Chef's Special – an Efficient Verification Recipe for Maximizing Productivity While Using a Third Party Verification IP

Varun S,
Amit Sharma,
Abhisek Verma
Synopsys
svarun@synopsys.com
amits@synopsys.com
abhiv@synopsys.com

Bhavik Vyas
R2 Semi
bhavik@gmail.com

ABSTRACT

With the increasingly competitive time to market of a SOC, early closure of verification has started to gain a lot of momentum in the design and verification community. As a result, based on widely used and emerging protocols, standards-compliant third-party Verification IPs are rapidly being adopted to accelerate the development of a complete verification environment. However, for the adoption of Verification IPs, there are challenges pertaining to the integration, the protocol specific intricacies, the underlying complexity of the code, and the mechanism to create user-specific tests and inefficiently debugging the stimulus and responses. Some of the standard VIPs provide test suites to reduce the effort of the verification engineer to code protocol specific tests. Typically, such a test suite would be generic in nature and will not necessarily cater to different flavors required by the user for DUT specific scenarios.

There have been efforts in the past from vendors and semiconductor organizations alike to have an intuitive user interface for creating tests but those have not been very successful for different reasons. With the UVM Methodology, we will see how some of these challenges are mitigated. UVM1.0, which brings together the best practices and techniques across existing methodologies and the ones proposed by verification engineers across semiconductor organizations, now enables verification engineers and VIP providers to create highly configurable and dynamic environments. Using the example of the usage of a HDMI UVM VIP, this paper discusses how the relevant features in the UVM methodology can be leveraged to come up with a semi-automated mechanism of creating user specific tests over a third party VIP without having to be very well acquainted with the VIP infrastructure and the underlying protocol.

Categories and Subject Descriptors

HDMI, UVM, Re-use philosophy, Automation

General Terms

Verification, Design, Standardization.

Keywords

HDMI, VIP, UVM, CEA-861-D

1. INTRODUCTION

Use of third-party Verification IP has become ubiquitous these days. The ease of adoption of such IPs varies across different titles and vendors. These IPs are also available in multiple different flavors with respect to methodologies and the language of implementation. Hence, from a user perspective, it becomes worthwhile to invest some effort to come up with an efficient scheme to adopt them. If a generic approach can be evolved, then the schemes can be deployed across different titles and IPs from different vendors as well. The Universal Verification Methodology (UVM) saw the three major verification vendors along with multiple semiconductor organizations aligning on a single IEEE1800-SystemVerilog base class library implementation. Hence, it was decided to have a UVM compliant VIP as a vehicle to understand what kind of guidelines we can evolve to come up with a structured approach of efficiently leveraging a third party Verification IP for a standard protocol. The standard protocol picked up for this exercise was the High-Definition Multimedia Interface (HDMI) VIP. The configurability in the VIP's infrastructure which the underlying UVM methodology brings in was fundamental for most of these guidelines. Hence, we will also dwell on the capabilities that should be a part of VIP which enables users to leverage UVM methodology.

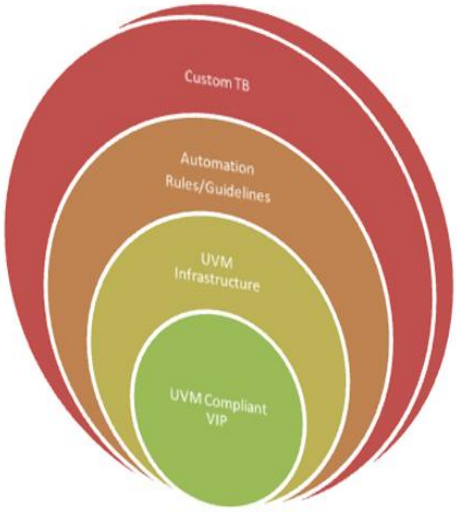

Figure 1: The Concept

As we explore different ways to effectively leverage Third Party VIPs, we come with a mechanism to create an automation layer on top of the UVM based testbench to help out in our objective.

Why do we typically need a Verification IP? Primarily, it is to generate appropriate stimulus which exercise all requirements from verification and to be able to monitor the responses and ensure that they are correct. In other words, earlier, we can come up with self checking tests which exercise the right stimulus, the closer we are to our objective. There is always a learning curve with the usage of third party VIP which can be reduced by ensuring a structured way of stimulus generation. The main focus of this paper is to reduce the test generation cycle for verification of any complex protocol based

DUT as shown in Figure-2.

