilog checkers
- One Verilog compilation per installed configuration for the entire test suite
- Various clock period simulations
- BFM simulation
- Writing new assembler tests
- Generation of simulation trace files for the programming model in simulation
- Transparent interface with SDT Bintools
- Running **.a** files

The following sections describe the simulation environment structure and components.

2.1 Simulation Environment Structure

Figure 2-1 shows the CEVA-XM4 /**simulation** directory structure.

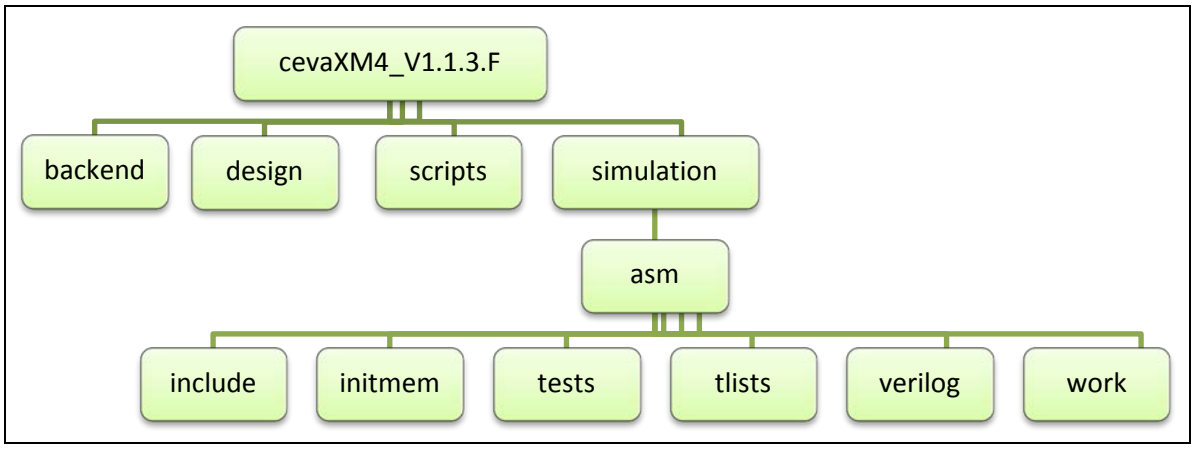


Figure 2-1: /simulation Directory Structure

Note: *Database directories that are not related to verification are not shown.*

Table 2-1 describes the directories in the CEVA-XM4 **/simulation/asm** directory.

Table 2-1: /simulation Directory Descriptions

Directory Name	Description
include	Macro files and definition files for the assembly tests
initmem	Internal and external data memory initializations
tests	Release ASM tests
tlists	Test list files
verilog	Simulative Verilog files

Directory Name	Description
work	Work directory for simulation. Tests must be executed from this directory. The following subdirectories are generated during simulation: • **lst**: Listing files for the compiled tests • **mem**: File post-assembler compilation • **jtag**: JTAG host instruction files The following files are generated during simulation: • **report<data><time>**: Simulation results summary • **envdef.v**: Verilog definitions file generated by the simulation script • **cevaxm4.vc**: The Verilog files list generated by the simulation script • **siggen_file.v**: Siggen utility simulation file (for more details, see Section 8) • **siggen_tasks_file.v**: Siggen utility simulation file (for more details, see Section 8) • **test_file.v**: Sets information file generated by the simulation script • **last.cmd**: The last simulation execution command

For details about the /**scripts** directory, see Section 5.1.

2.2 Simulation Environment Components

The simulation environment is built from the following components:

- **Verilog Test Bench**: Instantiates the device under test (DUT) and runs the test, as described in Section 2.2.1
- **PERL Simulation Script**: Sets up and runs the simulation, as described in Section 2.2.2

2.2.1 Verilog Test Bench

The Verilog test bench performs the following operations:

- Instantiates the RTL model of the device under test
- Loads program memory with the assembly test code
- Toggles input ports using the Siggen utility
- Monitors and analyzes test progress
- Reports the simulation status
- Iterates onto the next test (if it exists); otherwise, terminates the simulation

Figure 2-2 shows the top-level **cevaxm4_sim_top.v** verification and simulation environment module.

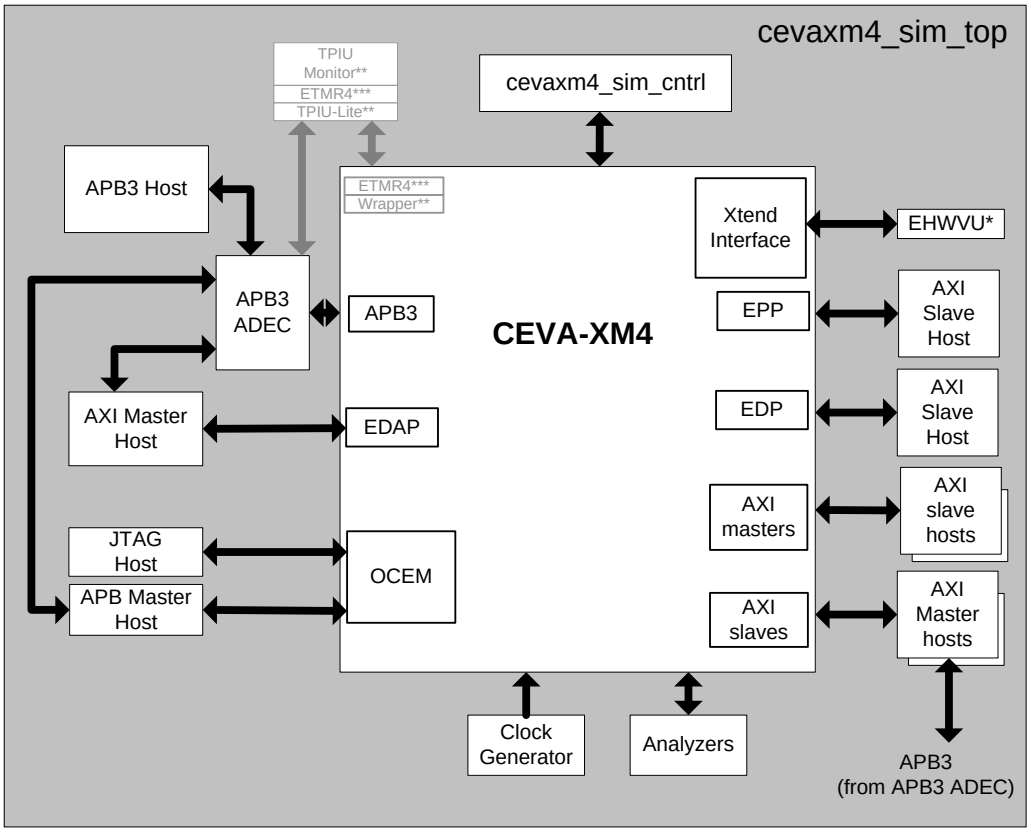


Figure 2-2: CEVA-XM4 Top-Level Simulation Module

The **cevaxm4_sim_top** module is a Verilog high-level code that includes the CEVA-XM4 module and other instances and logic, and is used to support the simulation environment.

Table 2-2 described the components shown in Figure 2-2.

Table 2-2: CEVA-XM4 Simulation Environment Components Description

Component	Description
cevaxm4_sim_cntrl	Simulative control unit
cevaxm4	DUT
XHWVU Hosts	Extend VU hardware hosts, connecting to VPU0/VPU1
apb3_adec	Address decoder of APB3 transactions to: • Core • AXI master • APB3 Master • APB3 Slave host

Component	Description
apb3_master	APB3 Master host
apb3_host	APB3 Slave port host
axi_if_master	AXI Master host
axi_host	AXI Slave host
ocem_host	Performs JTAG transactions to the OCEM
Clock Generator	Generates clock input to the CEVA-XM4 and to its peripherals
• cevaxm4_verifeq_an • cevaxm4_memory_interface_an • cevaxm4_axi_if_an • cevaxm4_apb3_an • cevaxm4_if_an	Analyzer modules, checks the correct behavior of the design
*CSTPIULITE **Tpiu_monitor ***ETMR4	Optional RTT components (TPIU-Lite, TPIU monitor, and external ETMR4)

2.2.2 Simulation Script

The **ceva_sim** simulation script executes the simulation and does the following operations:

- Processes user-input command line options
- Generates a test list according to the user's selections
- Compiles each assembly test using CEVA Software Development Tools (SDT)
- Generates memory output files
- Runs the Siggen utility on each test, extracting user-specified signals that are required to be driven during each test
- Creates several setup files that are required for the simulation
- Sets up and runs the simulation using the user-specified simulator
- Reports simulation status

3. Simulation Flow

3.1 Test Compilation Flow

During a test compilation, the **<test_name>.asm** assembly source test file is input to each test.

The tests' source code is written in the CEVA-XM4 native assembly syntax. They are assembled and mapped to the memory via the CEVA-XM4 Assembler tool.

Output files are generated when the Assembler is run on a test. Figure 3-1 shows the CEVA-XM4 test compilation flow:

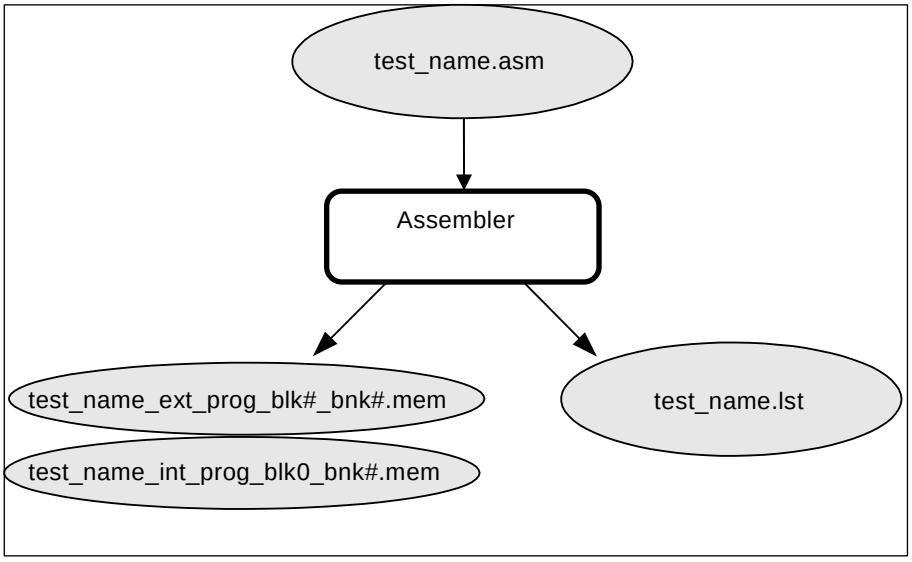


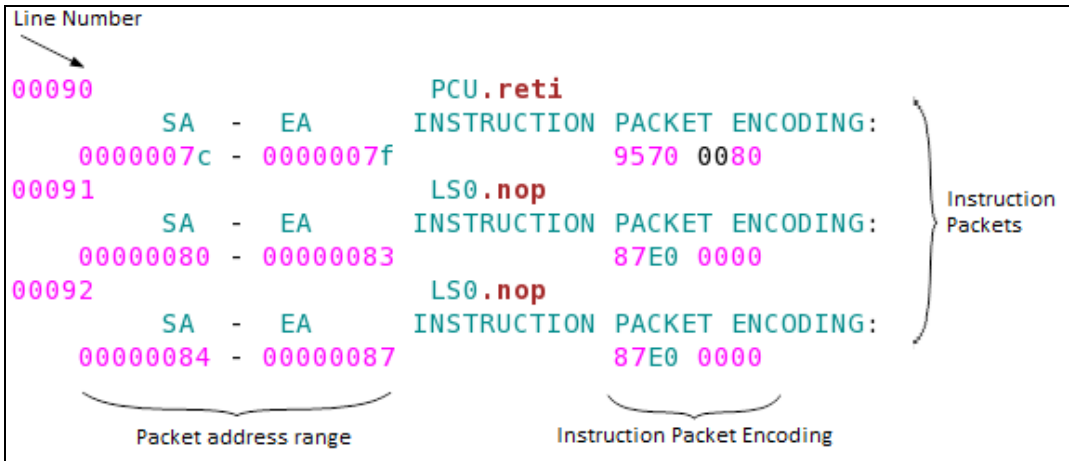
Figure 3-1: Test Compilation Flow

3.1.1 Listing File

The **<test_name>.lst** listing file contains the specific test assembly code (CEVA-XM4 native assembly), as used in the assembly source file. It also contains the instruction encoding and the relative program memory addresses of all instruction packets. This file is generated post-compilation and can be used for debugging.

Example 3-1 is an example of a listing file.

Example 3-1: Listing File



3.1.2 Memory Files

The memory files contain the program and data code of the tests. Output ***.mem** files are assembled from the memory address entry and its opcode content provided in hexadecimal notation. The **ceva_sim** simulation script partitions the source code into proper output files according to the code allocation. When simulation starts, the memory files are loaded to the correct memory.

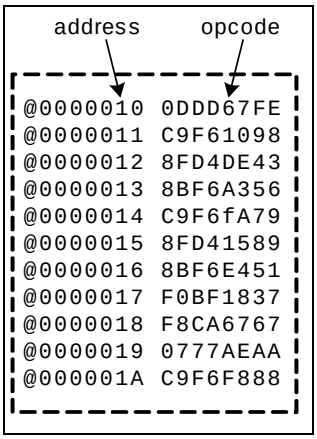
The generated output files are as follows:

- **<test_name>_ext_prog_blk#_bnk#.mem**: The program binary source code that is allocated to the external memory. These files are read to the external program memory via the **$readmemh** Verilog function.
- **<test_name>_int_prog_ blk#_bnk#.mem**: The program binary source code that is allocated to the internal core memory. These files are read to the internal program memory via the **$readmemh** Verilog function.
- **<test_name>_tag.mem**: The data source code that is allocated to the internal TAG memory. This file is read to the internal tag memory via the **$readmemh** Verilog function.
- **<test_name>_parity_blk#.mem**: The parity calculation of the source code. These files exist only if the ECC configuration is selected. These files are read into the internal core program memory via the **$readmemh** Verilog function.

Note: *The hash sign (#) represents the number of memory blocks/banks according to the memory configuration.*

Example 3-2 shows the structure of a memory file.

Example 3-2: Memory Output File



3.2 Simulation Termination

The simulation can be terminated by one of the following options:

- **Successful Pass**: When a test passes with no errors, the following message is issued:

```
TEST PASSED SUCCESSFULLY
```

- **Comparison Error**: When a comparison check fails during simulation, the test failure message is displayed. This message provides an indication of the program counter (PC) value related to the compare instruction that detected the mismatch.
- **Analyzer Error**: When an analyzer check fails during simulation, the analyzer failure message is displayed. This message provides an indication of the PC value related to the violation that was detected.
- **Timeout Error (forced completion)**: If, for some reason, the simulation enters an endless loop, it is stopped after a predefined period.
- **Compilation Error**: If the test assembly compilation fails, the simulation is stopped and a compilation error message is displayed.

4. Simulation Execution

4.1 Getting Started – Setting Variables

On the command line, execute the following command to set the environment variables for executing the simulation:

```
source <install_dir>/cevaXM4_V1.1.3.F/scripts/setenv.csh
```

The installation is completed.

Important: *Before starting simulation, ensure that the required HDL simulation tool and CEVA-XM4 Software Development Tools (SDT) are available.*

4.2 Executing the Simulation Script

To compile and execute a test, you must first change the directory to **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/work**.

Important: *The **ceva_sim** simulation script must be executed from this directory.*

To run a test, execute the following command:

```
ceva_sim -t <test_name>
```

To compile and execute a batch of tests, use the **-list** switch:

```
ceva_sim -list <list_name>
```

To view all available tests/lists, use the **-show** switch:

```
ceva_sim -show <tests|lists>
```

Where:

- **<test_name>** is one of the tests that reside in the **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/tests/release/** directory.
- **<list_name>** is one of the test lists that reside in the **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/tlists** directory.

When the simulation is completed, a **report_<DATE>** file containing the simulation results is created in the **/work** directory.

