OCP 2.2 MVC Accelerates Verification Productivity

Mark Peryer – Verification Practice, Mentor Consulting

In April 2009, Mentor Graphics released the 2009.1 version of its Multiview Verification Component (MVC) library which featured an upgrade to its OCP-IP MVC to make it OCP-IP 2.2 compliant. The OCP MVC also supports the OCP 3.0 disconnect and WR_NP without write_response enable features. Another major feature of the 2009.1 version of the MVC library was that it was re-packaged to align the MVCs with version 2.0 of the SystemVerilog based Open Verification Methodology (OVM).

The OCP MVC, like every component in the MVC library, comes with a comprehensive kit of parts and supporting files. The OCP MVC can be configured to be used as either an OCP Master or an OCP Slave. The OCP master can be used to generate OCP compliant stimulus for an OCP slave using OVM sequences to generate bus requests. The MVC OCP slave acts as a responder which means that it is able to return an OCP compliant response to a bus request, again OVM sequences can be used to model the behaviour of the responder. Both the OCP master and slave components have protocol checking built into them which checks for malformed responses to requests. However, for exhaustive checking, the Questa Verification Library (QVL) OCP-IP protocol monitor packages up a set of assertions which checks the validity of OCP transfers on the bus.

The OCP-IP MVC comes with an OVM Monitor component that emits different transactions according to which bus events occur. These transactions can be used with either a functional coverage monitor or a scoreboard to build up a picture of which OCP transfer types have occurred or to check the behaviour of a system. A set of example environments is also supplied as part of the kit to illustrate how to use the OCP-IP MVC and implement verification environments with it. One of these example environments is an exhaustive OCP-IP bus compliance test which can be adapted to a target architecture by changing its constraints. A verification plan that is compatible with Mentor’s Questa verification tracker is also supplied to make it easier for users to prove that they have checked all applicable modes of operation, or that a particular test case has exercised the expected set of transfer formats. Again, this plan can easily adapted to suit the specifics of a particular system. Mentor’s Consulting group has implemented a number of customer verification environments based on the compliance example and has found that it can be productive in finding protocol implementation bugs in a matter of hours.

The 2009.1 release of the OCP MVC has been re-packaged to align it with the SystemVerilog OVM 2.0 verification component base class library. This effect of this is that has become straight-forward to implement an OVM verification environment that incorporates a MVC. The foundation of the OVM is an Open Source SystemVerilog class library that allows end users to quickly put together a productive verification environment based on clearly defined interfaces.
The major challenge of developing verification IP (VIP) for the OCP-IP 2.2 standard is that there are an almost infinite number of possible interface behaviours that can be defined using the OCP-IP parameters. This requires an OCP VIP to be easily configured to meet all potential use cases. In addition to this behavioural aspect, the verification component must be parameterizeable in order to allow it to cope with variations in the width of bus fields and the presence or absence of different sideband signals. This calls for a flexible approach to VIP configuration which in the case of the OCP MVC is done using a SystemVerilog class to capture all of the possible properties that might be used to describe the behaviour and parameters of an OCP interface. For ease of use, the names of the protocol specific parameters used match those defined in the OCP-IP standard.

Figure 1 - Generic MVC_Agent structure

The generic structure of a MVC is illustrated in Fig 1. All MVCs use a common mvc_agent class which always contains the same active component component classes. The active part of an MVC is always contained within a SystemVerilog interface which is connected to the pins of the design under test. The classes within the MVC agent are passed a configuration class object which encapsulates details of the set-ups for the MVC interface and, most importantly, a pointer to the MVC interface itself. This object is then used by the MVC agent to configure itself and its sub-components to behave in the correct way.
The stimulus side of the MVC is driven using the OVM sequence stimulus approach whereby OCP bus activity streams are split into sequence_items, or transactions, which are sent from a sequence outside the MVC to the mvc_sequencer which then sends them to the target mvc_driver for execution over the pin level bus interface. A sequence is a SystemVerilog class object that contains a time consuming method (task) known as ‘body’. The algorithm implemented in the body method is responsible for either generating streams of sequence_items or for building and executing sub-sequences which in turn would be responsible for generating stimulus using sequence items or further sub-sequences. In the case of OCP, a sequence item could represent a request for a single bus transfer or for a burst of bus transfers. The MVC will return response sequence_items which again can represent the result of a single transfer or a burst of transfers. MVCs use an extended ovm_sequence class called a mvc_sequence. The mvc_sequence has tasks embedded in it which can be made specific to the protocol which is being verified and these are passed through the driver into the MVC interface where they are executed.

