Chapter 15.  Cosimulation

Author: Youngmin Yi, Youngpyo Joo, and Soonhoi Ha

15.1 Introduction

Hardware/Software cosimulation enables designers to evaluate or verify the system under design before it is manufactured. In PeaCE, cosimulation can be used in different design stages with different purposes; it can be used to evaluate the performance of the system, to verify the functional and timing correctness of the system, or to generate memory trace files that will be used in design space exploration for communication architecture selection.

PeaCE provides its own cosimulation tool based on virtual synchronization scheme as well as the environment to run Seamless CVE, a well-known commercial verification tool, directly in PeaCE. Seamless CVE is very accurate but also very slow. It is for verification of the entire system including all implementations. For other purposes than verification, such as evaluation of the system performance or trace generation of each processing elements, cosimulation with virtual synchronization scheme is more adequate.

The virtual synchronization scheme enhances the speed of cosimulation drastically while employing the same kind of cycle-accurate simulators as in Seamless CVE. The enhancement mainly comes from the fact that time synchronization between simulators is virtually removed through trace-driven simulation with communication architecture model and SW architecture model like OS. Surely the detailed discussion of virtual synchronization is beyond the scope of this user manual. Just enjoy the benefits of fast cosimulation.

NOTE: For HW/SW cosimulation, we need component simulators that would be integrated to the PeaCE design flow through foreign interface facility that the simulator provides. In this release, we support ARMulator for processor cores and ModelSim for hardware IPs. To use other types of processing element, contact us directly to know how to do it.

The detailed explanation for environment setting and how to perform each kind of cosimulation will be discussed in the following sections.

15.2 Cosimulation Environment Structure

Figure 15-1 describes the overall cosimulation environment structure. It comprises the Backplane executable, component simulators along with proper simulation interface for virtual synchronization.
The following is the explanation of the parts of the structure.

1) **Generated codes**: C codes and VHDL codes are automatically generated from the PeaCE.

2) **Interface codes**: OS API definition for multi-task execution and HW interface codes (IF) such as memory interface and synchronization logic are compiled and built with the generated application codes.

3) **BP executable**: It is the simulator engine that schedules the invocation of each component simulator and delivers data to and from the simulators. In case of virtual synchronization scheme, it includes additional simulation models such as communication channel model (i.e., memory, memory controller, arbitration logic) and OS model (i.e., scheduler and task synchronization APIs provided by target OS). In case of Seamless CVE coverification, it is simply Seamless CVE engine.

4) **Component simulators**: Cycle-accurate simulators

5) **Simulation interface**: Simulation interface is required to apply virtual synchronization technique to off-the-shelf simulators. It generates traces with timing information and data, and delivers them to the BP executable or delivers data from the BP executable to the simulator. It saves the traces into files or handles I/O requests, if necessary. In seamless CVE environment, it is not required.

### 15.3 Interface Code Generation

C code generation of a block mapped to SW component and VHDL code generation of a block mapped to HW component has been explained in chapter 5 and 6 respectively. To perform cosimulation, complete codes including interface codes must be built in advance. In this section, interface code generation is explained.

#### 15.3.1 Interface Code Generation for Seamless CVE

Interface is very much dependent on the target architecture. Synchronization logic and memory interface logic are obtained from the design library and application dependent interface codes are automatically generated.

**NOTE:** Assumptions and Limitations

1) **SW Parts**:
   - It doesn’t support blocking read/write and preemptive scheduling between tasks, because there is no OS environment (especially thread environment) yet. It supports only priority-based non-preemptive scheduling.
- System calls like open(), write() and read() need to be replaced by fopen(), fwrite(), fread(), etc. Take a look at the generated codes of DIVX_cve schematic as an example.
- System calls like printf(), and putc() are supported by Text Output Console.
- I/O handler that handles general I/O requests in Seamless CVE environment will be included in the next release.