4.2.1 Gate-Level Simulation with Timing (SDF Simulation)

To execute gate-level simulation with timing (SDF simulation), do the following:

1. Open the **cevaxm4_gtl.vc** file, and then add paths to the following:
   - The **CEVA-XM4** netlist
   - The standard cells libraries (STD)
   - The physical Verilog memories
2. Open the cevaxm4_gtl_def.v file, and edit it so the defined signals point to the actual registers in the netlist.

Note: *The hierarchy might be changed due to backend tools optimization.*

3. Change the directory to **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/work**.
4. On the command line, execute the following:

```
ceva_sim -t  <test_name>  -sdf  <sdf_file_name> -
insertion_delay <delay in ns> -disable_analyzers ALL
```

5. Point to the full path of the SDF directory via the **-sdf** switch.

Tip: *In general, running simulation components (simulation **top** and **hosts**) and netlist might introduce order-of-event issues in the simulation.*

*To prevent this, use the **-insertion_delay** switch. The simulative host's clocks will be balanced to adjust their timing to the design delays.*

*The insertion delay value (for the hosts) should be derived from the STA report as the average propagation delay of the design root clock. Using **-insertion_delay <delay>** in the command injects the insertion delay into the host clocks in the **cevaxm4_sim_top.v** file.*

Force notifier is created upon synchronized inputs. This is done to prevent 'xxx's from flowing into the design. Table 4-1 describes the force notifier F.Fs.

Table 4-1: Synchronizer List

ID	Signal Name	Description
1.	cevaxm4_psu_ocem_reset_synchronizer_ sample_r_reg	OCEM reset synchronizer
2.	cevaxm4_psu_global_reset_synchronizer_ sample_r_reg	Global reset synchronizer
3.	cevaxm4_psu_sys_reset_synchronizer_ sample_r_reg	System reset synchronizer
4.	cevaxm4_psu_core_reset_synchronizer_ sample_r_reg	Core reset synchronizer
5.	cevaxm4_psu_tms_sync_ar_datasync_r_ reg_0_	TMS synchronizer
6.	cevaxm4_psu_tdi_sync_ar_datasync_r_ reg_0_	TDI synchronizer
7.	cevaxm4_psu_stop_sd_sync_ar_datasync_ r_reg_0_	Stops shutdown synchronizer
8.	cevaxm4_psu_ext_wait_sync_ar_datasync _r_reg_0_	External wait synchronizer
9.	cevaxm4_psu_tck_sync_ar_datasync_r_ reg_0_	TCK synchronizer

The **force_notifier.v** statements are located in the **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/verilog/top/ force_notifier.v** file.

4.2.2 UPF Power Simulation

To simulate power domains in CEVA-XM4, the user has to run simulation with a power-gating definition written in a UPF file. This file describes the power domains during shutdown and isolation values out of these domains. To run with a UPF file, the **-upf** switch must be added to the **ceva_sim** command.

The UPF file for CEVA-XM4 is located in the **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/upf/cevaxm4.upf** file.

To execute power simulation with UPF, you must first change the directory to **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/work**.

To run a single test from the power list, execute the following command:

```
ceva_sim -t <test_name from power list> -real_mem
-upf ../upf/cevaxm4.upf  -disable_analyzers ALL
```

To run a list of tests, execute the following command:

```
ceva_sim -list cevaxm4_psu_power_list -real_mem -upf
../upf/cevaxm4.upf -disable_analyzers ALL
```

Note: *When installing the package with power gating, tests should be run with real memories.*

Simulation with real memories can also be executed without using power modes simulation. To run a test with real memories, execute the following command:

```
ceva_sim -t <test name> -real_mem
```

Before using the **-real_mem** switch, the user must create a new file called **real_mem.vc** under the **<install_dir>/cevaXM4_V1.1.3.F/design/vc_file** directory. The file should contain the path for the DMEM/PMEM wrapper files, as well as the path to the memory cells used inside the wrappers.

Two reference files named **cevaxm4_dmem_28hpm_arm_hd.v** (DMEM) and **cevaxm4_pmem_28hpm_arm_hd.v** (PMEM) are available in the **<install_dir>/cevaXM4_V1.1.3.F/design/top** directory.

5. Simulation Script

The **ceva_sim** simulation executable script executes all CEVA-XM4 tests, and does the following:

- Processes user-input command line options
- Generates a test list according to the user's selections
- Compiles/assembles each test in the test list using the SDT tools (if selected by the user)
- Runs the Siggen utility on each test, extracting user-specified signals that are required to be driven during each test
- Creates several setup files that are required for the simulation
- Sets up and runs the simulation using the user-specified simulator

5.1 ceva_sim Associated Scripts

Table 5-1 describes the files in the /**scripts** directory.

Table 5-1: CEVA-XM4 Script Files

Filename	Description
setenv.csh	Defines the basic project paths
ceva_sim	Simulation executable script

Table 5-2 describes the files in the **/scripts/conf** directory.

Table 5-2: CEVA-XM4 Configuration Files

Filename	Description
cevaxm4.cf	Script configuration file
logger.cf	Script logger configuration file

5.2 ceva_sim Input Files

The **ceva_sim** executable script uses the following input files to execute the CEVA-XM4 simulation:

- **<test_name>.asm**: The assembly test. The script uses the ASM file from the /**asm** directory. If the **-no_asm_compile** switch is used, the assembler compilations are not executed and this file is not used.
- **list_name**: When using a list of assembly tests (instead of a single assembly test), the script searches the /**tlists** directory and the /**work** directory for the list files.
- **<*>.vc**: These files are used by the script to locate all of the Verilog files required by the simulator to run a simulation (for example, RTL files, test bench files, analyzer files, memory model files, and so on). These file are located in the **<install_dir>/cevaXM4_V1.1.3.F/ design/vc_files** directory.

5.3 ceva_sim Intermediate Files

The **ceva_sim** executable script generates the following intermediate files during a CEVA-XM4 simulation:

- **mem/*.mem**: The simulation output files to be loaded to the correct memory component. For more details, see Section 3.1.2.
- **lst/<test_name>.lst**: The test listing file. For more details, see Section 3.1.1.

5.4 ceva_sim Output Files

The following files are the main report and error files that the user should check after a simulation run. These files are output to the /**work** directory:

- **report _<date><time>**: This is the main report file output from the simulator. It contains details about the type of simulation executed and lists the test's progress throughout the simulation. It also contains a final test PASS/FAIL count at the end of the file.
- **ceva_sim.log**: This is the main log file output from the simulator. It contains the simulation script messages.
- **Signal Recording Files**: These files reside in the **<test_name>** directory. The recording file type depends on the simulator used, as follows:
  - **NC Simulator**: **waves.trn** and **waves.dsn** are generated under **waves** if the **-record** switch is used for NC simulations.
  - **ModelSim Simulator: vsim.wlf** is generated in the **/work** directory if the **-record** switch is used for ModelSim simulations.
  - **VCS Simulator: vcdplus.vpd** is generated in the **/work** directory if the **-record** switch is used for VCS simulations.
- **last.cmd**: The last script command line is saved to this file to enable re-running with the same switches.

5.5 Verilog Test Bench Definitions

During the simulation process, some Verilog definitions are generated by the PERL script. These definitions reside in the **envdef.v** file under the /**work** directory. The **envdef.v** file is part of the Verilog files that are defined in the **cevaxm4.vc** Verilog compilation file list.

Example 5-1 shows an **envdef.v** file.

Example 5-1: envdef.v File

```
`timescale 1ns/10ps
`define INSERTION_DELAY 0
`define OUTPUT_DELAY 0
`define EXT_DATA_INIT_MEM_DIR
"<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/initmem/init_data_
ext"
`define TOP cevaxm4_sim_top
`define INT_DATA_INIT_MEM_DIR
"<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/initmem/4blk_256kb
"
`define INPUT_DELAY 0
`define APB3_RATIO 1
`define AXIS1_DIV_EN 1
`define MAX_TIME 2500000
`define MAIN_CLK_PERIOD 5
`define TCK_CLK_PERIOD 40
`define HCLK_RATIO 1
`define AXIM0_DIV_EN 1
`define RTL
`define NC
`define START_PULSE_DELAY 5
`define AXIS0_DIV_EN 1
`define RUN_WITHOUT_REAL_MEM
`define AXIM1_DIV_EN 1
`define AXIS2_DIV_EN 1
`define REPORT "Report_Thu_Feb_19_18-03-06_2015"
`define ENABLE_CHECKERS 64'hFFFFFFFFFFFFFFFF
```

5.6 ceva_sim Switches

The **ceva_sim** script supports several switches that enable various modes of simulation. Table 5-3 describes the different types of **ceva_sim** switches.

Table 5-3: ceva_sim Switches

Switch	Description
-add_elab_switches	Add switches to the elaboration *irun* command explicitly
-add_irun_switches	Adds specific *irun* switches to the IES simulator
-add_modelsim_switches	Adds specific ModelSim (vsim) switches to the ModelSim simulator
-add_simv_switches	Add switches to the *simv* command explicitly. Usage: `-add_simv_switches "<add switches here>"`
-add_vcs_switches	Adds specific VCS switches to the VCS simulator
-apb3_ratio	**<ratio>** = The ratio between the APB3 clock and the ceva_clk (default is **1**)
-axim0_clk_ratio	The ratio between the AXIm0 clock and the ceva_clk
-axim1_clk_ratio	The ratio between the AXIm1 clock and the ceva_clk
-axis0_clk_ratio	The ratio between the AXIs0 clock and the ceva_clk
-axis1_clk_ratio	The ratio between the AXIs1 clock and the ceva_clk
-axis2_clk_ratio	The ratio between the AXIs2 clock and the ceva_clk
-edap_clk_ratio	The ratio between the EDAP clock and the ceva_clk
-epp_clk_ratio	The ratio between the EPP clock and the ceva_clk
-benchmark_ref_file	**<filename>** = The name of the reference file for which you want to compare the data you dumped from the benchmark
-compile_only	Compiles the test and exits
-dbg_file	**<dbg file>** = Runs a benchmark test

-define	**<definition>** = Adds a definition to the simulation:
	• For nc:
	`-define "MACRO VALUE"`
	• For VCS/ModelSim (multiple switches):
	`-define "MACRO=VALUE"`

-disable_analyzers	**<Analyzer1|Analyzer2|...|ALL>** = Disables the following analyzers (all of the others are enabled). **ALL** disables all of the analyzers. When using this switch, use the implicit analyzer's name, like in the **cevaxm4_sim_top** file.

Switch	Description
-disable_dps_mode	Disables DPS mode
-edap_init_mem	Creates memory and address files from the **test.out** file
-enable_analyzers	**<Analyzer1|Analyzer2|...>** = Enables the following analyzers (all of the others are disabled). When using this switch, use the implicit analyzer's name, like in the **cevaxm4_sim_top** file.
-enable_mss_free_clk	Enables MSS free mode
-envdef	Uses the user's local **envdef** file
-envdef_path	Local path to the **envdef** file
-gui	Opens the Verilog simulator GUI
-h|help	Prints a help menu with all of the switches
-hclk_ratio	**<ratio>** = The ratio between the hclk and the ceva_clk (default is **1**)
-insertion_delay	**<time ns>** = Insertion delay for hosts in SDF simulation
-list	Lists of tests to run
-logger	Runs with logger
-macro	**<macro=value>** = Defines an ASM .EQU (multiple switches, can be used more than once)
-main_period	Main clock period (default is **5 ns**)
-max_cycles	Maximum number of cycles for each test simulation
-no_asm_compile	Does not compile the test
-noRstrCheck	Runs the ASM compiler with no restriction checks
-rec_end	**<end record time (ns)>** = Stops dumping waves at the time specified
-rec_hierarchy	**<module full path>** = The full path (as a delimiter) to the hierarchy you want to record (for example, **cevaxm4_sim_top.cevaxm4.cevaxm4_sys.cevaxm4_core_top**).
-rec_mem	Records memory as well as signals
-rec_start	**<start record time (ns)>** = Begins dumping waves at the time specified
-rec_tasks	Records internal tasks' signals
-record	Records simulation waves
-report	**<file_name>** = Writes the simulation report to this file
-rerun_all	**<report_name>** = Runs all of the tests from the report (multiple switches)
-rerun_failed	**<report_name>** = Runs only the tests that failed from the report (multiple switches)

Switch	Description
-rerun_pass	**<report_name>** = Runs only the tests that passed from the report (multiple switches)
-real_mem	Runs the simulation using RTL with real memory modules
-sdf	**<sdf file>** = Runs SDF simulation
-show	**<tests|lists>** = Shows available tests or lists to run
-sim	**<nc/vcs/modelsim>** = The Verilog simulator to run
-simulator_verbosity	Runs the simulator in system command mode and with full verbosity
-t	Test name to run (multiple switches, can be used more than once)
-tck_period	TCK clock period (default is **40 ns**)
-use_compiled_test	Uses an already compiled test (in which case you must have a **<test_name>.a** file in your working directory)
-use_local_lnk	**<lnk file>** = Uses the local lnk file
-vc_file	**<path to your vc file>** = Runs the simulation using your own VC file.
-vcd	**<vcd file>** = Uses the VCD database
-verbosity	Forces verbosity (even if running lists)
-verdi	Records waves for viewing with the NOVAS Verdi tool
-upf	Runs simulation with a power UPF format file

6. Simulation Test Files

6.1 Test Descriptions

All CEVA-XM4 assembler tests are in the **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/tests/release/** directory.

6.2 Test List Files

The assembly (ASM) tests are in the following lists:

- **cevaxm4_release_without_simulative_switches**: CEVA-XM4 ASM tests that **cannot** run with the simulative switches (as described in Section 6.2)
- **cevaxm4_release_with_simulative_switches**: CEVA-XM4 ASM tests that **can** run with simulative switches (as described in Section 6.2.2)
- **cevaxm4_etm_list**: RTT wrapper tests (as described in Section 6.2.3)
- **cevamx4_axi_interface**: A subset of tests from other lists, which operates the AXI interfaces of the CEVA-XM4

All of the list files are in the **<install_dir>/cevaXM4_V1.1.3.F/ simulation/asm/tlists/** directory.

6.2.1 Tests That Cannot Run with Simulative Switches

Table 6-1 describes the tests that cannot be run with the following simulative switches:

- **-apb3_ratio**
- **-axim0_clk_ratio**
- **-axim1_clk_ratio**
- **-axis0_clk_ratio**
- **-axis1_clk_ratio**
- **-axis2_clk_ratio**
- **-hclk_ratio**
- **-main_period**
- **-tck_period**

Table 6-1: cevaxm4_release_without_simulative_switches Test Descriptions

ID	Test	Description
1.	cevaxm4_dmss_gvi	Checks various GVIs
2.	cevaxm4_dmss_gvi_wrc	Checks the WRC GVI for EDP/AXIm write response counters reaching the maximum.
3.	cevaxm4_dmss_dma_q	Checks DDMA operation of uploads and downloads using EDP and AXIM0/1 (if present)
4.	cevaxm4_dmss_dma_q_ext	Checks DDMA operation of uploads and downloads using EDP with external control via next_ddma input
5.	cevaxm4_dmss_ext_rdwr	Performs all size LS reads/writes through EDP and AXIM0/1 (if present)
6.	cevaxm4_dmss_ext_bnd	• Performs read and write accesses crossing external 4KB memory boundaries (AMBA) • Shows writes with all zero-byte strobes
7.	cevaxm4_if_mask_out_int	• Checks asserting int0, int1, int2, and vint • Receives the acknowledge accordingly
8.	cevaxm4_pmss_pdma_epp_len2_thpt(8)	Checks the throughput of EPP read address and data channels. The PDMA performs downloads from the EPP to PTCM burst length 2.
9.	cevaxm4_pmss_pdma_epp_len4_thpt(8)	Checks the throughput of EPP read address and data channels. The PDMA performs downloads from the EPP to PTCM burst length 4.
10.	cevaxm4_pmss_pdma_epp_len8_thpt(8)	Checks the throughput of EPP read address and data channels. The PDMA performs downloads from the EPP to PTCM burst length 8.