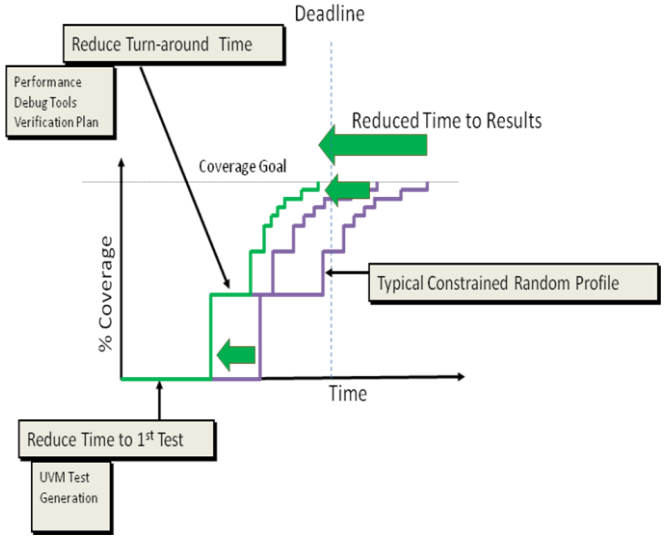

Figure 2: Objective: Reduce Time to Results With Third Party VIP

To start with, we demonstrate how a UVM compliant VIP enables us to create a highly configurable testbench template that can be designed to incorporate place-holders for hooking the DUT with the VIP to ease its integration with either a HDMI source or a sink RTL. The next step is to abstract this layer and provide the user with an automated way to create tests, application specific sequences and VIP configuration classes as extensions of the base HDMI VIP classes through a utility which reads the required information provided by the user. To give an example, a typical use of the HDMI VIP agent would be as a source in active mode or as a sink with EDID enabled and a sequence would constrain certain HDMI frame line parameters like values of R, G and B components during video active period either independently or depending on a certain HDMI VIP configuration parameter. A typical test would run different numbers of standard HDMI frames from source to sink of a certain format type and land the respective Video ID codes. Thus, based on the test flows, a distinct set of the permutations can appropriately be sequenced. The utility will map various HDMI VIP configurations or frame line parameters, tweak the relevant constraints and then create the logical order of atomic sequences as well as appropriate extensions of the VIP configuration classes.

2. HDMI PROTOCOL OVERVIEW

HDMI is a de-facto standard for digital connection for consumer electronics and PC products. It delivers highest quality audio/video signal over a single cable. HDMI system architecture is defined as consisting of Sources, Sinks, Repeaters, and Cable Assemblies. A given device may have one or more HDMI inputs, and one or more HDMI outputs. The HDMI cables and connectors carry four differential pairs that make up the TMDS data and clock channels as shown in Figure-3. These channels are used to carry video, audio and auxiliary data. Note that this paper doesn't talk about the Consumer Electronics Control (CEC) protocol associated with a typical HDMI device.

The HDMI source is responsible to send frames onto the TMDS interface, while a HDMI sink is responsible to receive them. The sink never responds back to the data from source. As shown in Figure-3, the HDMI link includes three TMDS data channels and a single TMDS clock channel. Each frame consists of a set of lines as per the HDMI specification. Each line is further segmented into video data, audio data and control periods. The complete feature list can be referred from [2].

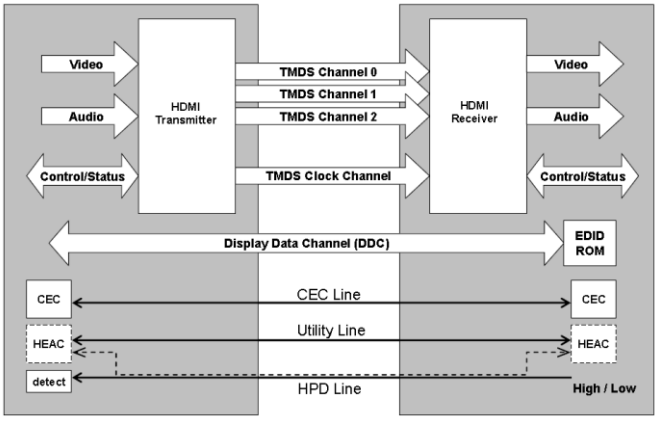

Figure 3: HDMI Source(Tx) and Sink(Rx) Block Diagram

Since the HDMI protocol supports a wide variety of audio and video formats, one of the major challenges in verifying all the different frames across all the different configurations.

3. HDMI UVM VIP ARCHITECTURE

Figure-4 shows the architecture of Synopsys SVT (SystemVerilog Technology) UVM based HDMI VIP.

In VIP, UVM compliant classes and attributes (members) are provided to represent the protocol activity and the characteristics of that activity. For example, a transaction object has members that define the type of audio and video information transmitted. A set of base classes provide common functionality and structure to form the foundation for the entire HDMI VIP system. We list down some of the features of UVM VIP which we would leverage later on to come with our VIP adoption guidelines.