The response monitoring side of the OCP MVC consists of a number of monitors which keep track of bus activity and then transmit an analysis transaction when specific bus behaviour has been observed. External components such as functional coverage monitors and score boards connect to analysis ports on the boundary of the MVC to subscribe to the broadcast analysis transactions. The number and purpose of the analysis ports is dependent on the configuration of the MVC.

The process of constructing an OCP verification environment using the OCP MVC starts by setting the parameter arguments which determine the size of the various bus fields, these same parameters have to be used to parameterise both the MVC SystemVerilog interface and the MVC configuration class. The most effective way to define and handle these parameters is to put them inside a container object. The next step is to connect the OCP bus pins on the design under test to the appropriate pins of the MVC OCP SystemVerilog interface, this is done in the structure of the static part of the test bench. Then an OCP MVC configuration class is declared and constructed, with its various fields configured to specify the required bus transfer characteristics. The fields inside the configuration object follow the same naming convention as the configuration parameters in the OCP-IP specification. Then a pointer to the MVC SystemVerilog interface is assigned to the m_bfm handle inside the configuration class. Once the configuration object has been configured it is put into the OVM configuration space using a set_config_object method, where the lookup key is set to the MVC top level configuration id string and the path field is set to the hierarchical path of the OCP MVC within the verification environment.

Once the environment is constructed, test cases can be implemented by writing sequences which can either be directed or use pseudo-random techniques. The OVM methodology allows for the creation of sequence libraries which facilitates a building block approach to stimulus implementation.
To support simulation debug, the OCP-IP MVC library supports a sophisticated graphical debug environment that allows bus transactions to be viewed at a high level of abstraction within Mentor’s Questa simulator. Bus phases can be debugged in a simple waveform window, but this involves having to reverse engineer transactions from signal transitions, something that is time consuming at best and a source of error at worst. The transactional GUI represents each bus transfer with a rectangle on the waveform window, allowing it to be mixed with signal viewing. In keeping with waveform signal visualisation, the life time of the transaction is represented by the horizontal length of the rectangle and the values of the various bus fields associated with the transaction are shown as a legend inside the rectangle. The transaction can then be broken down hierarchically into its individual phases and this allows rapid debug of phase or other errors. For instance, it may be that a burst request is met with a series of out of order responses, but not all of them are returned. In this case the transaction viewer will indicate at a high level that the burst transaction has not completed and then the break-down into the individual transactions will reveal which response transaction is missing.

![Figure 2 - OCP Transactions viewed alongside signals in the Questa signal waveform viewer](image)

Although integrating the OCP MVC into an OVM verification environment is straight-forward, a further level of automation is available through a utility which can auto-generate the OCP MVC component configuration classes and a test bench environment. The utility reads in a Core RTL configuration file, as defined in chapter 6 of the OCP-IP 2.2 specification, and is targeted at producing a complete OVM test bench for an OCP bus interconnect fabric. Typically, the bus fabric itself will be auto-generated and the RTL configuration file is either an input to, or an output from the generation process. The verification environment generation utility creates a series of OCP sockets which have OCP MVCs behind them in an OVM environment (ovm_env).
In the situation where the design under test (DUT) has other interfaces, the generated OVM environment can be added as a verification component class in a larger OVM test bench and used as a verification sub-system for the OCP sockets whilst the other DUT interfaces are stimulated and monitored by other components in the test bench hierarchy.

The 2009.1 MVC release puts a potent OCP-IP verification component within the grasp of OCP interface designers. The OCP-IP 2.2 MVC has a rich feature set which will boost OCP verification and debug productivity when deployed in any verification environment, but particularly in one based on the OVM.

Mark Peryer is a Program Manager in the Verification Practice at Mentor Consulting where he is responsible for implementing solutions based on Mentor Graphics verification products. Most recently this has involved the deployment of OVM based verification environments using MVCs. Prior to joining Mentor Consulting, Mark was the engineering manager at 3Soft where, amongst other duties, he served on the VSIA VSI on chip bus committee. Mark holds a B.Sc (Hons) degree in Electronic Engineering from Southampton University and is a Chartered Engineer.