INTRODUCTION
An assertion is a conditional statement that indicates the incorrect behavior of a design by flagging an error and thereby catching bugs. Assertions are used for validating a hardware design at different stages of its life-cycle, such as formal verification, dynamic validation, runtime monitoring, and emulation. Assertion-based verification provides significant benefits to the design and verification process. It aids easy detection of functional bugs, allows the user to find bugs closer to the actual cause, and ensures that bugs are found early on in the design process. Assertions bring immediate benefits to the whole design and verification cycle; thus any challenges engineers face in coding and testing them are worth resolving.

A callback in UVM is a mechanism for changing the behavior of a verification component (such as a driver or monitor) without actually changing the code of the component. The uvm_callback class provides a base class for implementing callbacks. Since extending a class from a uvm_callback is not a recommended coding practice, as it could lead to some potential ordering issues, callbacks in Questa® Verification IP (QVIP) are implemented by extending a base class and populating it with methods required to achieve callback implementation – such as replacing a sequence item with another sequence item containing a new set of attributes.

Assertion verification usually forms an integral part of the entire Verification IP development cycle. The first step is coding the assertions and the second step is validating those assertion scenarios through what is known as error injection. Callbacks can be very beneficial for this stimulus generation step as the verification engineer does not have to modify the source code and different scenarios can be tested simply by varying the extended classes. This is particularly useful in packet-based protocols, such as PCIe®, where fields are to be corrupted and callbacks provide fine-grained control. Thus callbacks increase the efficacy of the entire verification process, making it more robust.

This article covers how callbacks implemented in Questa® Verification IP can be used for assertion validation in designs using the PCIe® protocol.

CALLBACKS IN QVIP
In Questa® Verification IP, the callback methods and policies are first defined such that they are common across all protocols. For each specific protocol, the callback methods are subsequently extended from these base classes.

One of these base classes is called cb_policy, which extends itself from uvm_object. It defines methods such as register, get, and return callbacks. These methods are declared as pure virtual tasks or functions so that the derived class can override the base class definition.

A second base class extends from uvm_component and is called cb_manager. The run_phase() of this class is a forever loop in which first we get the sequence item from the BFM and then we return the changed sequence item back to the BFM. In separate external tasks defined inside the cb_manager class, callback sequence items are added and deleted from the callback queue and the register callback method is activated based on the queue size.

For each protocol specific callback implementation, each sequence item peculiar to the protocol extends itself from the aforementioned base classes. The names are assigned as: <sequence_item_name>_cb_manager for the class extending from cb_manager <sequence_item_name>_cb_policy for the class extending from cb_policy.
In the class `<sequence_item_name>_cb_policy`, the register, get, and return callback methods are populated based on the nature of each protocol. The process of enabling callbacks for a particular sequence item starts with enabling the agent settings in the build_phase of a test (the class extending from uvm_test), which instantiates the `<sequence_item_name>_cb_manager`. In the run_phase of the test, the register callback method is activated by calling the add callback task (defined inside `<sequence_item_name>_cb_manager`).

The main aim of the register callback method is to initiate the callback related task inside the BFM that sets a status variable. The get callback method gets called upon once the status variable inside the BFM is set and is used to get the sequence item from the BFM via DPI calls. The entire logic for manipulating the fields of the sequence item is placed in the `do_callback` task defined inside `<sequence_item_name>_cb_manager`. The altered sequence item is then returned back to the BFM via DPI calls, which is eventually driven onto the bus.

**ASSERTION TESTING VIA CALLBACKS**

Assertion testing is an indispensable part of the verification cycle. The job of the validation engineer is to make sure that the assertion gets validated in every relevant scenario. When assertion validation is to be carried out by manipulating the fields of the sequence item or by initiating a sequence item in a state in which it is not allowed, then callbacks can be of extreme importance. With QVIP callbacks, the entire basic structure is present and only the `do_callback` method is populated with each scenario; thus, callbacks ensure a faster way of assertion validation.