ID	Test	Description
11.	cevaxm4_pmss_pdma_epp_len16_thpt[8]	Checks the throughput of EPP read address and data channels. The PDMA performs downloads from the EPP to PTCM burst length 16.
12.	cevaxm4_pmss_swop_epp_thpt	Checks the throughput of EPP read address and data channels. The SWOP performs pre-fetch from the EPP to PCACHE.
13.	cevaxm4_pmss_core_epp_thpt	Checks the throughput of EPP read address and data channels. The core performs reads from the EPP. Each instruction packet consumes a full fetch line.
14.	cevaxm4_pmss_core_epp_cachable_thpt	First invalidates the entire cache, and then starts core reads. Each packet instruction consumes an entire cache line.
15.	cevaxm4_pmss_core_epp_cachable_hwpf_thpt	First invalidates the entire cache, and then start core reads. Each packet instruction consumes an entire cache line. Hardware pre-fetch indication is set.
16.	cevaxm4_dmss_iol	• Core IO and External Device Access Port (EDAP) and AXIS0 reads and writes to different registers • Interleaves reads and writes from Core IO and EDAP and AXIS0 to PMSS, DMSS, PSU, QMAN, MCCI, OCEM, PROFILER
17.	cevaxm4_dmss_edap_axi4[3]	EDAP read and write to TCM, R/W burst accesses with length = 256. Because it uses AXI loopback, the AXI Master (EDP) and Slave (EDAP) must have the same AXI bus width and clock ratio.
18.	cevaxm4_dmss_slv0_axi4[4][5][9]	AXI SLAVE0 read and write to TCM, R/W burst length = 128 (in AXI128 bus) or 256 (in AXI256 bus). Because it uses AXI loopback, the AXI Master (AXIm0) and Slave (AXIs0) must have the same AXI bus width and clock ratio.
19.	cevaxm4_dmss_slv1_axi4[4][6][10]	AXI SLAVE1 read and write to TCM, R/W burst length = 128 (in AXI128 bus) or 256 (in AXI256 bus). Because it uses AXI loopback, the AXI Master (AXIm1) and Slave (AXIs1) must have the same AXI bus width and clock ratio.

ID	Test	Description
20.	cevaxm4_dmss_slv2_axi4[7][11]	• AXI SLAVE2 read and write to TCM, R/W burst length = 128 (in AXI128 bus) or 256 (in AXI256 bus). • DMA upload and download using EDP then looped back to SLV2 or EDAP. • In AXI256 mode, read and write access loop from EDP to SLV2. • In AXI128 mode, read and write access loop from EDP to EDAP. Because it uses AXI loopback, the AXI Master (EDP) and Slave (EDAP/AXIs2) must have the same AXI bus width and clock ratio.
21.	cevaxm4_dmss_edap_cap[3]	• Read capability: De-asserts read data channel *rready* to prove that the EDAP can accept nine read addresses • Write capability: De-asserts write response channel *bready* to prove that the EDAP can accept seven write addresses. Because it uses AXI loopback, the AXI Master (EDP) and Slave (EDAP) must have the same AXI bus width and clock ratio.
22.	cevaxm4_dmss_slv0_cap[4][5][9]	• Read capability: De-asserts read data channel *rready* to prove that the SLV0 can accept nine read addresses. • Write capability: De-asserts write response channel *bready* to prove that the SLV0 can accept seven write addresses Because it uses AXI loopback, the AXI Master (AXIm0) and Slave (AXIs0) must have the same AXI bus width and clock ratio.
23.	cevaxm4_dmss_dma_dman_download_debug[2]	• Programs the DDMA to do a download transfer and turns on debug mode during the transfer • Checks that the DDMA stops after the current burst is completed and asserts ddma_ocem_idle output • Turns off debug mode and checks that the transfer completes successfully

ID	Test	Description
24.	cevaxm4_dmss_dma_dman_upload_debug[2]	• Programs the DDMA to do an upload transfer and turns on debug mode during the transfer • Checks that the DDMA stops after the current burst is completed and asserts ddma_ocem_idle output • Turns off debug mode and checks that the transfer completes successfully
25.	cevaxm4_dmss_dma_dman_iit_debug[2]	• Programs the DDMA to do an internal transfer and turns on debug mode during the transfer • Checks that the DDMA stops immediately • Turns off debug mode and checks that the transfer completes successfully
26.	cevaxm4_sys_watchdog_timer[4]	Checks sys watchdog. Check GVI and relevant DBG_GEN_2 bits are; asserted on reset-early and reset-late and not asserted on reset between min/max thresholds. Check sys timer. Check it's; not reset by read, it is not overwritten or reset by a write, it is reset by core reset.
27.	cevaxm4_ecc_test[1]	Generates one and two errors in: • Set data • Tag data The status of the **ecaddr** CPM register is verified via I/O every time it is updated.
28.	cevaxm4_ocem_id_change	Verifies that the PMEM OCEM external ID is changed during the test
29.	cevaxm4_mss_shw_gvi[12]	Assert all GVI violations by writing to the debug shadow register
30.	cevaxm4_axis0_iol[5]	• AXIS0 reads and writes to different registers • Interleaves reads and writes from EDAP to PMSS, DMSS, PSU, QMAN, MCCI, OCEM, PROFILER • Includes dw, 2dw, and 4dw single and burst access types
31.	cevaxm4_axis1_iol[6]	• AXIS1 reads and writes to different registers • Interleaves reads and writes from EDAP to PMSS, DMSS, PSU, QMAN, MCCI, OCEM, PROFILER • Includes dw, 2dw, and 4dw single and burst access types

ID	Test	Description
32.	cevaxm4_axis2_iol(7)	• AXIS2 reads and writes to different registers • Interleaves reads and writes from EDAP to PMSS, DMSS, PSU, QMAN, MCCI, OCEM, PROFILER • Includes dw, 2dw, and 4dw single and burst access types
33.	cevaxm4_edap_iol	• EDAP reads and writes to different registers • Interleaves reads and writes from EDAP to PMSS, DMSS, PSU, QMAN, MCCI, OCEM, PROFILER • Includes dw, 2dw, and 4dw single and burst access types
34.	cevaxm4_dmss_acu_lock_acu_slv_acc(13)	• Checks the **acu_lock** and **acu_slv_acc** core inputs • In AXI128 mode, read and write access loop from EDP to EDAP Because the test uses AXI loopback, the AXI Master (EDP) and Slave (EDAP) must have the same AXI bus width and clock ratio. • Because the test accesses the CPM, the access cannot be 256 bits.
35.	cevaxm4_dman_en_cnt_no_rptr_dec(2)	Checks that the DMAN does not update Q0_EN_CNT0/1 when the internal read pointer is different from the external read pointer and Q0_DSC_CNT_CFG_0/1 is set.
36.	cevaxm4_dmss_dma_dman_upload_fixed (2)	The QMAN is programmed with an upload task of 0x180 bytes with *BSZ* = 'fixed'. Task is to be broken into multiple chunks. The test checks that when the QMAN sends the second and third chunks to the DDMA, it does not increment the DDEA.

Notes:

1. *This test exists only when Memory ECC is installed.*
2. *This test exists only when Queue manager is installed.*
3. *This test exists only when AXIM WIDTH is 128 bits.*
4. *This test exists only when AXI Data Master Ports 0 and 1 exist.*
5. *This test exists only when AXI slave0 is installed.*
6. *This test exists only when AXI slave1 is installed.*
7. *This test exists only when AXI slave2 is installed.*
8. *This test exists only when Program TCM size is not 0.*
9. *This test does not exist when AXIM0 WIDTH is not equal to AXIS0_WIDTH.*
10. *This test does not exist when AXIM1 WIDTH is not equal to AXIS1_WIDTH.*
11. *This test does not exist when EDP WIDTH is not equal to AXIS2_WIDTH.*
12. *This test exists only when Memory ECC and BUS_ECC are installed.*
13. *This test exists only when EDP WIDTH is 128 bits.*

6.2.2 Tests That Can Run with Simulative Switches

Table 6-2 describes the tests that can be run with simulative switches.

Table 6-2: cevaxm4_release_with_simulative_switches Test Descriptions

ID	Test	Description
1.	cevaxm4_dman_qnen_cnt_ext_inc(4)	• Checks qnen_cnt0 increment by external inc signal • Checks only if the relevant Q qnen_cnt0 is updated
2.	cevaxm4_dman_qman_semaphore(4)	• Holds the queues' FSM by not granting the semaphore • Grants the semaphore queue by queue • Verifies that the queues continue as expected
3.	cevaxm4_ocem_jtag_dmem_rd	Checks internal data memory read option by the OCEM JTAG
4.	cevaxm4_ocem_jtag_vpu_vec_rd	OCEM reads VPU vectors through the JTAG
5.	cevaxm4_ocem_dadd_2w_ls0	Data address BP for some configurations of word accesses
6.	cevaxm4_ocem_padd1_bp_regular	Checks the program address breakpoint
7.	cevaxm4_ocem_jtag_ext_pmem_wr	Checks that the OCEM writes programs to the external program memory
8.	cevaxm4_dmss_dma_dbg_match_ocem	Checks that the DMA Debug match generates a BP to the OCEM
9.	cevaxm4_ocem_jtag_single_step	Checks the Single Step feature in the OCEM
10.	cevaxm4_ocem_jtag_stop_go	Checks the Stop/Go feature in the OCEM
11.	cevaxm4_ocem_jtag_bs_reg	Checks the Boundaries scan register
12.	cevaxm4_ocem_ext_bp_req	Checks the external BP request acceptance
13.	cevaxm4_ocem_apb_gp_out	Verifies that the GPIO is written to the JTAG
14.	cevaxm4_ocem_comb1_rd_b_vu	Checks combined BP read from VPU
15.	cevaxm4_ocem_comb1_wr_b_vu	Checks combined BP write from VPU
16.	cevaxm4_ocem_dvm1_bp_b_1641	Checks DVM BP from non-VPU access
17.	cevaxm4_ocem_jtag_core_id_all5	Checks core ID read by JTAG (all "5")
18.	cevaxm4_ocem_jtag_core_id_allA	Checks core ID read by JTAG (all "A")
19.	cevaxm4_ocem_apb_wrong_addr	Checks illegal address access by the APB
20.	cevaxm4_ocem_core_rst_boot	Checks core reset from the OCEM
21.	cevaxm4_ocem_mss_rst_boot	Checks MSS reset from the OCEM

ID	Test	Description
22.	cevaxm4_ocem_boot1_msk1	Checks that, when boot is set and boot mask is 1, the DSP starts from PC 0 and ignores the boot
23.	cevaxm4_boot_swop_ext	Checks the boot when the EDAP configures the PMSS SWOP to update the cache
24.	cevaxm4_boot_light_sleep_edap(10)	Checks that the EDAP can access the TCM while the core is in light sleep
25.	cevaxm4_boot_light_sleep_pdma(10)	Checks that the EDAP can access the PMSS and activate the PDMA while the core is in light sleep
26.	cevaxm4_boot_edap_pdma_core_reset(10)	Checks the boot sequence when the EDAP initializes the PDMA while the core is in reset. The program is downloaded to the PTCM while the core is still in reset. When the download is finished, the core reset is deasserted, and the core starts the execution from the PTCM.
27.	cevaxm4_boot_input	Checks the boot input mechanism
28.	cevaxm4_vpu_mechanism	Demonstrates different VPU command capabilities
29.	cevaxm4_xtend_scalar(2)	Checks the scalar external hardware
30.	cevaxm4_xtend_vpu(3)	Checks the vector external hardware
31.	cevaxm4_scalar_mech	Checks the following sub, shiftr, nand, mac, ffb scalar mechanisms
32.	cevaxm4_scalar_bypasses	Checks the scalar-to-scalar bypasses for GRF and predicates
33.	cevaxm4_if_gpout	Checks the gpout output bus
34.	cevaxm4_if_mask_out_int_3(10)	• Checks that the PDMA interrupt is asserted • Checks that the acknowledge is received accordingly
35.	cevaxm4_if_trap	Checks the cevaxm4_seq_trp_srv_r output
36.	cevaxm4_if_permission_violation	Checks the ext_vom and ext_pv inputs
37.	cevaxm4_if_op_mode	Checks operation mode violations (in user1 mode)
38.	cevaxm4_if_core_id	Checks reading the core_id inputs
39.	cevaxm4_if_general_input	Checks the boot and vector inputs
40.	cevaxm4_if_apb	Checks the apb3 interface
41.	cevaxm4_pmss_epp_cov(10)	• Toggles the EPP interface • Demonstrates accesses of the core and PDMA with different lengths to the EPP

ID	Test	Description
42.	cevaxm4_epp_toggle[10]	• Toggles the EPP interface • Demonstrates PDMA download (the download crosses IACU regions) • Checks EPP access with different AXI attributes
43.	cevaxm4_vpu0_bypass	Demonstrates VPU bypass capabilities
44.	cevaxm4_if_undef_opcode	• Checks encoding an illegal opcode • Checks the uop output • Reads the uop status register
45.	cevaxm4_if_mask_out_int_4	Checks int4 (DDMA interrupt) acknowledge
46.	cevaxm4_if_dma_bp_int	Checks cevaxm4_seq_bp_ack_n_r output by matching data address in the DDMA
47.	cevaxm4_dmss_edap_tcm	AXI EDAP read and write to TCM boundary: • Writes the top 4dw of memory and checks the OK response • Reads the top 4dw of memory and checks the data and OK response • Writes byte to memory top +1 and checks the SLVERR response • Reads byte from memory top +1 and checks the SLVERR response
48.	cevaxm4_dmss_slv0_tcm[7]	AXI SLV0 read and write to TCM boundary: • Writes the top 4dw of memory and checks the OK response • Reads the top 4dw of memory and checks the data and OK response • Writes byte to memory top +1 and checks the SLVERR response • Reads byte from memory top +1 and checks the SLVERR response
49.	cevaxm4_dmss_slv1_tcm[8]	AXI SLV1 read and write to TCM boundary: • Writes the top 4dw of memory and checks the OK response • Reads the top 4dw of memory and checks the data and OK response • Writes byte to memory top +1 and checks the SLVERR response • Reads byte from memory top +1 and checks the SLVERR response

ID	Test	Description
50.	cevaxm4_dmss_slv2_tcm[9]	AXI SLV2 read and write to TCM boundary: ● Writes the top 4dw of memory and checks the OK response ● Reads the top 4dw of memory and checks the data and OK response ● Writes byte to memory top +1 and checks the SLVERR response ● Reads byte from memory top +1 and checks the SLVERR response
51.	cevaxm4_dmss_edap_mcci	4dword EDAP to mcci access. Checks the command registers, status bits, interrupt enables, and interrupts.
52.	cevaxm4_dmss_slv0_mcci[7]	4dword SLV0 to mcci access. Checks the command registers, status bits, interrupt enables, and interrupts.
53.	cevaxm4_dmss_slv1_mcci [8]	4dword SLV1 to mcci access. Checks the command registers, status bits, interrupt enables, and interrupts.
54.	cevaxm4_dmss_slv2_mcci [9]	4dword SLV2 to mcci access. Checks the command registers, status bits, interrupt enables, and interrupts.
55.	cevaxm4_dmss_edap_snoop	EDAP SNOOP DETECT = Read and write accesses to start-1, start, top, and top+1 addresses
56.	cevaxm4_dmss_slv0_snoop[7]	SLV0 SNOOP DETECT = Read and write accesses to start-1, start, top, and top+1 addresses
57.	cevaxm4_dmss_slv1_snoop[8]	SLV1 SNOOP DETECT = Read and write accesses to start-1, start, top, and top+1 addresses
58.	cevaxm4_dmss_slv2_snoop[9]	SLV2 SNOOP DETECT = Read and write accesses to start-1, start, top, and top+1 addresses
59.	cevaxm4_dmss_edap_cpm_resp	● EDAP read and write to each programming model section ● Read and write addresses so both SLVERR and OK responses are returned.
60.	cevaxm4_dmss_slv0_cpm_resp[7]	● SLV0 read and write to each programming model section ● Read and write addresses so both SLVERR and OK responses are returned.
61.	cevaxm4_dmss_slv1_cpm_resp[8]	● SLV1 read and write to each programming model section ● Read and write addresses so both SLVERR and OK responses are returned.

