ABSTRACT
The challenges inherent in verifying today’s complex designs are widely known. Verification IP (VIP) helps address these challenges assuming it meets a wide range of requirements. For example, it needs to be proven, provide checks to ensure protocol compliance, provide a comprehensive compliance test suite, and include the ability to collect and analyze coverage. It should also be straightforward to instantiate, configure and exercise, thus minimizing the time to productive verification. Questa Verification IP, part of the Mentor Enterprise Verification Platform, meets these requirements by providing a collection of features referred to as EZ-VIP.

EZ-VIP features easy-to-use connectivity modules, one-stop configuration, quick-starer kits and portable utility sequences. This article demonstrates these features while referencing the PCIe and AXI4 Questa VIP components.

INTEGRATION
The first step to deploying Questa VIP (QVIP) is integration within the test bench. A decision needs to be made regarding what the VIP will model - should it model a PCIe End Point (EP) or Root Complex (RC)? At the MAC or PHY level? Or perhaps it should be deployed in monitor mode (thus providing for protocol compliance checks and functional coverage without driving or responding to stimulus)? Once this decision is made, the hookup requirements are known and the interface can be instantiated and wired-up.

QVIP provides wrapper modules, specific to each of the different use cases, to ease the port connection process. These modules instantiate the QVIP interface and provide per lane signals. Users just need to instantiate these modules and make the connections to the Device Under Test (DUT). It is not necessary to explicitly instantiate a QVIP interface or “set” it within the UVM configuration database, since this is done inside the wrapper module. These modules can be found at the following path:

<QVIP_install_path>/examples/pcie/wrapper

The following wrapper modules are provided with PCIe QVIP for connecting a DUT via various interfaces:

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>QVIP Device Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC Serial</td>
<td>pcie_rc_serial module to be instantiated as RC which can be connected to EP DUT in serial mode</td>
<td></td>
</tr>
<tr>
<td>EP Serial</td>
<td>pcie_ep_serial module to be instantiated as EP which can be connected to RC DUT in serial mode</td>
<td></td>
</tr>
<tr>
<td>Pipe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC MAC</td>
<td>pcie_rc_pipe_mac module to be instantiated as RC MAC which can be connected to EP PHY DUT in pipe mode</td>
<td></td>
</tr>
<tr>
<td>RC PHY</td>
<td>pcie_rc_pipe_phy module to be instantiated as RC PHY which can be connected to EP MAC DUT in pipe mode</td>
<td></td>
</tr>
<tr>
<td>EP MAC</td>
<td>pcie_ep_pipe_mac module to be instantiated as EP MAC which can be connected to RC PHY DUT in pipe mode</td>
<td></td>
</tr>
<tr>
<td>EP PHY</td>
<td>pcie_ep_pipe_phy module to be instantiated as EP PHY which can be connected to RC MAC DUT in pipe mode</td>
<td></td>
</tr>
<tr>
<td>Monitor</td>
<td>pcie_monitor_mac_phy_module can be used as a monitor in pipe mode</td>
<td></td>
</tr>
<tr>
<td>PIE8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC MAC</td>
<td>pcie_rc_pie8_mac module to be instantiated as RC MAC which can be connected to EP PHY DUT in PIE8 mode</td>
<td></td>
</tr>
</tbody>
</table>
Another potentially daunting step that needs to be completed prior to productive work is configuration of the VIP. As Questa VIP is fully featured per the protocol specification, there are understandably numerous configuration options and parameters that may need to be specified. However QVIP provides a convenient manner to specify the basic getting-started configuration parameters. This is accomplished via a protocol-specific agent. This agent utilizes a simple agent descriptor struct that allows the user to easily specify the most common configuration values, thus jump starting the path to productive verification. A simple example is using the descriptor to specify if the QVIP should model a RC or an EP. Savings are realized when utilizing this approach rather than having to write the UVM code to create needed analysis ports. Additionally the user can specify which, if any, types of coverage are to be collected and if transactions and symbols are to be logged.