2) HW Parts:
- It supports only single bus, AMBA AHB. Its clock frequency is 31.25 MHz. (modifiable)
- It supports only SRAM. Its clock frequency is the same as the bus clock frequency.
- It supports only one ARM926ej-s in system as for now. Its clock frequency is 250 MHz.
- It supports multiple HW IPs, but it has been tested when there is only one HW IP. Its clock frequency is the same as the bus frequency.
- Because of memory controller’s restriction, ROM can be mapped between 0x0 ~ 0x20000000, and RAM can be mapped between 0x40000000 ~ 0x70000000.
- It does not support FRDF due to synchronization controller’s limitation. It supports only single/multi rate synchronization for now. This restriction will be removed in the next release.

1) SW Interface Code

$HOME/PEACE_SYSTEMS/Project_Name/seamless/functionAPI_arm.c
: It contains sequential execution scheduler codes, read/write codes between tasks. C entry function, main() is defined in this file.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/libads/init.s
: Initialization codes. It calls the C entry function.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/libads/irq_handler.c
: Interrupt handler and system calls that handle interrupts for SW applications.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/libads/page_table.s
: Page table for MMU (automatically generated).

$HOME/PEACE_SYSTEMS/Project_Name/seamless/libads/retarget.c, strncasecmp.c
: System Calls like fopen(), fwrite(), strncasecmp() for SW applications.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/libads/scat.scf
: Memory map file for linker (automatically generated).

$HOME/PEACE_SYSTEMS/Project_Name/seamless/libads/system.*
: Channel (between SW and HW) information and related APIs (automatically generated).
2) HW Interface Code

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libahb/a926ahb.vhd
: Module that wraps ARM926ej-s CPU core module.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libahb/ahbarb.vhd
: AHB Bus arbiter.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libahb/ahbsync*.vhd
: Synchronization controller for interrupt based synchronization between CPU and HW IP.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libahb/ahbwrapper*.vhd
: AHB bus wrapper for HW IP. It provides interface for HW IP to access shared and local memories.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libmem/sram32.vhd
: 32bit SRAM module.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libetc/clkgen.vhd
: This module generates several clocks. If you want to change clock frequency, you should modify the value in this file. Clken is a clock enable signal for synchronization between Bus and CPU.

$HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/libetc/top.vhd
: Top entity of the entire system.

15.3.2 Interface Code Generation for Virtual Synchronization Scheme

For the cosimulation based on the virtual synchronization scheme, eCOS which is an open-source RTOS made in RedHat is used to provide a multi-task execution environment and a startup code for underlying target processor architecture. Data synchronization between SW and HW tasks are not simulated in the device driver level but modeled in higher abstraction. Therefore, SW-side interface codes such as device drivers and interrupt handler for synchronization logic are not generated.

HW-side interface codes are generated with additional signals for the simulation interface (i.e., FLI: see section 15.4.2). Some signals such as interrupt in synchronization logic are excluded for the same reason; in virtual synchronization scheme, they are not simulated but modeled.
15.4 Cosimulation Environment Setting For Virtual Synchronization

15.4.1 eCOS

For multi-task execution in a SW component simulator, a certain set of OS APIs (e.g., task preemption) as well as startup code is required. PeaCE provides eCOS (http://sources.redhat.com/ecos) port to ARMulator for this purpose. Note that only the minimal necessary OS APIs excluding OS scheduler is used in virtual synchronization scheme since scheduling and task synchronization are performed in the OS model in BP executable. The eCOS port to arm922T processor in ARMulator (see section 15.4.2) is provided in $PEACE/ecos/ and this path must be set as $ECOS_HOME. Designers might build their own eCOS port for other target processor core. In that case, make sure to locate the newly ported eCOS image to the following directory.

- $ECOS_HOME/ecos-target/install/lib/: eCOS image and linking script file
- $ECOS_HOME/ecos-target/install/include/: header files

NOTE: To change the target processor, there must be a port of eCOS to that processor. Currently, PeaCE provides ARM720T and ARM922T ports (or ARM926EJ-S). Designers must copy all the files in $ECOS_HOME/ecos-target/simulator_port to the $ARMHOME/linux/Source/armulext/ and run ‘compile’ script.