ID	Test	Description
62.	cevaxm4_dmss_slv2_cpm_resp[9]	• SLV2 read and write to each programming model section. • Read and write addresses so both SLVERR and OK responses are returned.
63.	cevaxm4_dmss_mem_boundaries	• Checks the boundaries of internal TCM memories • Toggles all possible memory interface signals
64.	cevaxm4_dmss_ecc_err[1]	• Forces a TCM Read data ECC error, which causes a core reload of V1 data • Has Read after Write NO-Wait and reload together
65.	cevaxm4_dmss_monitor	Checks exclusive access and associated error response checking
66.	cevaxm4_dmss_raw_wait_nowait	Loads/stores RAW at all WB stages
67.	cevaxm4_dmss_so	Loads/stores SO at all WB stages, plus full/partial matching
68.	cevaxm4_dmss_st_contention	Stores bank contention plus APV, blank region, and predicate aborts
69.	cevaxm4_dmss_raw	Loads/stores RAW mixed vector and scalar
70.	cevaxm4_dmss_st_raw_contention	Loads/stores RAW plus bank store contention
71.	cevaxm4_dmss_ap	• Tests read and write access protection violations • Checks the MAPV bit and MAPSR and MAPAR registers
72.	cevaxm4_psu_standby	Checks standby mode and recovery
73.	cevaxm4_psu_axi_low_power	Checks AXI low power interface in the PSU
74.	cevaxm4_psu_lightsleep_pgr_ae	Checks the activation of the AE bits in the PGR register
75.	cevaxm4_dmss_dma_io_upload_linear_EDP	Programs the DDMA (via I/O) to upload data using a linear transfer method
76.	cevaxm4_dmss_dma_io_download_message	Programs the DDMA (via I/O) to download data using a message transfer method
77.	cevaxm4_dmss_dma_io_download_1b_wr	Programs the DDMA (via I/O) to download data using a 1bank transfer method
78.	cevaxm4_dmss_dma_io_iit_2d_2d	Programs the DDMA internal 2D-to-2D transfer mode I/O
79.	cevaxm4_dmss_dma_dman_download_2d[4]	Programs the DDMA (via dman) to download data using a 2d transfer method (data in the external memory will be 2d as well)

ID	Test	Description
80.	cevaxm4_dmss_dma_dman_iit_2d_dup2b[4]	Programs the DDMA internal 2D-to-duplicated 2bank transfer mode dman
81.	cevaxm4_dmss_dma_internal_match_upload[4]	• Programs the DDMA to do an upload transfer and set the debug match registers to match on an internal address used by the transfer • Checks that the DDS register is set accordingly and that the dbg_match output pin is set
82.	cevaxm4_dmss_dma_internal_match_download[4]	• Programs the DDMA to do a download transfer and set the debug match registers to match on an internal address used by the transfer • Checks that the DDS register is set accordingly and that the dbg_match output pin is set
83.	cevaxm4_dmss_dma_read_match_iit[4]	• Programs the DDMA to do an iit transfer and set the debug match registers to match on the read address used by the transfer • Checks that the DDS register is set accordingly and that the dbg_match output pin is set
84.	cevaxm4_dmss_dma_write_match_iit[4]	• Programs the DDMA to do an iit transfer and set the debug match registers to match on the write address used by the transfer • Checks that the DDS register is set accordingly and that the dbg_match output pin is set
85.	cevaxm4_dmss_dma_external_match_upload[4]	• Programs the DDMA to do an upload transfer and set the debug match registers to match on an external address used by the transfer • Checks that the DDS register is set accordingly and that the dbg_match output pin is set
86.	cevaxm4_dmss_dma_external_match_download[4]	• Programs the DDMA to do a download transfer and set the debug match registers to match on an external address used by the transfer • Checks that the DDS register is set accordingly and that the dbg_match output pin is set
87.	cevaxm4_gvi_exceptions	Checks the correct behavior of: • Stack violation exceptions • Overflow exceptions When masked, there will not be a notification; when unmasked, the relevant bit is set to 1 in the **DBG_GEN** register.
88.	cevaxm4_mss_apv_check	Checks access protection violation in both the DMSS and PMSS

ID	Test	Description
89.	pmss_iacu_illegal_acc	Checks that illegal access does not affect the IACU, and returns an error response to slave access: ● lock = 0, slv = 0, user0/user1: ○ Checks that CPM WR access does not affect the IACU ○ Checks that an EDAP write attempt does not affect the IACU and gets an error response ● lock = 0, slv = 1: Checks that CPM WR access does not affect the IACU. ● lock = 1: ○ Checks that a supervisor access attempt to the CPM does not affect the IACU ○ Checks that an EDAP write attempt does not affect the IACU and gets an error response
90.	cevaxm4_pmss_wait	● Sets the PMSS wait due to a PMSS read address buffer full indication ● Checks that the program is executed correctly
91.	cevaxm4_dmss_apb3_iop	● Checks the following: ○ Back-to-back external writes, back-to-back external reads, *nop* between each access ○ Back-to-back external writes, back-to-back external reads, internal write between each access ○ Back-to-back external writes, back-to-back external reads, internal reads between each access ○ Back-to-back external/internal read/writes ● Checks that there are no errored transfers
92.	cevaxm4_core_iol	● Checks I/O reads and writes to different registers ● Interleaves reads and writes from the core to PMSS, DMSS, PSU, QMAN, MCCI, OCEM, PROFILER
93.	cevaxm4_pmss_epp_4k_cross(10)	Checks a few PDMA downloads with different burst sizes cross the 4k boundary

ID	Test	Description
94.	cevaxm4_pmss_b0_boundaries(10)	Checks read and write from the first and last PTCM addresses for each PTCM size
95.	cevaxm4_pmss_pcache_boundaries	Checks read and write to the last and first tag indexes of each way of the program cache
96.	cevaxm4_vhist	● VHIST adds 32-bit consecutive accesses to alternate blocks ● VHIST adds 16-bit consecutive accesses to alternate blocks ● VHIST subtracts 32-bit consecutive accesses to alternate blocks ● VHIST subtracts 16-bit consecutive accesses to alternate blocks ● Drives a VHIST access outside TCM (DMBA), and then: ○ Checks the HIST_NO_TCM GVI and M_HIST_NO_TCM GVI masks ○ Checks W0C ● Drives more than two VHIST accesses in any two cycles, and then: ○ Checks the HIST_OV GVI and M_HIST_OV masks ○ Drives another two VHIST accesses in any 2 cycles ○ Check set after set ○ Checks W0C ● Drives VHIST/weight, and then: ○ Checks the HIST_CARRY and HIST_CARRY_MSK registers and the HIST_CARRY GVI and M_HIST_CARRY masks ○ Checks W0C

ID	Test	Description
97.	cevaxm4_dmss_blank_region	• Sets up the start address and blank region for all regions • Reads/writes each region and checks GVI • Sets the DBG_GEN2 rd blank region mask bits • Reads/writes each region and checks No GVI
98.	cevaxm4_dmss_unmapped_exc	• Checks the unmapped memory access exception for both read and write accesses • Sets up Access Protection=111, AP enable bit DCFG_DAPE • Sets the MASK enable bit DBG_xUNMPD_MSK0 • Reads/writes access and checks No GVI[2], GVI[26] • Checks that DBG_xUNMPD0 is not set • Clears the MASK enable bit DBG_xUNMPD_MSK0 • Reads/writes access and checks GVI[2], GVI[26] • Checks that DBG_xUNMPD0 is set
99.	cevaxm4_dmss_edap_ptcm_boot(10)	• Writes to PMEM via EDAP during external_wait • Boot from the PMEM program
100.	cevaxm4_dmss_edap_pmem	• Writes a 32-byte program to block0_pmem_top -32 and executes it • Write a 32-byte program to block0_pmem_top +1 and checks the error response
101.	cevaxm4_dmss_slv0_pmem(7)	SLV0 write to PTCM, and gets error response
102.	cevaxm4_dmss_slv1_pmem(8)	SLV1 write to PTCM, and gets error response
103.	cevaxm4_dmss_slv2_pmem(9)	SLV2 write to PTCM, and gets error response
104.	cevaxm4_pcache_interface	Test toggles the program cache interface signals

ID	Test	Description
105.	cevaxm4_edp_ecc[5]	• Asserts errors on all EDP AXI channels (input ports) • Checks that the error status is correctly reported
106.	cevaxm4_epp_ecc[5]	• Checks single error on input parity bus (Fatal error) • Checks single error on RDATA bus (Correctable) • Checks double error on RDATA bus (Fatal error) • Checks primary output notifications and CPM registers
107.	cevaxm4_axim0_ecc[5][6]	• Asserts errors on all AXIM0 AXI channels (input ports) • Checks that the error status is correctly reported
108.	cevaxm4_axim1_ecc[5][6]	• Asserts errors on all AXIM1 AXI channels (input ports) • Checks that the error status is correctly reported
109.	cevaxm4_edap_ecc[5]	• Asserts errors on all EDAP AXI channels (input ports) • Checks that the error status is correctly reported
110.	cevaxm4_axis0_ecc[5][7]	• Asserts errors on all AXIs0 AXI channels (input ports) • Checks that the error status is correctly reported
111.	cevaxm4_axis1_ecc[5][8]	• Asserts errors on all AXIs1 AXI channels (input ports) • Checks that the error status is correctly reported
112.	cevaxm4_axis2_ecc[5][9]	• Asserts errors on all AXIs2 AXI channels (input ports) • Checks that the error status is correctly reported
113.	cevaxm4_iop_ecc[5]	• Asserts errors on pready, perror, and prdata inputs • Checks that the error status is correctly reported
114.	cevaxm4_dmss_barrier	• Internal barrier activated after external data stores and VHIST instructions are issued • Core halted until all stores are completed
115.	cevaxm4_ports_watchdog[6]	Checks that watchdog violations are asserted when transactions have not completed before the threshold is reached (for EPP, EDP, IOP, AXIM0, AXIM1 ports)

ID	Test	Description
116.	cevaxm4_divstep	Demonstrates a divstep sequence (as well as advance options, such as interrupts and valid bypasses)
117.	cevaxm4_dmss_rmodw	Checks the rmodw instruction to TCM and EDP
118.	cevaxm4_bank_conflict_detection	Checks the VPLD instruction bank conflict detection
119.	cevaxm4_mm3k_compatibility(12)	Checks a basic MM3K (VS2D) compatibility command

Notes:

1. *This test exists only when Memory ECC is installed.*
2. *This test exists only when SPU XTEND is installed.*
3. *This test exists only when VPU XTEND is installed.*
4. *This test exists only when Queue manager is installed.*
5. *This test exists only when ECC_BUS is installed.*
6. *This test exists only when AXI Data Master Ports 0 and 1 exist.*
7. *This test exists only when AXI slave0 is installed.*
8. *This test exists only when AXI slave1 is installed.*
9. *This test exists only when AXI slave2 is installed.*
10. *This test exists only when Program TCM size is not 0.*
11. *This test exists only when EDP WIDTH is 128 bits.*
12. *This test exists only when MM3K compatibility mode is enabled.*

6.2.3 RTT Test List File

If RTT is present in the installation, then an additional set of assembly tests is provided in **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/tlists/cevaxm4_etm_list.**

Table 6-3 describes the RTT tests.

Important: *The tests in Table 6-3 should not be run with the following simulative switches:*

- ***-apb3_ratio***
- ***-axim0_clk_ratio***
- ***-axim1_clk_ratio***
- ***-axis0_clk_ratio***
- ***-axis1_clk_ratio***
- ***-axis2_clk_ratio***
- ***-hclk_ratio***
- ***-main_period***
- ***-tck_period***

Table 6-3: cevaxm4_etm_list Test Descriptions

ID	Test	Description
1.	cevaxm4_rtt_break	Checks the trace of *break*/*breako* instructions with/without delay slots in nested loops
2.	cevaxm4_rtt_cof	Checks the change-of-flow (*br*, *brr*, *brar*, *call*, *callar*, *callr*) and nested loop traces with/without delay slots
3.	cevaxm4_rtt_int0	Checks trace of *int0* in delay slots and loops
4.	cevaxm4_rtt_ints	Checks that *int0*, *int1*, *int2*, int3, *int4*, *vint*, *trap*, *trape*, and *bp* are encoded correctly by the trace wrapper during trace
5.	cevaxm4_rtt_jtag	• Configures wrapper configuration via JTAG • Checks for correct read/write access to wrapper registers

6.2.4 Power Test List File

When simulating power domains in the CEVA-XM4, the simulation test list in **<install_dir>/cevaXM4_V1.1.3.F/simulation/asm/tlists/ cevaxm4_psu_power_list** must be run with UPF.

Table 6-4 describes the power tests.

Important: *The tests in* Table 6-4 *should not be run with the following simulative switches:*

- ***-apb3_ratio***
- ***-axim0_clk_ratio***
- ***-axim1_clk_ratio***
- ***-axis0_clk_ratio***
- ***-axis1_clk_ratio***
- ***-axis2_clk_ratio***
- ***-hclk_ratio***
- ***-main_period***
- ***-tck_period***

Notes:

- *If the power gating option is not selected during installation, then neither the* ***cevaxm4_psu_power_list*** *tests nor the power test list will be present.*
- *For all* ***cevaxm4_psu_power_list*** *tests to pass successfully, the power gating memories must be integrated.*

Table 6-4: cevaxm4_psu_power_list Test Descriptions

ID	Test	Description
1.	cevaxm4_psu_ret_dmss	Checks DMEM blocks in Retention mode
2.	cevaxm4_psu_ret_pmss	Checks PMEM TCM in Retention mode
3.	cevaxm4_psu_ret_pmss_cache	Checks PCache in Retention mode
4.	cevaxm4_psu_deepsleep	Checks Deep Sleep mode and recovery
5.	cevaxm4_psu_shutdown_mem_on	• Checks Shutdown mode of the DSP while the memories are still active • Checks recovery from this mode
6.	cevaxm4_psu_shutdown(1)	Checks Shutdown mode and recovery
7.	cevaxm4_psu_shutdown_pcache	Checks PCache in Shutdown mode
8.	cevaxm4_psu_debug_off	Checks Debug module power off and recovery

Note: *1. This test exists only when Program TCM size is not 0.*

6.2.5 Boot Sequence Tests

This release includes dedicated tests that simulate the process of loading the internal program memory via EDAP to either write directly to the internal program memory, or to configure the program DMA to download code from the external program memory to the internal program memory.

Table 6-5 describes the boot sequence tests.

Table 6-5: Boot Sequence Test Descriptions

ID	Test	Description
1.	cevaxm4_boot_input	Checks boot strap functionality (boot from the vector address located in the external memory)
2.	cevaxm4_boot_swop_ext	Checks the boot sequence when the internal memory is empty at reset and the cache is being updated by the SWOP when running from external memory, in the following stages: 1. *pc* start at external memory (vector) 2. Cache invalidated by the strap pin on boot 3. Configure the cache by SWOP to address 0x1000000 4. *br* to address 0x1000000 5. Core activates the code from the cache
3.	cevaxm4_boot_edap_pdma_core_reset(1)	Checks the boot sequence when the EDAP initializes the PDMA while the core is in reset. The program is downloaded to the PTCM while the core is still in reset. When the download is finished, the core reset is deasserted, and the core starts the execution from the PTCM.

Note: *1. This test exists only when Program TCM size is not 0.*

6.2.6 CPM Register Test Files

This release includes dedicated tests to simulate read and write accesses to the CPM registers.

Table 6-6 describes the CPM register tests.

Important: *The tests in* Table 6-6 *should not be run with the following simulative switches:*

- ***-apb3_ratio***
- ***-axim0_clk_ratio***
- ***-axim1_clk_ratio***
- ***-axis0_clk_ratio***
- ***-axis1_clk_ratio***
- ***-axis2_clk_ratio***
- ***-hclk_ratio***
- ***-main_period***
- ***-tck_period***

Table 6-6: CPM Register Test Descriptions

ID	Test	Description
1.	reg_all_cpm_edap	Tests read and write access to the CPM register through the core and EDAP
2.	reg_all_package_slv0(1)	Tests read and write access to the CPM register through axi_slv0
3.	reg_all_package_slv1(1)	Tests read and write access to the CPM register through axi_slave1
4.	reg_all_package_slv2(1)	Tests read and write access to the CPM register through axi_slv2

Note: *1. This test only exists when the relevant axi_slave<n> is installed.*

7. Simulative Modules

The **cevaxm4_sim_top** module contains an extensive simulation environment for the CEVA-XM4, including clock generation, simulation control logic, checkers and analyzers, and instantiations of simulative modules that act as hosts that connect to the CEVA-XM4 interface.

The following sections describe the usage and functionality of the simulative modules in the CEVA-XM4 simulation environment.