The struct and its usage are shown in the following code snippets (on this and the following page):

```
typedef struct {
    // PCIE QVIP Configuration
    // Set configuration options via a protocol-specific agent descriptor
    pcie_vip_config rc_agent_cfg; // handle of PCIe QVIP config class
}

env_cfg.rc_agent_cfg = rc_agent_cfg; // type pcie_vip_config
```
**Link Training and Enumeration**

The next step in bringing up a PCIe test bench is the Link Training and Enumeration process which is done by running the `pcie_bring_up_sequence`. This sequence provides the following functionality:

1. Initializes configuration space
2. Automatically trains the link to the desired speed
3. Waits for the DL_ACTIVE state indicating successful link up
4. Performs enumeration by starting sequence `pcie_tlp_bus_enumeration_sequence` if QVIP is configured as RC. It enumerates the PCIe fabric using the default BAR configuration.
5. Generates completions by starting `pcie_completer_sequence`. This sequence runs in the background and sends completion responses to DUT for requests received by QVIP.

The following code snippet shows how simple it is to perform the complex PCIe functionality shown above by simply starting this bring-up sequence in the run phase of test case.

```vhdl
rc_agent_cfg.agent_descriptor.pcie_details =
    '{version:PCIE_3_0,
gen:PCIE_GEN3,
if_type:PCIEPIPE};
rc_agent_cfg.agent_descriptor.bfm_node =
    '{node_type:PCIE_RC, mac:0};
rc_agent_cfg.agent_descriptor.dut_node =
    '{node_type:PCIE_NEP, mac:1};
rc_agent_cfg.agent_descriptor.options =
    '{ext_clock:1,
ext_reset:1,
en_sb:1,
en_pm_cvg:1,
log_txns:1,
negotiate_max_speed:1,
monitor:0};

// wait for link up and run enumeration
bring_up_seq.start(m_env.rc_agent.m_sequencer);
// start user sequence here
...// test complete
phase.drop_object(this, "Ending test");
endtask
```

**Quick Starter Kits**

Most of today’s SoC designs utilize third party design IP. For these scenarios, Questa VIP provides customized quick starter kits that are pre-configured to work with industry standard third party design IP. These kits reduce the time it takes to install, instantiate, configure and bring up QVIP to less than a working day. In the case of PCIe, the majority of designs are endpoints and thus kits are provided for the use case of QVIP as RC with the DUT as EP. Integration flow of the kits can be summarized in a few quick steps described below as well as depicted in figure 1.

1. **DUT TB connections**: Each of the kits provides a top level module that shows how to connect the PCIe design IP to a QVIP wrapper module. Separate kits are available for both serial and pipe connections.
2. **QVIP configuration**: Each kit contains a configuration policy class that contains a number of settings which have been found to be necessary in order to bring up the PCIe design IP during link training, these should not be edited. The EP configuration policy is designed to be used in the test class of the user’s testbench and it contains a method called `configure()` which is used to configure the PCIe QVIP configuration object.
3. **Link training and enumeration**: The link training has the potential to be the most problematic part of the process, but the settings in the kit configuration policies should mitigate this risk. Once the link is trained, the enumeration process is straight-forward and can also be accomplished using the generic `pcie_bring_up_sequence` as explained in the previous section.

**Application Verification**

At this point, the user can begin verification of their particular application. QVIP provides a comprehensive
test suite and library of sequences and sequence items for different packet formats, complex protocol flows, error injection, coverage and compliance tests. However, there is always a requirement of simple, easy-to-use sequences especially in the initial phase of verification. All protocols have a common objective to transfer data between two ends, and the simplest transactions can be represented as reads and writes. QVIP provides simple task based sequences to perform the read and write APIs. The following sections describe these easy-to-use read and write APIs.

**QVIP EASY-TO-USE READ/WRITE APIs**

QVIP provides read and write APIs which are generic in nature and thus protocol agnostic, meaning the same interface of read and write APIs are available for various ARM® AMBA® protocols (e.g. AHB, AXI3/4, ACE, and CHI). Simple sequences are also available to perform PCIe memory, config and I/O reads and writes.

The following table shows the generic API's available for the AMBA® family of protocols.