Configuration: A protocol like HDMI gives the flexibility of working with different parameters. As an example, the device can take a varying number of frames to stabilize the video signal. Hence, to address all such requirements, we would need to bring in UVM Resource mechanism to provide the configurability required. The UVM VIP has a configuration class which is shared across all the components. This class is randomized in the *build_phase* and then propagated down to different individual components using UVM resource mechanism[6]. Thus, the individual components would reconfigure themselves dynamically at different points in time. If a user needs to change the configuration properties for specific tests, it requires setting constraints on a derived configuration class and overriding the configuration class in the environment using factories or through a uvm_config_db#(T)::set mechanism.

Stimulus generation: To stay consistent with the architecture of the HDMI and Consumer Electronic Control (CEC) protocols, a layered approach has been adopted by the UVM VIP with respect to stimulus generation. There are transaction classes corresponding to each of these layers (HDMI and CEC). These are typical UVM data descriptors, which translates to protocol specified frames.

Transaction Level Interfaces: UVM Analysis Ports to broadcast the required parameters to the coverage and the scoreboard models.

Extension Points: A rich set of UVM based callbacks have been provided across different layers enabling the user to add in project or test specific extension.

Data Exceptions: The extension points can additionally be used for changing the default stimulus and generate appropriate conditions for negative tests. A number of exception data classes have been defined within the VIP library for this purpose.

Factory Infrastructure: The VIP provides the user with the benefit of overriding the default behavior of VIP components by providing user specified extensions. This allows the user to meet the unpredictable needs of different tests.

Event Synchronization: A number of UVM events have also been provided which users can use for the purpose of decision making to synchronize their testbench with transition of data or states within VIP. Most events are tied to the HDMI standard but there are a few that are generic notifications from the data class.

Sequence Library: A rich set of sequences are available with HDMI VIP. These can be readily leveraged in user tests by setting them as the *default_sequence* of the HDMI source sequencer or by explicitly starting them on the HDMI source sequencer. These sequences help generate various types of HDMI compliant frames. These are the building blocks for the user to stitch together a complicated scenario if required.

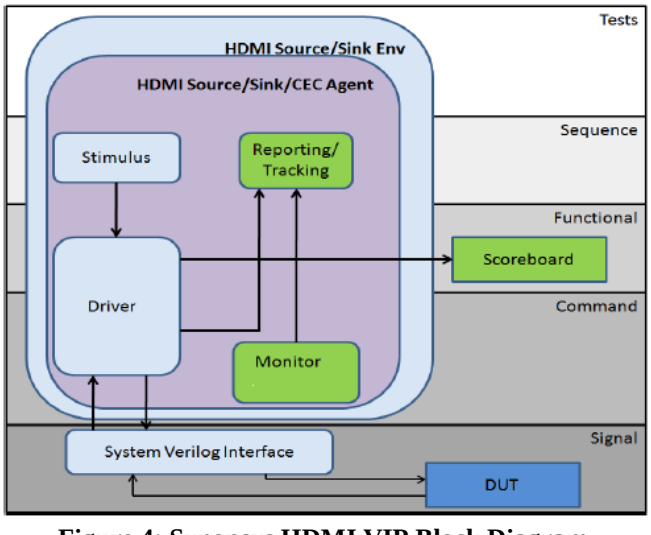

Figure 4: Synopsys HDMI VIP Block Diagram

3.1 VIP USAGE AND CONFIGURABILITY

The HDMI VIP can be configured to have either or both of the following two environments:

Source Environment - The Source Environment encapsulates the Source Agent and the CEC Agent (if CEC is enabled). It also contains the source configuration object and a virtual sequencer to orchestrate the HDMI and CEC sequencers.

Sink Environment - The Sink Environment encapsulates the Sink Agent and the CEC agent (if CEC is enabled). It also contains the Sink Configuration object and the CEC sequencer.

The typical use-model of the HDMI VIP is to be configured either as source or sink. This would require either of the Source/Sink Environment to be instantiated and hooked onto the TMDS interface. The Environment should be configured with the corresponding configuration object descriptor.

Both Source and Sink configuration objects encapsulates audio and video configuration objects to support various types of audio and video attributes such as ASP audio or 24 bit color video etc. The complete list of attributes can be referred from [3]. Additionally, these objects have parameters to control a host of features: the number of frames to be sent; enabling/disabling coverage etc.

These configuration objects are created in the UVM testbench. Their attributes are then set or randomized and then propagated to either the Source or Sink Environments using UVM Resource mechanism to get the individual VIP component configured. The various modes of operation and the complete feature list can be referred from [3].