7.1 AXI Slave Host

Each AXI Slave Host contains two blocks:

- **axi_host_ctl:** The host control block handles the interface and generates the signals to the memory.
- **axi_host_memory:** The memory block is a simulative memory.

The responses and two-bit *rresp* (in a read transaction) or *bresp* (in a write transaction) values for AXI transactions to the slave hosts (not from the Master host) within certain address ranges are listed in Table 7-1.

Table 7-1: AXI Transaction Responses

AXI Transaction to the Slave Hosts (Not from the Master Host) to Address Range	Response	Two-Bit *rresp* or *bresp* Values
0xffff_1200 - 0xffff_12ff	**SLV_ERR**	10
0xffff_5600 - 0xffff_56ff	**DEC_ERR**	11
0xffff_9A00 - 0xffff_9Aff	**EXOKAY**	01

Figure 7-1 shows the AXI Host flow.

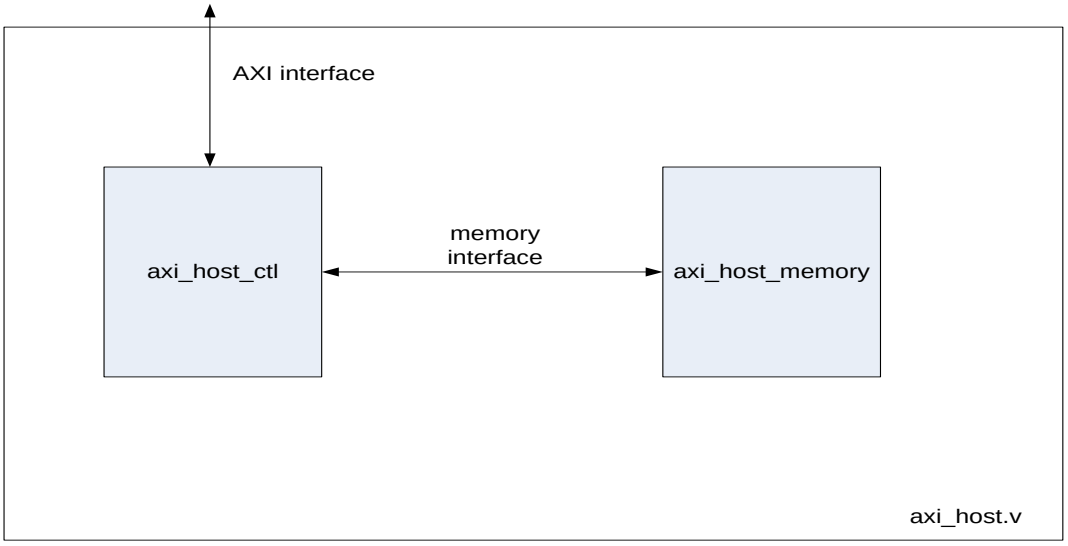


Figure 7-1: AXI Host Block Diagram

7.1.1 AXI Slave Host – Ready and Valid Control

The *awready*, *wready*, and *arready* signals can be controlled via Siggen. In addition, the delay pattern of data read from the host can be controlled by *rvalid_pattern*, which is a 64-bit number that is composed of 16 nibbles of 4-bit length. Each nibble represents the delay for a transfer in a read burst. The response pattern of the first transfer is described by the hexadecimal value of the 4 LSBs of the *siggen_rvalid_pattern* signal, as shown in Example 7-1.

Example 7-1: siggen_rvalid_pattern signal

```
.EQU ADDRESS_EXT1          #0x1234500E
;{{ siggen_arready -v 1'b0 -ve 1'b1 -m 12}}
SC0.mov ADDRESS_EXT1, r10.ui  ;{{ siggen_arready -v 1'b0
-ve 1'b1 -m 12}}
SC0.nop                       ;{{ siggen_rvalid_pattern -
v 64'h3210}}
PMEM_PADDING_2_FETCH_LINES


## Store data on external memory
LS0.st r1.ui, (r10.ui).di
```

The ASM code in Example 7-1 will result in a read data return with zero delay for the first transfer, a one-cycle delay for the second transfer, a two-cycle delay before the third transfer, a three-cycle delay before the fourth, and zero delay for any further transfers. For a pattern to take place, a read request must be issued to any AXI slave host.

7.2 AXI Master Hosts

The four AXI Master host-initiated transactions are controlled by the control registers in the AXI Master host, as described in Table 7-2. The internal AXI Master host registers are accessed via the core APB I/O interface. The core can configure the AXI Master host using in/out instructions via these registers.

Based on the configuration, the AXI Master host initiates an AXI transaction either towards the EDAP or towards one of the AXI slaves (AXIs0-AXIs2). The AXI slave translates the transfer into an internal CEVA-XM4 memory transaction, and performs it when it wins arbitration to the memory.

The AXI master outputs the *active_slvPort_axi_master* signal that is set when a transaction is under progress and the AXI Master is busy. This port can be used with the Siggen utility (as described in Section 8) to reflect this information into a register and use it in ASM tests (polling).

Note: *Due to the usage of assignment delay (#delay) in the AXI hosts/master, it is mandatory to use a timescale directive when the time precision is less than or equal to 1/100 of the time unit (for example, timescale 1ns/10ps). Otherwise, the simulative hosts will not work properly.*

Table 7-2: AXI Master Host APB3 Address Map

Base Address	Name	Description
0x00000000	EDAP_MASTER	AXI Master connected to EDAP
0x0000f000	AXIS0_MASTER	AXI Master connected to AXI slave 0
0x0000f100	AXIS1_MASTER	AXI Master connected to AXI slave 1
0x0000f200	AXIS2_MASTER	AXI Master connected to AXI slave 2

Table 7-3: AXI Master Host APB3 Address Offset

Address Offset	Name	Description
0x00000068	AXI_MASTER_START_REG	32 bits, for launching a group of tasks
0x0000006c	AXI_MASTER_STATUS_REG	32 bits, to hold the received status of the EDAP in the AXI Master
0x00000070	AXI_MASTER_WRITE_DATA	Data to be written by the AXI Master to the EDAP
0x00000074	AXI_MASTER_WRITE_ADDRESS	Address for the AXI write address channel
0x00000078	AXI_MASTER_READ_ADDRESS	Address for the AXI read address channel
0x0000007c	AXI_MASTER_READ_CTRL	Controls for the AXI read address channel
0x00000080	AXI_MASTER_WRITE_CTRL	Controls for the AXI write address channel
0x00000084	AXI_MASTER_LAUNCH_TASK	Master initiates the task
0x00000088	AXI_MASTER_READ_DATAL_ REG	Address for reading the low part of the AXI read data (AXI Master hosts only)
0x0000008C	AXI_MASTER_READ_DATAH_ REG	Address for reading the high part of the AXI read data (AXI Master hosts only)

7.2.1 AXI Master Test Flow

7.2.1.1 Simple Write Task

The following is the flow of a simple write task:

1. Write the destination address to the **AXI_MASTER_WRITE_ADDRESS** register (address 0x74).
2. Write the data bus (64 or 128) as 32-bit parts to the **AXI_MASTER_WRITE_DATA** register (address 0x78), in little endian (bits 31 to 0 first).

   This step must be repeated based on the burst length and the data bus size. For example, if the burst is **1** (two transactions), and the AXI data bus size is **128**, the **AXI_MASTER_WRITE_DATA** register must be written eight times.

3. Write the control bits to the **AXI_MASTER_WRITE_CTRL** register (address 0x80), as described in Table 7-4.

Table 7-4: Control Bits

Bits	Description
7:0	Length (how many transactions the read/write should last)
11:8	Size (which bits of the bus are actually used for the transaction
15:12	Burst type (fixed/incremental/wrap)
31:16	ID

4. Start the task by writing the value **zero (0x0)** to the **AXI_MASTER_TASK_LAUNCH** register (address 0x84).

7.2.1.2 Simple Read Task

The following is the flow of a simple read task:

1. Write the source address to the **AXI_MASTER_READ_ADDRESS** register (address 0x78).
2. Write the control bits to the **AXI_MASTER_READ_CTRL** register (address 0x7c), as described in Table 7-4.
3. Start the task by writing the value **one (0x1)** to the **AXI_MASTER_TASK_LAUNCH** register (address 0x84).

7.2.1.3 Write-and-Read Task

The following is the flow of a write-and-read task:

1. Configure the APB3 registers related to the AXI as described in Sections 7.2.1.1 and 7.2.1.2 (write address, write data, write control, read address, and read control).
2. In the last step, start the task by writing the value **two (0x2)** to the **AXI_MASTER_TASK_LAUNCH** register (address 0x84).

7.2.1.4 Write from Saved Data Task

This task writes like the simple write task described in Section 7.2.1.1, except that the data was saved in a former read task, and there is no need for data saving before (that is, no writing to the **AXI_MASTER_WRITE_DATA** register is required). The flow of the self-check is as follows:

1. Write the internal memory through ASM
2. Read the internal memory through AXI Master – EDAP interface
3. Write back the data saved through the AXI Master – EDAP interface via the "writing saved data task".
4. Start the task by writing the value **three (0x3)** to the **AXI_MASTER_TASK_LAUNCH** register (address 0x84).

   The control signals (length, size, and burst type) must be the same in both read and write. In other words, *arlen*, *arsize*, and *arburst* must be equal to *awlen*, *awsize*, and *awburst* afterwards (*arid* can be different from *awid*). If they are not, the test will not work properly.

7.2.1.5 Group of Tasks

To avoid initiating the task after writing to the **AXI_MASTER_TASK_LAUNCH** register, do the following:

1. Set bit **0** of the **AXI_MASTER_START_REG** register (address 0x68)
2. Write all of the launch registers with the tasks
3. Write the value of **three** (**0x3**) to the **AXI_MASTER_START_REG** register.

   All of the pending tasks are launched.

7.2.1.6 Delay Control

The delay inside a burst between transfers can be controlled by the *wvalid_pattern* Siggen, which is a 64-bit Siggen where each nibble sets the delay of one transfer. In addition, the delay between the address and data channels on a write task can be controlled via the *write_channel_delay* Siggen. A positive value will delay the data with respect to the address, and a negative value (in 2's complement representation) will delay the address with respect to data, as shown in Example 7-2.

Example 7-2: Delay Control

```
.EQU WRITE_CTRL_REG 0x00000080;
.EQU WRITE_TASK      0x0
.EQU WRITE_ADDRESS_REG 0x74
.EQU LAUNCH_TASK_REG 0x00000084;


SC0.mov #WRITE_CTRL_REG,   r5.ui ;{{write_channel_delay -v 8'hFB
}}
SC0.mov #WRITE_TASK,       r6.ui ;{{siggen_wvalid_pattern -v
64'h4321}}
SC0.mov #LAUNCH_TASK_REG,  r7.ui ;{{uia1 -f active_axi_master  -v
1'b1 }}
SC0.mov #WRITE_ADDRESS_REG,r8.ui


;## (id = 0x1234: burst: 1 = inc, Size: 16 =  100 128bit,
;## len: 11 = 4 transfers)
SC0.mov #0x12341403,  r4.ui;
SC3.nop
SC0.out r4.ui , (r5.ui).di; storing first write ctrl
SC0.out r10.ui, (r8.ui).di; storing write address
SC0.out r6.ui , (r7.ui).di;
```

The ASM code in Example 7-2 will result in a write burst of four transfers with a one-cycle delay for the first transfer, a two-cycle delay for the second transfer, a three-cycle delay before the third transfer, and a four-cycle delay before the fourth. The address channel valid will rise five cycles after the data channel valid rise.

7.2.2 Verilog Module Structure

The AXI Master module is built from register logic and Verilog tasks. The registers are programmed by software, using the core's I/O APB port. Based on the configuration, the Verilog tasks that initiated an AXI transfer are performed.

Note: *The AXI hosts/Master modules should not be synthesized.*

7.3 AXI Loopback

An AXI loopback is implemented in the simulation environment so the AXI slave ports can be controlled by the AXI Master ports of the CEVA-XM4. The available connectivity is between:

- EDP master port to EDAP slave port (AXI bus width 128 bits only)
- EDP master port to AXIs2 slave port (AXI bus width 256 bits only)
- AXIm0 Master port to AXIs0 slave port
- AXIm1 Master port to AXIs1 slave port

The AXI loopback is controlled by the *axi_host_sel* control signal.

When the AXI host select signal is set, the slave ports are connected to the simulative master host modules. This is the environment default.

When the AXI host select signal is not set, all CEVA-XM4 AXI Master port transactions are mapped to the corresponding slave port.

The AXI host select signal is controlled by the following Siggen utility (as described in Section 8):

```
;#################################
;## setup the axi loopback
;{{axi_host_sel  -v 1'b0 }}
;#################################

;#################################
;## turn off the axi loopback – setup axi host
;{{axi_host_sel  -v 1'b1 }}
;#################################
```

7.4 APB3 Slave Host

The APB3 slave host module interfaces with the CEVA-XM4 I/O port APB3 interface. The slave contains eight 32-bit registers that can be accessed by the CEVA-XM4. The user can write to, or read from, the registers in the external APB3 slave host by issuing in/out instructions (which are translated in the MSS to valid APB3 transactions).

The CEVA-XM4 I/O port APB3 interface is also used to configure and control the AXI Master host, APB3 Master host, and Xtend host. The addresses assigned to these hosts are listed in Table 7-5. All other addresses are assigned to the APB3 slave host.

In the APB3 slave, only the three LSBs of the address are used to decide which of the eight (8) registers are accessed.

If the address 0xffff_ffff is accessed (for read or for write), then an error indication is asserted (pslverr set).

In Example 7-3, the data 0xdeadbeef is written to the register in address 0x2 on the APB3 address space:

Example 7-3: APB3 Slave Host

```
SC0.mov #0xdeadbeef, r0.ui
SC0.mov #0x2, r1.ui
SC0.out r0.di, (r1).di
```

Table 7-5: APB3 Address Map

Address	Name	Description
0x00000000 – 0x00000064	APB3 slave host	8 x 32-bit general read/write registers
0x00000068 – 0x0000008C	AXI Master host registers	See Section 7.2
0x0000f068 – 0x0000f08C	AXI Master host connected to slave 0	See Section 7.2
0x0000f168 – 0x0000f18C	AXI Master host connected to slave 1	See Section 7.2
0x0000f268 – 0x0000f28C	AXI Master host connected to slave 2	See Section 7.2
0x70000001, 0x70000002, 0x70000003, 0x70001001	APB3 Master host queues	See Section 7.5
0x000000A8 – 0xFFFFFFFE	APB3 Slave host	8 x 32-bit general read/write registers
0xFFFFFFFF	APB3 Error address	When the address is accessed, an error indication is asserted

7.5 APB3 Master Host

The APB3 Master host simulates an external APB3 Master that initiates transactions towards the On-Chip Emulation (OCEM) using the OCEM's APB3 interface.

The APB3 Master host-initiated transactions are controlled by the queues in the APB3 Master host, as described in Table 7-6. The APB3 Master host queues are accessed via the core APB I/O interface. Through these queues, the core can configure the APB3 Master host using in/out instructions.

Based on the configuration, the APB3 Master host initiates an APB3 transaction towards the OCEM APB port, which translates the transfer into a scan-chain that is then executed by the OCEM.

The APB3 master has the following operating modes:

- **Host Passive Mode**: When accessed via an I/O interface by the core
- **Host Activate Mode**: Initiates transactions to the OCEM and receives data from it

  The OCEM will accept the transaction only if the *jt_ap* signal, which is an input to the CEVA-XM4, is set.

Important: *These two different modes should not overlap.*

The APB3 Master host has several FIFO queues in order to store the required transactions. The queues' unique addresses in the I/O space are described in Table 7-6.

Table 7-6: APB Master Host Queues Address Map

Queue	Address	Description	Parameters
Address queue	• For write: 0x70001001 • For read: 0x70000001	Saves the address to access the OCEM	APB_MASTER_ADDR_Q_W APB_MASTER_ADDR_Q_R
Read/write queue	The read/write queue is filled along with the address queue, as follows: • If the 0x70001001 address was used, it is filled with 0x1. • If the 0x70000001 address was used, it is filled with 0x0.		
Write data queue	0x70000002	Saves the data to write to OCEM	APB_MASTER_WDATA_Q
Read data queue	0x70000003	Saves data read from OCEM	APB_MASTER_RDATA_Q

Notes: *Writing to the read/write and the address queues is done in parallel. The value written to the read/write queue depends on which address was used for the address queue (as described in the table).*

After every access to the address queue, there must be access to the write data queue even if the address written to the address queue is configured to be for a read transaction (written through address 0x70000001).