<table>
<thead>
<tr>
<th>Generic API's for the AMBA® Family of Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract Writes</strong></td>
</tr>
<tr>
<td>write8() Write a single byte of data at addr</td>
</tr>
<tr>
<td>write16() Write a single 16-bit word of data at addr</td>
</tr>
<tr>
<td>write32() Write a single 32-bit word of data at addr</td>
</tr>
<tr>
<td>write64() Write a single 64-bit word of data at addr</td>
</tr>
<tr>
<td>write() Write a variable-length array of data starting at addr</td>
</tr>
<tr>
<td><strong>Abstract Reads</strong></td>
</tr>
<tr>
<td>read8() Read a single byte of data from addr</td>
</tr>
<tr>
<td>read16() Read a single 16-bit word of data from addr</td>
</tr>
<tr>
<td>read32() Read a single 32-bit word of data from addr</td>
</tr>
<tr>
<td>read64() Read a single 64-bit word of data from addr</td>
</tr>
<tr>
<td>read() Read size number of bytes into data starting from addr</td>
</tr>
<tr>
<td><strong>Transactional R/W</strong></td>
</tr>
<tr>
<td>execute_txn() Execute a transaction whose attributes are specified in txn, of type rw_txn.</td>
</tr>
<tr>
<td><strong>Attributes API</strong></td>
</tr>
<tr>
<td>This API is used to set attributes that do not typically change with each transaction or that are determined by other means, e.g. an address_map.</td>
</tr>
</tbody>
</table>
The Read/Write API is for conducting basic reads and writes as if from a program running on a processor. It provides a generic protocol agnostic interface, letting the API sequence implementation choose the most appropriate bus transaction and corresponding attributes. These API methods have only a handful of simple arguments: address, data and id with no direct dependency on bus-specific attributes. Bus-specific attributes will be determined by less frequently changing configuration such as the address map and the attributes API. The following code snippet shows the generic function prototype.

```verilog
task write8 (ulong addr, ubyte data, int id = -1)
virtual task write8 (ulong addr, ubyte data, int id = -1)
```

Write the given data starting at address addr.

```verilog
task read8 (ulong Addr output ubyte data, input int id = -1)
virtual task read8 (ulong Addr output ubyte data, input int id = -1)
```

Read the given data starting at address addr. The optional id is used to identify a transaction stream. Writes and reads with the same id will finish in order of execution.

**USAGE OF READ/WRITE APIs**

QVIP provides a generic base class sequence, rw_api_base_seq, along with protocol specific sequences such as axi4_master_seq. The protocol-specific sequences are derived from the generic base class and provide the protocol-specific Read/Write API functionality.

The user code on the opposite page demonstrates extending the axi4_master_seq to perform the desired Read and Write operations.

**CONCLUSION**

The EZ-VIP features showcase the ability to rapidly deploy QVIP and quickly start productive verification of today’s complex SoC designs. These features include easy-to-use connectivity modules, one-stop configuration, quick-starter kits and portable utility sequences which are applicable whether verifying at the block level or within an end-to-end test bench, potentially containing multiple protocols, as shown in the diagram on the opposite page.

In addition to the ease of use features, Questa VIP is also fully featured in that it provides monitors to ensure protocol compliance as well as analysis components such as a scoreboard and coverage collectors. Comprehensive test suites and functional test plans are included thus allowing the verification team to track and achieve coverage goals. The building blocks are written in unencrypted SystemVerilog and encapsulated within a ready to be deployed UVM environment. In summary, Questa VIP provides a fast track to verification productivity via its EZ-VIP set of features as well as all the other tools necessary for exhaustive verification of complex protocols such as PCIe.
Please refer to `<QVIP_install_path>/examples/axi4/simple/sequence_lib` for complete example

class user_test_a #(
  int   AXI4_ADDRESS_WIDTH   =   32,
  int   AXI4_RDATA_WIDTH   =   32,
  int   AXI4_WDATA_WIDTH   =   32,
  int   AXI4_ID_WIDTH     =   4,
  int   AXI4_USER_WIDTH   =   4,
  int   AXI4_REGION_MAP_SIZE  =   16
) extends axi4_master_seq #( AXI4_ADDRESS_WIDTH, AXI4_RDATA_WIDTH, AXI4_WDATA_WIDTH, AXI4_ID_WIDTH,
  AXI4_USER_WIDTH, AXI4_REGION_MAP_SIZE)

virtual task body();
  ubyte    wr_data_8  = 'h11;
  ubyte    rd_data_8;
  ubyte    wr_data[] = new[16];
  ubyte    rd_data[];

  // write/read 1 byte
  write8('h1000, wr_data_8);
  read8 ('h1000, rd_data_8);

  wr_data = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

  // write/read data array
  write('h1000, wr_data);
  read ('h1000, rd_data, 16);
endtask
endclass
21 Video Courses Available Covering

- Intelligent Testbench Automation
- 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 5000 topics
UVM Connect and UVM Express Kits

www.verificationacademy.com