5 Developing Acceleratable Universal Verification Components (UVCs)

This chapter discusses the following topics:

- Introduction to UVM Acceleration
- UVC Architecture
- UVM Acceleration Package Interfaces
- SCE-MI Hardware Interface
- Building Acceleratable UVCs in SystemVerilog
- Building Acceleratable UVCs in e
- Collector and Monitor

5.1 Introduction to UVM Acceleration

The acceleratable Universal Verification Methodology (UVM) packages allow portions of a standard UVM environment to be accelerated using a hardware accelerator. The extended UVM acceleration packages include support for SystemVerilog and the e high-level verification languages (HVLs). Though this chapter only discusses UVM, acceleration for both the Open Verification Methodology (OVM) and UVM are supported. So, any references to UVM equally apply to OVM.

The purpose of extending UVM to include hardware acceleration is to enable the verification environment to execute faster. Hardware acceleration can dramatically increase run time performance and, therefore, allow more testing to be done in a shorter amount of time, and making the verification engineer more productive.

Although the main purpose of using the UVM acceleration library is to allow a hardware accelerator to be used, it is not restricted to hardware acceleration alone. UVM acceleration is truly an extension of the standard simulation-only UVM, and is fully backwards compatible with it. This means that Universal
Verification Components (UVCs) architected to be acceleratable can be used in either a simulation-only environment or a hardware-accelerated environment. However, UVCs that were not architected to leverage hardware acceleration will require some modifications to enable them to be used in a hardware-accelerated environment.

5.2 UVC Architecture

This section briefly describes the standard UVC architecture and explains how this differs from the acceleratable UVC architecture.

5.2.1 Standard UVC Architecture

UVCs based on the standard UVM typically contain the following three main components, which are themselves contained within an agent component, as shown in Figure 5-1 below. Each agent contains:

- A sequencer (also known as sequence driver)
- A driver (also known as BFM)
- A monitor

![Figure 5-1 Standard UVC Architecture]

5.2.2 Active Agent

The architecture shown in Figure 5-1 is typical of an agent that actively drives stimulus into the device under test (DUT).

Stimulus is provided by the sequencer in an abstract form known as a data item. Data items are transactions that only contain stimulus information; the interface and protocol details related to the DUT are abstracted out. Data items in SystemVerilog are classes extended from the uvm_sequence_item class. In e, these are extended from the any_sequence_item item.
The driver connects to the DUT interface and applies the data items provided by the sequencer to this interface in accordance with the interface protocol.

A monitor is used to observe the activity on the DUT interface as well as activity on internal nodes of the DUT to collect coverage metrics about what parts of the DUT have been exercised. A standard UVM monitor usually includes a hard-coded connection to the interface as well as the coverage-collection functionality. Having a hard-coded connection to the interface is not ideal if the UVC is to be used to verify a DUT at multiple levels of abstraction because a new monitor will need to be created for each abstraction level.

5.2.3 Passive Agent

A UVC can be configured solely to collect DUT activity rather than to stimulate activity. The collected information can then be used by checkers, coverage tools, and the testbench itself for cases where up-to-date status is required. This is a typical scenario when the DUT is integrated into a system. Under these circumstances, the sequencer and driver components are disabled leaving only the monitor. The agent in this scenario is referred to as a passive agent.

Coverage information allows the verification team to ensure that the DUT is thoroughly tested by measuring the features that have been exercised, and the ones that have not. Coverage information can also be used by a scoreboard component that can be used to track the features that have been tested.

A UVC can be used to verify models at various levels of abstraction, each with different types of interfaces. Decoupling the stimulus generation from driving the physical DUT interface allows stimulus to be reused for verifying different abstractions of a given model by simply selecting the appropriate driver. This is most applicable to simulation environments that support the broadest range of HVL constructs.

5.2.4 Acceleratable UVCs

Acceleratable UVCs benefit from a slightly different architecture than simulation-only UVCs in order to maximize the performance gain provided by the hardware accelerator. Therefore, in order to describe acceleratable UVCs, a brief introduction to hardware acceleration must be given. More information about hardware acceleration can be found in the UXE User's Guide, which is included with the Cadence Palladium XP family of hardware accelerators.

5.2.4.1 Hardware Acceleration

Hardware acceleration is performed by combining a software simulator that executes on a workstation with a dedicated hardware-acceleration machine. The complete verification environment is partitioned to have some models executed by the simulator and others by the hardware accelerator. Models described using high-level verification language (HVL) constructs are executed by the simulator, and are said to reside in the HVL partition. Models described using hardware description language (HDL) constructs are executed by the hardware accelerator, and are said to reside in the HDL partition.

Hardware accelerators can only accelerate models that have been described using the acceleratable subset of an HDL. This subset is usually assumed to be the same as the register-transfer-level (RTL) subset defined for hardware synthesis, but this is often not the case. Hardware acceleration platforms usually accept a number of
behavioral constructs as well as synthesizable constructs. So, the level of support is greater than that contained in the synthesizable subset of constructs; however, it is still a subset of the complete HDL. Any component that cannot be modeled using this subset must remain in the HVL partition. One requirement that must be fulfilled is that all models in the HDL partition must have signal-level interfaces. However, signal-level connections joining components in the HVL partition to components in the HDL partition are not efficient for achieving high runtime performance. Instead, a transaction-based connection must be used. The industry recognized this concept as being a key requirement in order to achieve high runtime performance when connecting a software simulator to a hardware accelerator. This led to the creation and standardization of the Accellera Standard Co-Emulation API: Modeling Interface, more commonly referred to as SCE-MI.