The last out transaction must execute before activating the APB3 Master host.

There are two ways to enter activated mode, both via the Siggen utility:

- **With Debug Mode**: Assert *activate_ocem_apb_debug* (using Siggen). The APB Master host mode is changed to activate mode only when the core enters Debug mode (that is, the core jumps to *trape* address and then receives a wait signal).

  To exit this mode, deassert *activate_ocem_apb_debug* or exit Debug mode.
- **Without Debug Mode**: Assert *activate_ocem_apb_without_debug* (using Siggen). In this mode, the core pipeline continues progressing while the APB3 Master host sends transactions to the OCEM.

  To exit this mode, deassert *activate_ocem_apb_without_debug*.

When the host is in activate mode, the APB3 Master starts sending transactions either until the address queue is depleted or until the host enters normal mode, whichever comes first.

Table 7-7: APB Master Host Example Usage

Macros	Description
WRITE_OUT_MEM APB_MASTER_ADDR_Q_W, address WRITE_OUT_MEM APB_MASTER_WDATA_Q, data	Write to APB3 Master host write transaction (in host passive mode): ● Write (*out* instruction) the requested address to the address queue and to the write queue ● Write (*out* instruction) the data to be written to the data queue
WRITE_OUT_MEM APB_MASTER_ADDR_Q_R, address WRITE_OUT_MEM APB_MASTER_WDATA_Q, data	Write to APB3 Master host read transaction (in host passive mode): ● Write (*out* instruction) the requested address to the address queue and to the read queue ● Write (*out* instruction) the data to be written to the data queue (the data can be dummy)

7.6 OCEM Host

The OCEM is typically operated by an external debugger, using the standard JTAG interface. The OCEM host simulates the external debugger, which enables the verification of the various features of the OCEM.

When running an OCEM test, the OCEM host should generate a sequence of JTAG transactions. Each test has a unique transaction sequence associated with it. The sequence appears as a remark section within the assembly test.

The simulation script extracts the OCEM host sequence from the assembly test, and then loads it to the host internal memory. The sequence contains codes that stand for instructions and data that the host uses. The host reads these codes and translates them to JTAG transactions.

For example, the following sequence is used to write to an OCEM register:

- A code that indicates which register is accessed, including the control code of the register that is being accessed.

  This control code is loaded to the JTAG instruction register in the OCEM. For more details about the control codes of the OCEM registers and scan chains, see the *CEVA-XM4 On-Chip Emulation Reference Guide*.
- The data that should be written to the register.

The host is made up of the following modules:

- **ocem_host**: Contains all host registers, wires, logic, and connectivity
- **jtag_tasks**: A submodule of the ocem_host; contains tasks that create the JTAG transactions

Figure 7-2 shows the OCEM host structure.

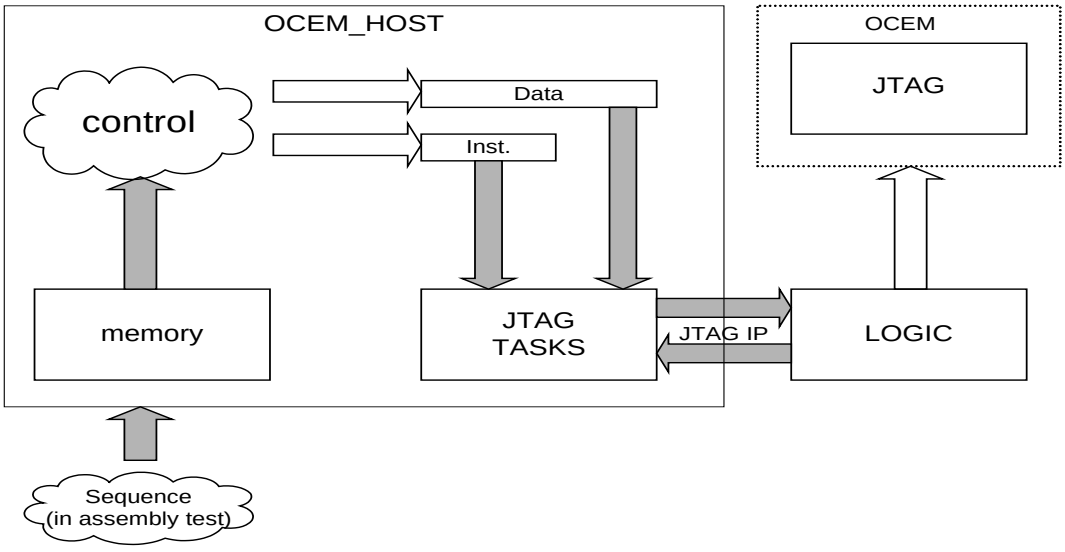


Figure 7-2: OCEM Host Structure

7.7 Analyzer Modules

The CEVA-XM4 analyzers are checkers that detect illegal behavior of the CEVA-XM4. These checkers are instantiated in the top-level simulation module (**cevaxm4_sim_top.v**). The analyzer modules check the validity of various signals and events that cannot be monitored by the self-check mechanism during the simulation.

Each analyzer has multiple inputs and one output that is an error indication. If a failure is detected by an analyzer, the error bit is set, the corresponding error message is reported on the screen and written into the report file (under the **work** directory), and the test operation is stopped. Each analyzer has an enable input pin. These can be disabled via the **-disable_analyzers <"analyzer_name">** script switch.

Note: *The analyzer modules should not be synthesized.*

Table 7-8 describes the CEVA-XM4 analyzers.

Table 7-8: CEVA-XM4 Analyzers

Analyzer	Analyzer Name	Description
cevaxm4_memory_ interface_an.v	MEMORY_INTERFACE_ AN	Monitors all of the I/Os that form the memory interface of the DSP core: ● The program memory interface ● The data memory interface ● The MMIO interface It is designed to detect undefined values (X or Z) on the memory interface buses and control signals. The input buses from all memories are monitored only when a read access is performed. All other signals and buses are constantly monitored throughout the simulation.
cevaxm4_axi_if_an.v	● EPP_AXI_AN ● EDP_AXI_AN ● EDAP_AXI_AN	● Checks that the CEVA-XM4 AXI hosts (DMSS and PMSS) perform the correct AXI transactions (based on the AXI protocol). ● Checks that: ○ Control signals are not X/Z when the valid signals are high ○ Size is not wider than the AXI data bus width ***Note****: This analyzer has three instantiations: EDP, EPP, and EDAP.*

Analyzer	Analyzer Name	Description
ceva_verifeq_an.v	VERIFEQ_AN	Checks that there are no Xs or Zs on the *verifeq* signals. The input signals give error indications from the test. If the signals are X or Z, the test might pass when it actually should fail.
cevaxm4_apb3_an.v	EPB3_AN	Checks that the APB3 port does not have X/Z when the signals are valid (under *psel*, *pwrite*, and so on)
cevaxm4_if_an.v	IF_AN	Checks that the interface output signals do not have X/Z
tpiu_monitor.v	TRACE_MONITOR_AN	Does the following: 1. Takes data from the core, aligns it, and puts it into a FIFO. 2. Compares the data to the trace output from the wrapper. 3. Decodes the raw trace output from the TPIU and compares it with the wrapper trace output. Any mismatches between either the core and the wrapper, or the wrapper and the TPIU, cause the error signal to be set.

Note: *The tpiu_monitor.v analyzer exists only if the customer acquires TRACE.*

7.8 Clock Generation Module

The CEVA-XM4 clock generation module, **clk_gen.v**, generates clocks for simulation purposes. The module is instantiated in the top-level simulation module (**cevaxm4_sim_top.v**).

Note: *The clk_gen.v modules should not be synthesized.*

Table 7-9 describes the CEVA-XM4 clocks that are generated by the **clk_gen.v** module.

Table 7-9: CEVA-XM4 Clock Generation

Clock	Description
ceva_clk	Main simulation clock
ceva_dly_clk	The clock delayed by an insertion delay for SDF simulation
hclk	Bridge master clock
div_en	Division enable for hclk
clk	Clock with wait-states
clk_siggen	Siggen clock
clk_mask_siggen	Siggen mask clock
internal_tck	TCK clock (to ocem_host)

7.9 CEVA-Xtend VU Host

The CEVA-Xtend host simulates the Xtend hardware functional unit. It contains logic that simulates all of the possible transactions between the Xtend hardware device and the CEVA-XM4, as described in the *CEVA-Xtend Architecture Specification*.

Each Xtend instruction can be extended by a 26/10-bit extension.

7.10 ETM Module Integration

Figure 7-3 shows the simulation environment structure and the simulation components used for RTT simulation when there is an **internal** ETM installation.

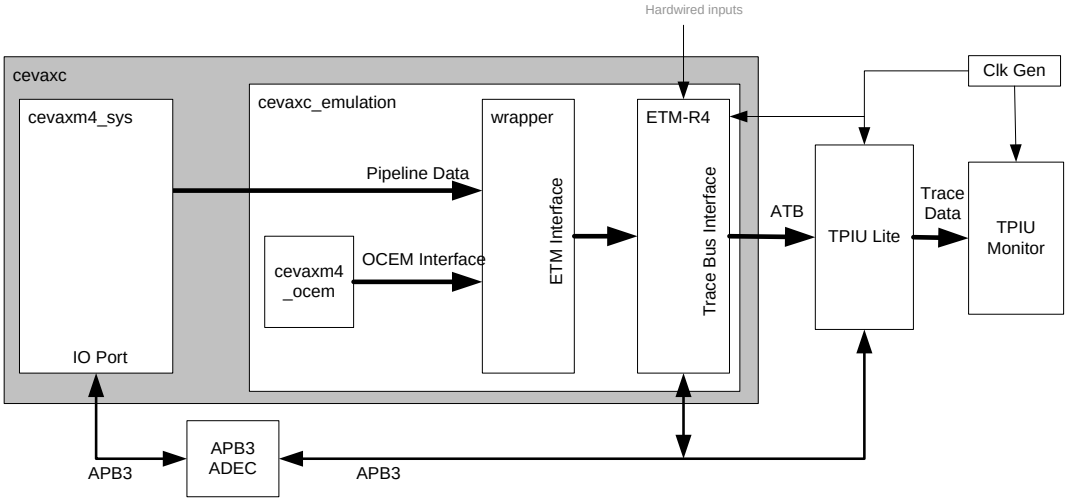


Figure 7-3: Integration of RTT Related Modules (Internal ETM)

Figure 7-4 shows the simulation environment structure and the simulation components used for RTT simulation when there is an **external** ETM installation.

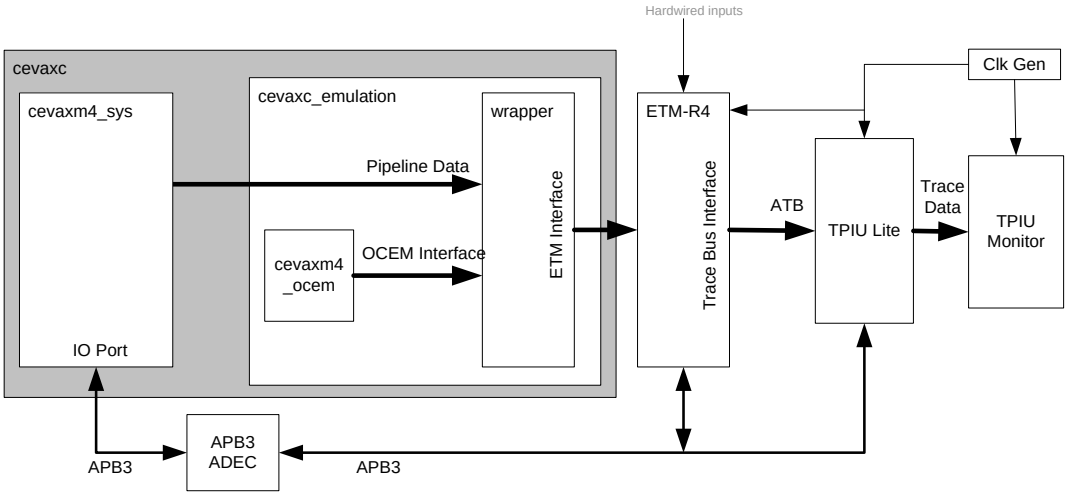


Figure 7-4: Integration of RTT Related Modules (External ETM)

Notes:

- *The APB3 ADEC connects the CEVA-XM4 I/O port to allow the configuration of the ETM_R4 and TPIU modules by the CEVA-XM4 core.*
- *The ClkGen module generates the clocks for the ETM_R4, TPIU, and TPIU monitor.*
- *The TPIU module converts the raw trace data into a format that can be analyzed more easily.*
- *The TPIU monitor checks the trace output at different stages for equivalence.*

8. Siggen Utility

8.1 Introduction

Siggen is a CEVA in-house tool used for forcing signal values in the Verilog top-level module during simulation, such as inputs to the Design Under Test (DUT). The input to the tool is an **.lst** file that contains an indication of the specific signals/buses to be generated and inserted.

Events are defined in the text of the assembly test. This information is written as a remark (after a semicolon) as follows:

```
; {{SIGGEN EVENT DEFINITION}}
```

Each external event happens either to a rising or falling edge, or to a value of the specific signal selected.

The simulation script searches in the ***.lst** file (the output from the assembler) for the Siggen commands. These commands are translated into Verilog events that are combined in the Verilog simulation, which provides an efficient and convenient method to toggle chip inputs during the simulation of the test.

Each test case is assigned a Siggen Verilog task that is called in the **cevaxm4_sim_cntrl.v** simulation control unit when that test case is simulated.

Note: *Only signals generated in the **cevaxm4_sim_cntrl** module can be driven by Siggen. This means that only core inputs can be modified and not internal signals of the core.*

8.2 Siggen Commands

Siggen commands are entered as comments in the **<test_name>.asm** assembly file using the following syntax:

```
; {{<signal name> -<switch#1 value#1> -<switch#2
value#2> … }}
```

where <**SIGNAL NAME**> is the Verilog signal name (or bus name) of the signal (or bus) to be modified. The switches define the timing and the duration of the modification.

Note: *If there are two remark characters (;;) at the line with the Siggen command, the Siggen command is disregarded. This provides a useful way to disable the Siggen command.*

8.2.1 Siggen Switches

8.2.1.1 Trigger Switches

Table 8-1: Trigger Switches

Switch	Description
-r <reference_signal>	Trigger event is rising edge of reference_signal.
-f <reference_signal>	Trigger event is falling edge of reference_signal.
-com <reference_signal> <value>	Trigger event is when reference_signal equals the value (value can be a number or a signal).

When no switch is used, the trigger event is the *pc* of the instruction next to which the Siggen was written.

8.2.1.2 Delay Switches

Table 8-2: Delay Switches

Switch	Description
-df <number>	Wait number of free clock cycles (ceva_clk) before modifying the signal
-dm <number>	Number of core clock cycles before modifying the signal
-n <number>	Wait number of triggering events before modifying the signal

Note: *To determine the delay in the assertion of a signal, it is recommended to use the* ***-dm*** *switch, which indicates the delay in machine cycles.*

If the ***-d*** *option is used, then it is better to define a Verilog real value of* ***clock_period*** *in the test bench and assign the Verilog time unit delay value to be a division of this* ***clock_period*** *(that is,* ***-d (clock_period*2/5)****). If* ***clock_period*** *is set to be equal to the core clock period, then the* ***-d*** *delay is a multiple of the clock period for all frequencies of operation of the DUT.*

8.2.1.3 Hold Switches

Table 8-3: Hold Switches

Switch	Description
-m<number>	After modifying the signal to the start value, wait number of core clock cycles before modifying the signal to the end value
-mf<number>	After modifying the signal to the start value, wait number of free clock cycles before modifying the signal to the end value

8.2.1.4 Value Switches

Table 8-4: Value Switches

Switch	Description
-v<start_value>	Start value switch; the signal is modified to this value first
-ve<end_value>	End_value for use with **-m** and **-mf**

If no value is specified, the start value assigned is **1** for active high signals, and **0** for active low signals (a signal name that ends with the letter **n** indicates an active low signal). Accordingly, the end value is **0** for active high signals, and **1** for active low signals.