15.4.2 Simulation interface

It is required to have proper simulation interface in each component simulator to support virtual synchronization technique. Currently, only ARM architecture is supported as a target processor and ARMulator is used as an cycle-accurate ISS. As an HW component simulator PeaCE supports ModelSim. In this section, inserting simulator interface codes for ARMulator and ModelSim is described.

1) ARMulator interface

The interface codes are provided in $PEACE/ads1.2/ assuming ADS 1.2 (Linux version) has been installed. Designers must copy these simulation interface codes to the proper path. Brief description of each source code along with its path is given as below:

<Modified source code>

$ARMHOME/linux/Source/armulext/flatmem.c
: Flat memory model of ARMulator: modified to call APIs defined in below files
NOTE: This file is part of ADS 1.2 distribution. Thus, we only provide patch for this file. Type ‘cat flatmem.patch | patch –p0’ to generate modified version of flatmem.c

<New source codes>

$ARMHOME/linux/Source/armulext/socksrc.{c, h}
: It contains APIs that establishes socket connection between simulation engine (BP executable) and delivers data to and from it. Also memory trace information is delivered to the simulation engine.
$ARMHOME/linux/Source/armulext/tracegen.{c, h}
: It contains APIs that generate memory access traces
$ARMHOME/linux/Source/armulext/iomodel.{c, h}
It contains the APIs that handle I/O requests such as file I/O or audio and display device access. It is very useful when designers cannot use semihosting feature of ARMulator to use other compiler than armcc (e.g., arm-elf-gcc).

<Makefile>
$ARMHOME/linux/Source/armulext/flatmem.b/linux86/Makefile
$ARMHOME/linux/Source/armulext/compile
: Run ‘compile’ script to build new Flatmem.so that contains simulation interface between ARMulator and PeaCE

2) ModelSim interface

HW simulator interface codes are implemented as FLI (Foreign Language Interface) and PeaCE provides ModelSim simulation interface implemented using ModelSim FLI. Since they are implemented as FLI, the interface codes are located where HW IP codes are generated and are built together with them. Brief description of each FLI code along with its path is given as below:

$HOME/PEACE_SYSTEMS/Project_Name/\{cosim|dse\}/VHDL/libahb/ahbwrapper.c
: It establishes socket connection between the simulation engine (BP executable) and ModelSim for data delivery. It also generates memory access traces and delivers them to the BP executable.

$HOME/PEACE_SYSTEMS/Project_Name/\{cosim|dse\}/VHDL/libahb/ahbsync.c
: Subsidiary interface codes modeling synchronization logic.

15.5 Cosimulation for Trace Generation

Even after partitioning has been determined, there remain a lot of design choices for communication architecture selection. In chapter 14, design space exploration scheme for communication architecture selection is explained in detail. The memory traces of processing elements needed in the design space exploration step are obtained through cosimulation.

15.5.1 How to perform cosimulation for trace generation

Do the following to perform cosimulation for trace generation:

- Select the dse tab and check whether the target is correctly configured as TraceGen. (Figure 15-2)
- Click ‘Run’. This cosimulation does not have ‘Run Count’ because, in trace generation, cosimulation is finished automatically when all of the tasks have been executed at least one iteration.
  - SW code is generated as C codes in $HOME/PEACE_SYSTEMS/Project_Name/dse/
  - HW codes are generated as VHDL codes in $HOME/PEACE_SYSTEMS/Project_Name/dse/VHDL.
- To start cosimulation, type the name of BP executable, Project_Name_0 in $HOME/PEACE_SYSTEMS/Project_Name/dse/. In this example, type DIVX_0 and cosimulation will start. (Figure 15-3)
Figure 15-2 Running cosimulation for trace generation

Figure 15-3 Cosimulation for trace generation
At first, BP executable or simulation engine will invoke the I/O server that handles I/O requests from the simulators. Then, it will invoke each simulator in turn and waits for the user to press any key to start execution. After completion of cosimulation, it also waits for the user to press any key to close all simulators.