For a layered protocol like PCIe® where communication between receiver and transmitter elements takes place via structures known as packets, use of callbacks for assertion validation can improve efficacy. In QVIP, the packet information can be modified via callbacks and all packet-based assertions can be easily verified by making use of this mechanism. Now let’s take a look at a scenario from each of these layers.

**Case 1: Transaction layer based callbacks**

The high-level transactions occurring at the device core are known as transaction layer packets (TLP). QVIP allows alteration of particular fields of a TLP (both request as well as completion). So, if users want to validate an assertion related to the different fields of a TLP, they can simply inject an error into it via callbacks. For instance, the PCIe® protocol states that for a request of Length = 1 DW, the value of the Last Byte Enable field should be zero, where Last Byte Enable is the Byte Enable value for the last DWORD of the request.

![Figure 1: The basic sequence of events that take place when callbacks are enabled in an agent](image-url)
So, in simulation, if users want to inject this error to every Memory Write packet of length 1 DW, then they may do it as illustrated in Figure 2.

**Case 2: Data link layer based callbacks**

Data link layer packets (DLLP) are used for a variety of purposes; such as ensuring the integrity of TLPs, flow control, and power management. Just like for TLPs, callbacks can be used to inject errors in DLLPs.

For instance, lcrc is used for checking the data integrity of TLPs and DLLPs. lcrc is appended in the TLP at the data link layer (DLL) and is used for checking the integrity of the packet. If the value of lcrc attached with a packet is not the same as the calculated value, then it is a protocol violation. This incorrect behavior or assertion firing can be verified, as illustrated in Figure 3.

**Case 3: Physical layer based callbacks**

Perhaps the most efficient use of callbacks comes from modifying ordered set fields. In the cases of DLLPs and TLPs, a packet can still be executed via a sequence once linkup is done. But using a sequence to inject an error into an ordered set before linkup can be more convoluted and more susceptible to errors because the ordered set rules change with each LTSSM state. Callbacks, on the other hand, allow the user to inject an error in a much more controlled way. For instance, if the user intends to change the link number that gets transmitted in polling.active to non-PAD, one can do so using callbacks. If a control SKP ordered set is to be sent in place of a TS2 OS, one can easily use callbacks. And hence this invalid protocol scenario too can be easily validated via callbacks, as illustrated in Figure 4.
CONCLUSION
When a large number of assertions are to be validated, callbacks save time by making sure that the engineer does not have to code a new sequence for each scenario. It provides for more dynamic and fine-grained control. The scenarios covered in the article are very basic ways in which callbacks can be implemented. But another advantage of callbacks is the control it gives in terms of which packets are to be injected with an error. For instance, if an engineer needs to change the byte enable only for a TLP that originates from a certain Requester ID, the engineer can populate the do_callback method to implement this condition. Similarly, one can easily control the number of packets in which an error is to be injected. Callbacks allow for the creation of nuanced and complex stimulus creation with ease and are, hence, instrumental for assertion verification.

REFERENCES
i) PCI Express® Base Specification Revision 4.0 Version 1.0
ii) Cookbook - Online Methodology Documentation from the Mentor Graphics Verification Methodology Team
iii) GoldMine: Automatic Assertion Generation and Coverage Closure in Design Validation by David Sheridan, Lingyi Liu, and Shobha Vasudevan
The Most Comprehensive Resource for Verification Training

33 Video Courses Available Covering

- DO-254
- UVM Framework
- UVM Debug
- Portable Stimulus Basics
- SystemVerilog OOP
- Formal Verification
- Metrics in SoC Verification
- Verification Planning
- Introductory, Basic, and Advanced UVM
- Assertion-Based Verification
- FPGA Verification
- Testbench Acceleration
- Power Aware Verification
- Analog Mixed-Signal Verification

UVM and Coverage Online Methodology Cookbooks
Discussion Forum with more than 10,000 questions asked
Verification Patterns Library

www.verificationacademy.com