Example 8-1: Value Switch

```
SC.nop; {{external_wait -r start_pulse -d 100 -m 5 -v
1'b1 -ve 1'b0}}
```

In Example 8-1, *external_wait* is 1'b1 100 ns after the rising edge of *start_pulse*, it stays high for five core clocks, and then it receives 1'b0.

8.3 Siggen Utility Register File Module

The Siggen utility register file module can register values in its registers based on Siggen events. The registers are readable by an in instruction through the APB3 slave host.

The inputs to the register file (**log_data0** – **log_data15**) are controllable through the following Siggen events:

```
nop; {{log_data4 -r bman_sniffer_ext_inc[0] -n 1 -v
32'hffffffff }}
```

The **log_data0** – **log_data15** inputs are sampled in the register file by respective **reg0** – **reg15** registers every simulation cycle.

For the core to read the registers of the Siggen register file, the *cevaxm4_siggen_rf_en* simulation signal should be asserted, and the *cevaxm4_siggen_rf_addr* simulation signal should get the number of the register to be read. Both signals are controllable through Siggen events.

For example, to enable reading of the **reg4** register, which samples **log_data4**, the following Siggen events can be used:

Example 8-2: Siggen Events

```
nop; {{ cevaxm4_siggen_rf_en -v 1'b1 }}
nop; {{ cevaxm4_siggen_rf_addr -v 4'h4 }}
```

The registers can then be read by the core using the following *in* instruction:

```
SC0.mov #0x0,r3
SC0.in{dw} (r3), r2
```

Note: *When the cevaxm4_siggen_rf_en simulation signal is asserted, the APB3 slave host register addressed by 0x0 cannot be written by the core, which sees it as a read-only register.*

9. Adding a New Test Case

The following sections describe how to create a new test assembly file (for example, named **mytest.asm**) in the simulation environment. The new test assembly file should be located in the **tests/asm** directory.

The test file structure is as follows:

- Test Header
- General Files
- Test Body
- Test Termination

9.1 Test Header

The test header should include the test name and description, for example:

Example 9-1: Test Header

```
;** Test name:  mytest.asm
**

;** Test Description:
**

;**    Example to assembly test case form
**
```

9.2 General Files

Some general files are included in all tests. These files declare the common parameters and macros of the tests, as follows:

Example 9-2: General Files

```
.INCLUDE "cevaxm4_macros.asm"
.INCLUDE "cevaxm4_param.asm"
.INCLUDE "cevaxm4_program_cache_conf.asm"
.FORMAT 1000,500
```

Notes:

- *Included files are located in the **tests/include/** directory.*
- *When including a file, only the fil name is specified, without the path name.*
- *Nesting included files is forbidden.*
- *Included files can be added when there are routines that are shared among different tests.*

9.3 Test Body

The test body is written using CEVA-XM4's native assembly. All of the parameters/macros are defined either in the test or in the included files.

9.4 Test Termination

A test can be terminated either due to an error or in a successful way via the *verifend* instruction.

Example 9-3: Test Termination

```
PCU.verifend
PMEM_PADDING_4_FETCH_LINES
PMEM_PADDING_4_FETCH_LINES
```

Note: *The **PMEM_PADDING_4_FETCH_LINES** macro adds four nop fetch lines to the test. This adds an actual value (nop opcode) to the program code to protect the core from any penetration of undesired Xs.*

It is recommended to add these lines after any branch-type command.

9.5 Writing a Test List File

The test list file should be located in the **tlists** directory (for example, **tests/tlists/all_mytests_list**; no extension is needed). It should be of the form:

```
mytest1
mytest2
mytest3
```

Note: *Further test cases can be added to the test list by placing the test name (without the **.asm** file extension) in this file. Running all of the test cases in the list is possible via the **-list <list _name>** switch in the **ceva_sim** script.*

10. Loading Data to the Data Memory

You can load data directly to the data memory using the **-dbg_file <file_name>** switch. The input file should be in the same format as the **debugger .dbg** file.

The script interprets the following debugger commands:

- **fill [d:start_address, number_of_elements] 0**: Fills the memory with zeros from *start_address* (hexadecimal) and on with *number_of_elements* (decimal) zeros.
- **copy "input file" [d:start_address]**: Loads the data from the input file to the data memory beginning from *start_address* and on.
- **copy [d:start_address, number_of_elements] "output file**": Dumps *number_of_elements* (decimal number) data words from the data memory beginning from *start_address* (hexadecimal) and on.

Example 10-1 and Example 10-2 illustrate this.

Example 10-1: Debugger Commands

```
fill [d:100,400] 0 # Will fill the internal data memory
with 400 zero words beginning with address 0x100

copy input.in [d:200]# Will load the internal data
memory with data from input.in file.

copy [d:200:400] output.out # Dump 400 words starting
from address 0x200.
```

Example 10-2: Input Data Memory File

```
0x0002

0x0001

0xFFFC

0xFFFF

0xFFFD

0xFFFF
```

If you want to compare the dumped data from the memory to a reference file, you can use the **-benchmark_ref_file <reference file>** switch. After the test ends, the data that was dumped from the memory is compared to this reference file. A success indication is a complete identification of both files (the dumped data output file and the reference file). If the files are not matched, a table with the mismatches is displayed on the screen.

Notes: *The user can load more than one memory file into different locations in the internal data memory by using the copy command more than once. The same is true for the fill command.*

There is no need to load data that is in the test data section (DSECT).

Example 10-3 illustrates this for a test named **try**:

Example 10-3: Debugger Commands (try Test)

```
copy vY.hex [d:20000,512]      ; load input data to 'Y';
copy vH.hex [d:20200,1024]     ; load input data to 'H';
copy vomega.hex [d:20600,256]  ; load input data to
'omega';
go
copy [d:110B4,512] Xhv1.out
```

The first three instructions are to copy the initial data to the memory. The **ceva_sim** script executes those in the begging of the run.

The fourth instruction is to copy from the memory to an output file. The script executes this instruction at the end of the run.

To run this, type:

```
ceva_sim -t  try  -dbg_file try.dbg -record
```

11. RTL Order of Compilation

Table 11-1 lists the order in which the RTL design, simulation, and Designware files are compiled.

Note: *Not all files are required for all CEVA-XM4 configurations.*

Table 11-1: RTL Order of Compilation

File #	Filename
1	./envdef.v
2	$CEVAXM4_RTL_LOCAL_ROOT/top/conf2_cevaxm4_gendef.v
3	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_top_rtl_def.v
4	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/cevaxm4_crand_delay_def.sv
5	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_ctl.v
6	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_adec_byte.v
7	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pir_placement_256.v
8	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla15.v
9	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_rd_aif.v
10	$CEVAXM4_RTL_LOCAL_ROOT/top/profiler/cevaxm4_prf_counter.v
11	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_core_tree_ls.v
12	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_ext_rd_proc.v
13	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vlogic.v
14	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_pmss_cpm.v
15	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rd_buf_bs.v
16	$CEVAXM4_RTL_LOCAL_ROOT/top/cevaxm4_emulation.v
17	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_task_proc_fsm.v
18	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_find_tag_parity.v
19	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf_reg_wr.v
20	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_fifo.v
21	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/general/cevaxm4_5to32_enc.v
22	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_cmp_swap.v
23	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_hist_rf.v
24	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_l2d_z2.v
25	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dpm.v
26	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pslot_decoder.v
27	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pdispatch_256_single.v

File #	Filename
28	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_s2d_sat.v
29	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_mult.v
30	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_sum.v
31	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pfetch.v
32	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_hist_wad.v
33	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_dvm_break.v
34	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pseq_trace.v
35	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_mov.v
36	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_rab_rxfsm.v
37	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_line_int_byte_sel.v
38	$CEVAXM4_RTL_LOCAL_ROOT/top/conf2_cevaxm4.v
39	$CEVAXM4_RTL_LOCAL_ROOT/top/cevaxm4_dmem.v (Can be replaced by the user to the real memories top file)
40	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edp_ls_rd.v
41	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_int2fp.v
42	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_exception_det.v
43	$DW_LOCAL_ROOT/DW01/DW_lbsh.v
44	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table1_sqrt_odd_exp.v
45	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/daau_decoders/cevaxm4_dls_d2_decoder.v
46	$DW_LOCAL_ROOT/DW01/DW_rash.v
47	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gmux_split_one_hot_cfg.v
48	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/cevaxm4_daau_top.v
49	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table0_sqrt_inv_odd_exp.v
50	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_dman_cpm_qman_bman_cntl.v
51	$CEVAXM4_RTL_LOCAL_ROOT/top/cevaxm4_iop.v
52	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_alu.v
53	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_eos.v
54	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_core_tree.v
55	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_mul_calc.v
56	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_tsts.v
57	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_vcu_sample.v
58	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_axi_master.v
59	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla6.v
60	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_ffs.v

File #	Filename
61	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vld_path.v
62	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_rab.v
63	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_add_sub.v
64	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_rvalid_rdata_gen.v
65	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_core_tree_spu.v
66	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla5.v
67	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dagu_a2.v
68	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_shifts_top.v
69	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/daau_decoders/cevaxm4_dls_a1_decoder.v
70	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_case_cfg.v
71	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_fifo_line.v
72	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla12.v
73	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_l1pc_swop_single_fsm.v
74	$DW_LOCAL_ROOT/DW01/DW_lsd.v
75	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_mult_comb.v
76	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_flags.v
77	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcmp.v
78	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_ecc_correct.v
79	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf_pr_wr.v
80	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gfacla.v
81	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_cmp.v
82	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcmp_2_sel_logic.v
83	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_rd_mif.v
84	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vrf_vacc.v
85	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_queue_db.v
86	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_tap_state.v
87	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_find_parity.v
88	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_clock_gater.v
89	$DW_LOCAL_ROOT/DW01/DW01_addsub.v
90	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_l1dm_jbox_ecc.v
91	$DW_LOCAL_ROOT/DW01/DW_sra.v
92	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_task_fetch_fsm.v
93	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dfacla.v
94	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_pmss_axi_master.v

File #	Filename
95	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_scan_chains.v
96	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_fp2int.v
97	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcmp_compare_unit.v
98	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_source.v
99	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_rbsh.v
100	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_dman_cpm.v
101	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_l1dm_jbox_arbiter_ecc.v
102	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_extract_combine.v
103	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_store_en_gen.v
104	$DW_LOCAL_ROOT/DW01/DW01_inc.v
105	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_dstore_arf_sops.v
106	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_tcm_port.v
107	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_xtend.v
108	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla8.v
109	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_mult_2_stages.v
110	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rb_align.v
111	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_src1_select.v
112	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_epp.v
113	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_div_sqrt_sqrti.v
114	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcarrygen.v
115	$CEVAXM4_RTL_LOCAL_ROOT/top/wrapper/cevaxm4_prm.v
116	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr.v
117	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_gfifo.v
118	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_mpy.v
119	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_3to7_enc.v
120	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_ash.v
121	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr_mif.v
122	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/cevaxm4_daau_src2.v
123	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_logic.v
124	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_asu.v
125	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla4.v
126	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_pmss.v
127	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edp_praw_buf.v
128	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/cevaxm4_daau_to_vpu_bypass.v

File #	Filename
129	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_shift_32.v
130	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_4to16_enc.v
131	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_sra.v
132	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pseq_uop.v
133	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_adec_bank.v
134	$CEVAXM4_RTL_LOCAL_ROOT/top/general/geq_gt_cfg.v
135	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_mul_res_select.v
136	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dcarrygen.v
137	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf_sp_wr.v
138	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_intx1_mask.v
139	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_slb.v
140	$DW_LOCAL_ROOT/DW01/DW01_add.v
141	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_daau_dmss_intr.v
142	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_flcopy.v
143	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_dst_en_sel_gen.v
144	$DW_LOCAL_ROOT/DW04/DW_ecc.v
145	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_dmswb_pcu.v
146	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_parallel_pm.v
147	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_line_ext_byte_sel.v
148	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_cmp6.v
149	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rra_4to12mask.v
150	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_axim.v
151	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_parallel_agu_a1.v
152	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_iacu.v
153	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dagu_a1.v
154	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_lsd.v
155	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edp.v
156	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_adl1.v
157	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_l1dm_jbox_arbiter.v
158	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_dman_cpm_bman.v
159	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vpr_1el.v
160	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_frame_min_arb.v
161	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vasu.v
162	$DW_LOCAL_ROOT/DW01/DW01_cmp2.v

File #	Filename
163	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_s2d.v
164	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vrf_src.v
165	$DW_LOCAL_ROOT/DW01/DW_minmax.v
166	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_modv_reg.v
167	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_sys_wdog.v
168	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vrf.v
169	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/decoder/cevaxm4_vpu_decoder_e1.v
170	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_shifter.v
171	$DW_LOCAL_ROOT/DW01/DW01_bsh.v
172	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vst.v
173	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla16.v
174	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_dst_arf_w_dp.v
175	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_tcm_rd_proc.v
176	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pblock_repeat.v
177	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_lzd.v
178	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_vintra_unit.v
179	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_src0_select.v
180	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gmux2_1_cfg.v
181	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_adec.v
182	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_apbs.v
183	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/decoder/cevaxm4_vpu_decoder.v
184	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_l2d_z0.v
185	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_rd_stg.v
186	$DW_LOCAL_ROOT/DW03/DW03_pipe_reg.v
187	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table1_sqrt_even_exp.v
188	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_cmp2.v
189	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_dman.v
190	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_vpr_mask.v
191	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_parallel_agu_a2.v
192	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu.v
193	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_intx_mask.v
194	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_spu.v
195	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_palignment_256.v
196	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_top.v

File #	Filename
197	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_find_parity_core.v
198	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ios.v
199	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_core_tree_vcu_s5.v
200	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_fic_dec.v
201	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pdispatcher_256_single.v
202	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_generate_mss_clock.v
203	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_dbg.v
204	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_daau_dmss_parallel_intr.v
205	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_jtag.v
206	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman.v
207	$CEVAXM4_RTL_LOCAL_ROOT/core/top/cevaxm4_core_top.v
208	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf_wr.v
209	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_cyc_counter.v
210	$DW_LOCAL_ROOT/DW02/DW02_mult_2_stage.v
211	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf.v
212	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_rd.v
213	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_rash.v
214	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_mport_wdog.v
215	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_cosem_y0_y1.v
216	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rd_vu.v
217	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_edp_port.v
218	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_alu_2.v
219	$DW_LOCAL_ROOT/DW01/DW_shifter.v
220	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_afifo.v
221	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla32.v
222	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/daau_decoders/cevaxm4_dls_main_decoder.v
223	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla16_3in.v
224	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_lbsh.v
225	$DW_LOCAL_ROOT/DW01/DW01_sub.v
226	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_flags_calc.v
227	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_ram_access_unit.v
228	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_inc.v
229	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_mask_32_l1.v
230	$DW_LOCAL_ROOT/DW01/DW01_csa.v

File #	Filename
231	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_int_count.v
232	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_dma.v
233	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_q.v
234	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rab.v
235	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_dstore_arf.v
236	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_ram_access_arbiter_rr.v
237	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_qman_watermark.v
238	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_wrap_add.v
239	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vpr_src.v
240	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_ppc_calc_256.v
241	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_daau_sops_2ls_a2.v
242	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_axim_port.v
243	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_32bit_add.v
244	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_vacccast_unit.v
245	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vrf_vector.v
246	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_pipe.v
247	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_gcla20.v
248	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pfunc_unit_sel.v
249	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rd_buf_vu.v
250	$DW_LOCAL_ROOT/DW01/DW_lod.v
251	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_mcci.v
252	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_top.v
253	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_execute.v
254	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_rd_con.v
255	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_absub.v
256	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vffb.v
257	$DW_LOCAL_ROOT/DW01/DW01_cmp6.v
258	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_dman_cpm_qman_dpu.v
259	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_l1pc_swop_fsm.v
260	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_queue_access_arbiter_rr.v
261	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/cevaxm4_daau_prg_cond.v
262	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vacccast_vec_sat64.v
263	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/cevaxm4_daau_sops.v
264	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_cosem_cmd_range_sel.v