The use of a transaction-based interface between the software simulator and the hardware accelerator not only allows the communication between the two engines to be made more efficient, it also allows the execution of simulation models to be made more efficient. This is because simulation performance is reduced when the models being executed become more detailed and require timing. Therefore, simulating untimed models at the transaction level improves simulation performance.

The partitioning of transaction-level components and cycle-accurate signal-level components between the software simulator and hardware accelerator respectively, leads to a change in the overall verification environment architecture. Two separate top levels of hierarchy are created for each of the two partitions, with all communication between the two partitions being performed at the transaction level. Components like scoreboards, sequencers and monitors are placed in the HVL partition, while components like clock generators, reset generators, and the signal-level DUT are placed in the HDL partition, as shown in Figure 5-2.

**Figure 5-2 Components of the HVL and HDL Partitions**

![Components of the HVL and HDL Partitions](image)

**Acceleratable UVC Architecture**

To enable high runtime performance to be achieved, all models that reside in the HVL partition should execute at the transaction level, and all models that require cycle-accurate timing should reside in the HDL partition. Transactors are used to allow components within each partition to communicate with each other efficiently. However, the architecture shown in Figure 5-1 on page 196 does not allow a clean division of
Acceleratable UVCs

functionality to be made because the monitor operates at the same abstraction level as the DUT, which for acceleration would be at the signal level. To address this, the monitor should be split into two components, a monitor and a collector, as shown in Figure 5-3.

Figure 5-3 Acceleratable UVC Architecture

The purpose of the collector is to allow the physical interface required by the DUT to be separated from the functionality provided by the monitor. This means that the monitor and sequencer, and all hierarchical levels above, can operate at the transaction level, irrespective of the type of interface required by the DUT. Modeling these components at this level of abstraction is good for reuse as well as for increasing execution performance. The collector and driver components implement the physical interface required to enable the UVC to connect to the DUT, which can be easily altered depending on the type of interface required without affecting the rest of the UVC.

As mentioned previously, for the hardware acceleration mode, models that reside in the HVL partition operate at the transaction level, while those that reside in the HDL partition execute at the signal level. One consequence of configuring the UVC to use hardware acceleration is that the acceleratable collector and driver components must incorporate transactors to convert signal-level activity to transactions, and vice versa.

Acceleratable Transactors

Transactors are an abstraction bridge between the components that operate at the transaction level and the components that operate at the signal level. For hardware acceleration, transactors extend this capability by bridging between transaction-based components being executed by a software simulator and signal-based components being executed by a hardware accelerator as shown in Figure 5-4 on page 200.
To bridge between the HVL partition and the HDL partition, the transactors have three main components:

- **Proxy model**
  
  The proxy model is instantiated in the HVL partition and accesses the communication channel by way of an Application Programming Interface (API).

- **Bus Functional Model (BFM)**
  
  The BFM is instantiated in the HDL partition and also accesses the communication channel by way of an API.

- **Communication channel that connects between the Proxy and BFM**
  
  Each channel is uni-directional and this is reflected in the choice of interface used within each of the partitions.

A simple transactor with one input and one output channel is shown in Figure 5-5.

**Figure 5-5  Transactor Example with Input and Output Channels**

To enable transactors to operate on different vendor’s hardware acceleration platforms, a standard vendor-independent API was defined and standardized by Accellera for connecting any software simulator to any hardware accelerator. The standard, known as the Standard Co-Emulation API: Modelling Interface (SCE-MI), defines a multichannel communication interface.
SCE-MI initially defined a macro-based interface. But later, it added a simpler Direct Programming Interface (DPI) and a more complex but feature-rich pipes-based interface. All of the above interfaces are described in the *Standard Co-Emulation API: Modelling Interface (SCE-MI) Reference Manual*, Version 2.0, or later, and is available from Accellera. The UVM Acceleration interface uses SCE-MI pipes communications channels.

SCE-MI pipes are unidirectional channels that allow transactions to be streamed from components in the HVL partition to components in the HDL partition and vice versa. A C language API is available to components residing in the HVL partition and a SystemVerilog API is available to components residing in the HDL partition.

To simplify the use of SCE-MI in the development of acceleratable UVCs, a UVM Acceleration library, `uvm_accel`, is provided in the Cadence UXE software release to hide the semantics of the SCE-MI C API presented to models that reside in the HVL partition. The `uvm_accel` package provides a UVM-based API that is native to the verification language being used. Most UVM verification environments are built using SystemVerilog, e, or a combination of both, and the `uvm_accel` library supports both. The `uvm_accel` package allows the proxy model part of a transactor to be written in SystemVerilog or e, whichever is the most suitable language, which is often the same as the language used to model the rest of the verification environment.

The BFM part of the transactor, implemented using the acceleratable subset of SystemVerilog and Verilog, uses the SCE-MI SystemVerilog interfaces to access the SCE-MI pipes based channels. These interfaces provide SystemVerilog tasks and functions that greatly simplifies the usage. More information about the SCE-MI Pipes interfaces can be obtained from the *Standard Co-Emulation API: Modeling Interface (SCE-MI) Reference Manual* from Accellera.

### 5.3 UVM Acceleration Package Interfaces

The UVM package provides two unidirectional interfaces, one to access input channels and the other to access output channels. The terms input and output are defined in relation to the hardware accelerator with input being into the hardware accelerator and output being out of the hardware accelerator.

For SystemVerilog, each interface is defined as a class that inherits from the `uvm_accel_pipe_proxy_base`. For e, each interface is defined as a unit that inherits from the `uvm_accel_pipe_proxy_base` unit.