15.5.2 Generated trace files

As a result of cosimulation, traces for each block in the system are generated and saved in separated files. The path and the name of the file is made as follows:

$HOME/PEACE_SYSTEMS/Project_Name/trc/full_name_of_a_block.txt

The format of the trace file is explained in chapter 11.

15.6 Cosimulation for Evaluation of System Performance

To enhance the speed of cosimulation while employing cycle-accurate simulators, a technique called *virtual synchronization* has been adopted in PeaCE. In this scheme, communication architecture and SW architecture like OS is not actually simulated but modeled. This cosimulation is useful for verifying component IPs other than communication architecture and for evaluating the whole system in a short time.

15.6.1 How to perform cosimulation for system evaluation

![Figure 15-4 Running Cosimulation](image)
Select the cosim tab and check whether the target is correctly configured as Cosimulation. (Figure 15-4)

Enter iteration count of a task with the minimal period in the system in the text box next to ‘RUN’ button.

Click Run.

- SW code is generated as C codes in $HOME/PEACE_SYSTEMS/Project_Name/cosim/
- HW codes are generated as VHDL codes in $HOME/PEACE_SYSTEMS/Project_Name/cosim/VHDL.

To start cosimulation, type the name of BP executable, Project_Name_0 in $HOME/PEACE_SYSTEMS/Project_Name/cosim/. In this example, type DIVX_0 and cosimulation will start just like in Figure 15-3

At first, BP executable or the simulation engine will invoke the I/O server that handles I/O requests from the simulators. Then, it will invoke each simulator in turn and waits for the user to press any key to start execution. After completion of cosimulation, it also waits for the user to press any key to close all simulators.

15.7 Coverification

Designers can verify the whole entire system, or virtual prototype, through highly accurate but very slow cosimulation such as Seamless CVE directly in PeaCE.

15.7.1 How to perform coverification with Seamless CVE

DIVX_cve is an example for coverification with Seamless CVE. This example is a modified version of DIVX example in which mp3 decoder task is removed.

NOTE: Preconditions

1) default.map:
   This file is needed in $HOME/PEACE_SYSTEMS/Project_Name/ directory. This file defines the system memory map. Its recommended form is in the Table 15-1 below. The third or the fourth column means the device name, and the first column means its address, the second column means its size.
2) Opening a file:
Designers must load an input file to a specific memory region before it is read. In DIVX_cve example, input file for Avi file parser is assumed to be located in 0x06000.0000. This can be done using armsd command, getfile. For example, type “getfile friend.avi 0x6000.0000” in armsd.

3) Compiler option of H.263 decoder task is set with the value, ‘-D_READ_OPTIMIZE’.
- Select the cosim tab and check whether the target is correctly configured as Seamless CVE. (Figure 15-5)
- Click Run.
  - SW codes are generated as C codes in $HOME/PEACE_SYSTEMS/Project_Name/seamless/.
  - HW codes are generated as VHDL codes in $HOME/PEACE_SYSTEMS/Project_Name/seamless/VHDL/.
- If ‘Compile?’ and ‘Run?’ parameters are set to ‘YES’, Coverification will start automatically. (Figure 15-6)
- In the armsd command prompt, type “getfile friend.avi 0x6000.0000” where friend.avi is the input file name.
- Type “go” in armsd command prompt to continue the execution.

Figure 15-6 Coverification with Seamless CVE

15.7.2 Manual compilation and running for Seamless CVE

- Set ‘Compile?’ and ‘Run?’ parameters to ‘NO’.
- Click Run. Then, SW and HW codes and makefiles are generated.
- SW codes can be compiled by typing ‘make’ in $HOME/PEACE_SYSTEMS/Project_Name/seamless/.
- HW codes have more than two compile depths in PeaCE, so it’s not recommended to compile manually.
- Run the coverification by typing the generated script ‘./platform.cve’ in $HOME/PEACE_SYSTEMS/Project_Name/seamless/