File #	Filename
265	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_sub.v
266	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_bsh.v
267	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_csa.v
268	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_mss_sys_timer.v
269	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gdec_cfg.v
270	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_vl2d_dst_en_gen.v
271	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gsync_ar.v
272	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_ext_wr.v
273	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gsample_cfg.v
274	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gfull_adder.v
275	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_adl_stg.v
276	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_pou.v
277	$DW_LOCAL_ROOT/DW02/DW02_mult.v
278	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_algn1.v
279	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_reset_synchronizer.v
280	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_adders_v3.v
281	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_kind.v
282	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vdecim.v
283	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_algn.v
284	$CEVAXM4_RTL_LOCAL_ROOT/top/wrapper/cevaxm4_wrapper.v
285	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_daau_dls_intr.v
286	$DW_LOCAL_ROOT/DW02/DW02_sum.v
287	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_dmswb_arf.v
288	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dscalar.v
289	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gnf_ff.v
290	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_neg_int_count.v
291	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_io_logic.v
292	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_l1pc_regs.v
293	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_axi_master_wr.v
294	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/general/cevaxm4_3input_msb_calc.v
295	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_tcm_rd.v
296	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr_bif.v
297	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vdecim_vec_sat16.v
298	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_dw_aligner.v

File #	Filename
299	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dcla16.v
300	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_intx2_mask.v
301	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcast.v
302	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ghalf_adder.v
303	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_addsub.v
304	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_adl_1stg.v
305	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb.v
306	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_absub_subtract.v
307	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_wrap_sub.v
308	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_edp_port.v
309	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vintrasum.v
310	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap.v
311	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_dacu.v
312	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_add_flags.v
313	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla3.v
314	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr_stg.v
315	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_count_ones_wrapper.v
316	$CEVAXM4_RTL_LOCAL_ROOT/top/cevaxm4_iol.v
317	$DW_LOCAL_ROOT/DW01/DW_rbsh.v
318	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_vcu_vpackunpack.v
319	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_adec_block.v
320	$CEVAXM4_RTL_LOCAL_ROOT/top/profiler/cevaxm4_profiler.v
321	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_daau_sops_2ls_a1.v
322	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcntbits.v
323	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_psequencer.v
324	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/daau_decoders/cevaxm4_dls_m_decoder.v
325	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_vsops.v
326	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_lod.v
327	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_src0_selectors_gen.v
328	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_merge_reset_4in.v
329	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_adl_1stg_vu.v
330	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pbranch.v
331	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf_addr_ptr_wr.v
332	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_vacccast_vec.v

File #	Filename
333	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_control.v
334	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_rab_swopf_end_gen.v
335	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vabscmp_v2.v
336	$DW_LOCAL_ROOT/DW01/DW_lzd.v
337	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_prm.v
338	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vshift_16bit.v
339	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_pmc.v
340	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_rra.v
341	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr_dif.v
342	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_3to8_enc.v
343	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_dls_en_sel_gen.v
344	$CEVAXM4_RTL_LOCAL_ROOT/core/fp/cevaxm4_fp_round.v
345	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_pmss_axi_master_wr.v
346	$DW_LOCAL_ROOT/DW01/DW01_dec.v
347	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_core_tree_vcu.v
348	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcntbits_2.v
349	$CEVAXM4_RTL_LOCAL_ROOT/top/wrapper/cevaxm4_control.v
350	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vrf_el.v
351	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_hist_rwp.v
352	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dls.v
353	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_two_stages_gated_clock.v
354	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vdecim_vec_sat32.v
355	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu_merge_reset.v
356	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table1_sqrt_inv_even_exp.v
357	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_shift_right_32.v
358	$CEVAXM4_RTL_LOCAL_ROOT/top/cevaxm4_pmem.v (Can be replaced by the user to the real memories top file)
359	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_rab_ptr_incrementor.v
360	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edp_ctl_fifo.v
361	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table0_sqrt_even_exp.v
362	$CEVAXM4_RTL_LOCAL_ROOT/top/wrapper/cevaxm4_glue.v
363	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_alu2_res_select.v
364	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_wrap_comp.v
365	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gmux_one_hot_cfg.v

File #	Filename
366	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vld.v
367	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_iarb.v
368	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla9.v
369	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_daau_vst_sample_16.v
370	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table0_sqrt_inv_even_exp.v
371	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_ram_access_arbiter_adec.v
372	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table0_sqrt_odd_exp.v
373	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_mpy.v
374	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pfetch_mux.v
375	$CEVAXM4_RTL_LOCAL_ROOT/top/psu/cevaxm4_psu.v
376	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_snoop.v
377	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pseq_glue.v
378	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_daau_ld_path.v
379	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_top.v
380	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vcmp_2.v
381	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/vrf/cevaxm4_vpu_vrf_vacc_el.v
382	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_16bit_add.v
383	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_l1pc_swop_top.v
384	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pslotnum_dec_256.v
385	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table_div.v
386	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr_aif.v
387	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_rd_dif.v
388	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_cpm.v
389	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_cosem_exp.v
390	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_add.v
391	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_dmswb.v
392	$CEVAXM4_RTL_LOCAL_ROOT/top/wrapper/cevaxm4_alignment.v
393	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vdecim_vec.v
394	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_dpu.v
395	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_axi_master_rd.v
396	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_axim_port.v
397	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_queue_arb.v
398	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_l2d_z3.v
399	$CEVAXM4_RTL_LOCAL_ROOT/top/ocem/cevaxm4_ocem_break.v

File #	Filename
400	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_dcla16_3in.v
401	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_table1_sqrt_inv_odd_exp.v
402	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_tag_ecc.v
403	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_tcm_port.v
404	$CEVAXM4_RTL_LOCAL_ROOT/core/dispatch/cevaxm4_pwordnum_dec_256.v
405	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pexcept_handle.v
406	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_wr_con.v
407	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_decode.v
408	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_bm.v
409	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma.v
410	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_l2d.v
411	$CEVAXM4_RTL_LOCAL_ROOT/core/sequencer/cevaxm4_pseq_decoder.v
412	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_lru.v
413	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_leading_one_detector.v
414	$CEVAXM4_RTL_LOCAL_ROOT/top/cevaxm4_sys.v
415	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/general/cevaxm4_4input_adder.v
416	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_l1pc.v
417	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gcla4_3in.v
418	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_qman_simple_pr.v
419	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cevaxm4_vpu_byte_aligner.v
420	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_hmd.v
421	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/darf/cevaxm4_darf_mod_wr.v
422	$DW_LOCAL_ROOT/DW01/DW_sla.v
423	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edap_ptcm.v
424	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_axi_limit.v
425	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_addsub_flags.v
426	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/ls/cevaxm4_byte_aligner.v
427	$DW_LOCAL_ROOT/DW01/DW01_ash.v
428	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_algn.v
429	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edp_praw.v
430	$CEVAXM4_RTL_LOCAL_ROOT/top/general/gmul16x16_cfg.v
431	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/scalar/cevaxm4_dsc_cmps.v
432	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_vshift.v
433	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_ext_rd.v

File #	Filename
434	$CEVAXM4_RTL_LOCAL_ROOT/top/general/greg_cfg.v
435	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_vorf_mux.v
436	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_cosem_norm.v
437	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_sub_flags.v
438	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ginc_cfg.v
439	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_l1dm_jbox.v
440	$CEVAXM4_RTL_LOCAL_ROOT/core/daau/mswb/cevaxm4_daau_vst_sample.v
441	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_q.v
442	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_hist_ib.v
443	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_hist.v
444	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_edp_set_rst.v
445	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/cosem/cevaxm4_vpu_div_sqrt_sqrti_top.v
446	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_mp_earb.v
447	$CEVAXM4_RTL_LOCAL_ROOT/top/general/ceva_dw_dec.v
448	$CEVAXM4_RTL_LOCAL_ROOT/mss/pmss/cevaxm4_pmss_axi_master_rd.v
449	$CEVAXM4_RTL_LOCAL_ROOT/top/general/cevaxm4_5to32_encoder.v
450	$CEVAXM4_RTL_LOCAL_ROOT/core/vpu/alu/cevaxm4_vpu_l2d_z1.v
451	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_ddma_tcm_wr.v
452	$CEVAXM4_RTL_LOCAL_ROOT/dman/cevaxm4_dman_cpm_qman.v
453	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_dmss.v
454	$CEVAXM4_RTL_LOCAL_ROOT/mss/dmss/cevaxm4_wb_adl.v
455	$ETM_LOCAL_ROOT/EtmR4Fifo.v
456	$ETM_LOCAL_ROOT/EtmR4Cell8StageC.v
457	$ETM_LOCAL_ROOT/EtmR4AddrAsy.v
458	$ETM_LOCAL_ROOT/EtmR4AddrCmp.v
459	$ETM_LOCAL_ROOT/EtmR4Comparators.v
460	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_sim_top.v
461	$ETM_LOCAL_ROOT/EtmR4Cell16StageC.v
462	$ETM_LOCAL_ROOT/EtmR4SOC.v
463	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/ehwvu/ehwvu_host.v
464	$ETM_LOCAL_ROOT/EtmR4SOCClkGen.v
465	$ETM_LOCAL_ROOT/EtmR4Cell8StageA.v
466	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/tpiu_monitor.v
467	$ETM_LOCAL_ROOT/EtmR4FifoPeek.v

File #	Filename
468	$ETM_LOCAL_ROOT/EtmR4APBInterface.v
469	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_vrf_mx.v
470	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_host_memory.sv
471	$ETM_LOCAL_ROOT/EtmR4Cell16StageA.v
472	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/mem/cevaxm4DSPmem.v
473	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/sim_rptu.v
474	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_ar_interface.sv
475	$ETM_LOCAL_ROOT/EtmR4ResControl.v
476	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteDefs.v
477	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/ocem/ocem_host.v
478	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_vrf_logger.v
479	$ETM_LOCAL_ROOT/EtmR4Cell16StageD.v
480	$ETM_LOCAL_ROOT/EtmR4PowerReset.v
481	$ETM_LOCAL_ROOT/EtmR4DataSuppress.v
482	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/mem/cevaxm4DSPmemX64_assoc.sv
483	$ETM_LOCAL_ROOT/EtmR4SyncCount.v
484	$ETM_LOCAL_ROOT/EtmR4ContextID.v
485	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/rand_toggle.v
486	$ETM_LOCAL_ROOT/EtmR4ClkGate.v
487	$ETM_LOCAL_ROOT/EtmR4EventGen.v
488	$ETM_LOCAL_ROOT/EtmR4TraceOut.v
489	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_if_master.sv
490	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_host_ctl.sv
491	$ETM_LOCAL_ROOT/EtmR4FifoIn.v
492	$ETM_LOCAL_ROOT/EtmR4AddrComp.v
493	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/mem/cevaxm4DSPmemX16_be_cfg.v
494	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteTraceClk.v
495	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/mem/cevaxm4DSPmemX32_be_cfg.v
496	$ETM_LOCAL_ROOT/EtmR4SingleSync.v
497	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteSerial.v
498	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_thpt.sv
499	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteRegP.v
500	$ETM_LOCAL_ROOT/EtmR4AddrBls.v
501	$ETM_LOCAL_ROOT/EtmR4SyncReq.v

File #	Filename
502	$ETM_LOCAL_ROOT/EtmR4SOCResetGen.v
503	$ETM_LOCAL_ROOT/EtmR4Defs.v
504	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_axi_throughput_an.sv
505	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_sim_cntrl.v
506	$ETM_LOCAL_ROOT/EtmR4Cell8StageB.v
507	$ETM_LOCAL_ROOT/ETMR4.v
508	$ETM_LOCAL_ROOT/EtmR4Fifo16.v
509	$CORESIGHT_LOCAL_ROOT/CSTPIULITE.v
510	$ETM_LOCAL_ROOT/EtmR4Counter.v
511	$ETM_LOCAL_ROOT/EtmR4DerivedRes.v
512	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_host.sv
513	$ETM_LOCAL_ROOT/EtmR4ProcIF.v
514	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_ready_toggle.sv
515	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/apb3/apb3_host.v
516	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteAtbIf.v
517	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/mem/host_mem_64x1024.v
518	$ETM_LOCAL_ROOT/EtmR4TrcEn.v
519	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_logger_arf.v
520	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteFtCtl.v
521	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_logger.v
522	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_memory_interface_an.v
523	$ETM_LOCAL_ROOT/EtmR4CORE.v
524	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/sim_erru.v
525	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteRegA.v
526	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_logger_acf.v
527	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_if_an.sv
528	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_freerun_an.sv
529	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/apb3/apb3_master.sv
530	$ETM_LOCAL_ROOT/EtmR4Fifo8.v
531	$ETM_LOCAL_ROOT/EtmR4Seq.v
532	$ETM_LOCAL_ROOT/EtmR4SOCSync.v
533	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_apb3_an.sv
534	$ETM_LOCAL_ROOT/EtmR4FifoRotate.v
535	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/ehwu/ehwu_host.v

File #	Filename
536	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_power_isolation_an.sv
537	$ETM_LOCAL_ROOT/EtmR4Cell16StageB.v
538	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_apb_throughput_an.sv
539	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/mem/cevaxm4DSPmemX64_be_cfg.v
540	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_logger_vflag.v
541	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/clk_gen/clk_gen.v
542	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/cevaxm4_calc_axi_slave_parity.v
543	$ETM_LOCAL_ROOT/EtmR4TrigGen.v
544	$ETM_LOCAL_ROOT/EtmR4ViewData.v
545	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_axi_if_an.sv
546	$ETM_LOCAL_ROOT/EtmR4SingleSyncSet.v
547	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/ocem/jtag_tasks.v
548	$ETM_LOCAL_ROOT/EtmR4SyncAck.v
549	$ETM_LOCAL_ROOT/EtmR4COREResetGen.v
550	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/analyzers/cevaxm4_verifeq_an.v
551	$ETM_LOCAL_ROOT/EtmR4Trace.v
552	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_siggen_rf.v
553	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/apb3/apb3_adec.sv
554	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/apb3/cevaxm4_calc_apb3_slave_parity.v
555	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/apb3/gapb_if_cfg.v
556	$ETM_LOCAL_ROOT/EtmR4DataCmp.v
557	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_vrf.v
558	$ETM_LOCAL_ROOT/EtmR4ControlReg.v
559	$ETM_LOCAL_ROOT/EtmR4CORESync.v
560	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/cevaxm4_calc_axi_master_parity.v
561	$ETM_LOCAL_ROOT/EtmR4ResetSync.v
562	$ETM_LOCAL_ROOT/EtmR4EventGenMux.v
563	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_logger_srf.v
564	$ETM_LOCAL_ROOT/EtmR4Control.v
565	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/hosts/axi/axi_aw_interface.sv
566	$CORESIGHT_LOCAL_ROOT/CSTpiuLiteApbIf.v

File #	Filename
567	$ETM_LOCAL_ROOT/EtmR4FifoOut.v
568	$CEVAXM4_SIM_LOCAL_ROOT/asm/verilog/top/cevaxm4_sim_counter.v
569	$ETM_LOCAL_ROOT/EtmR4CORE ClkGen.v
570	$ETM_LOCAL_ROOT/EtmR4Trigger.v

12. Glossary

Table 12-1 defines the acronyms used in this document.

Table 12-1: Acronyms

Term	Definition
ADEC	Address Decoder
AMBA	AMBA protocol
APB	APB protocol
APV	Access Protection Violation
AXIM	AXI Master
DDMA	Data DMA
DMEM	Data Memory
DMSS	Data Memory Subsystem
DPS	Dynamic Power Save
DSECT	Data Section
DUT	Design Under Test
DVM	Data Value Match
ECC	Error Checking and Correction
EDAP	External Device Access Port
EDP	External Data Port
EPP	External Program Port
FIFO	First In, First Out
GVI	Global Violation
HIST	Histogram
IACU	Instruction Access Control Unit
IOP	I/O Port
LSB	Least significant bit
MCCI	Multi-core Communication Interface
MSS	Memory Subsystem
OCEM	On-Chip Emulation Module
PCACHE	Program Cache
PDMA	Program DMA
PMEM	Program Memory
PMSS	Program Memory Subsystem
PSU	Power Scaling Unit

Term	Definition
PTCM	Program TCM
QMAN	Queue Manager
RTT	Real-Time Trace
SDT	Software Development Tools
SIP	Silicon Intellectual Property
SPU	Scalar Processing Unit
STD	Standard cell libraries
SWOP	Software Operation
VPU	Vector Processing Unit
WRC	Write Response Counter