4. FRAMEWORK FOR EFFECTIVE VIP ADOPTION

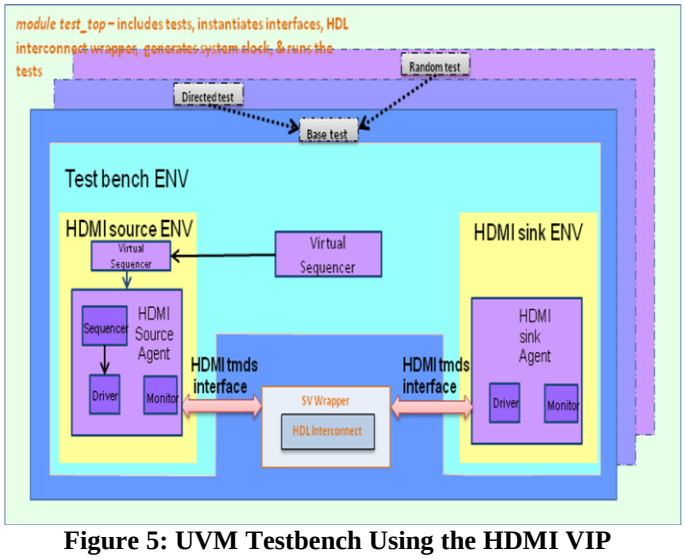

Figure 5: UVM Testbench Using the HDMI VIP

Figure 5 represents a typical UVM testbench. We will use this framework going forward to lay down our guidelines and create possible automation flows. The testbench comprises a HDMI VIP Source Environment and a Sink Environment connected back to back through an interconnect. The interconnect serves as a place-holder for the user to hook up either a Source DUT or a Sink DUT. Once the DUT is hooked up, the respective Sink Environment or the Source Environment (VIP) would be shunted out of the testbench. The Virtual Interfaces representing the Source and the Sink interfaces are propagated through the UVM configuration mechanism as well.

A collection of tests which derive from a common base test are a part of this framework. The base test essentially sets up the VIP hierarchy by instantiating the top level environment class. User tests extend from the base test to create scenarios based on the user requirements.

4.1 MAPPING TESTS TO TESTBENCH ATTRIBUTES

The following steps can be followed to create a structured approach for deciding the testcase creation or generation schemes:

Step 1: Enumerating mechanism and testbench attributes for tests creation

For each VIP, we need to understand the attributes which are available for the user to modify to create testcases. In a typical UVM testbench catering to the architecture described above, creating tests can be done in one of the following ways as described below. Though these are not exhaustive in terms of the different ways by which a test can be created:

1. Adding appropriate constraints on a derived configuration class and overriding the configuration class in the environment using factories of the UVM Configuration Mechanism
2. Adding appropriate constraints on a derived transaction class and overriding the transaction class in the sequencer using the factory infrastructure.
3. Creating new sequences by stitching together the sequences available in HDMI VIP sequence library. The sequence extensions can also be created with a slightly modified behavior. Once created, they are exercised by running them by setting the *default_sequence* property of the sequencer class or by explicitly starting it on a specified sequencer using the *start*() method of the sequence class.

Step 2: Mapping Verification Requirements to Testbench attributes

This step involves the mapping of the list of verification requirements in terms of a matrix of configuration attributes of the testbench. In a UVM testbench, these are typically the configuration and transaction descriptors.

For HDMI VIP, the following figure provides the matrix for three tests. For this VIP, most other tests would fit into this matrix and we need to add additional rows.

Test	Audio config	Video config	VIP config	Frame line
CEA-861-D-2	NO AUDIO	No deep color 720x480 p(2D)	VIP used as source. 2 frames	No of data island packets less than 50
CEA-861-D-5	One bit ASP	1920x1080i	VIP used as sink With EDID and no HDCP	--
CEA-861-D-1	ASP	24bit color 640x480 p @ 59.94/60 Hz	VIP as source 5 frames Def. coverage enabled	di_pkts distribution of ASP := 96, NULL := 4

Figure 6: HDMI Test Specification in Terms of the HDMI UVM VIP Configuration and Transaction Descriptors

Each row in the above matrix (Figure-6) corresponds to a user test to be created, while each column provides various attributes in terms of either configuration or transaction descriptors that needs to be manipulated on the HDMI VIP.

The first column specifies the name of the test as extracted from the verification planner or specification, the second column provides the information about the HDMI protocol specific audio configuration attributes in the VIP, the third column provides the information about the HDMI protocol specific video configuration attributes to be supported by the VIP and the fourth column provides the other configurable properties of the VIP (Enabling Extended Display Identification data (EDID) or enabling High-bandwidth Digital Content Protection (HDCP) etc). The fifth and the final column provide the attributes for the transaction descriptor of the HDMI UVM VIP which defines the audio, video or control data to be transmitted by each frame. For more complicated protocols, more columns would be required to map to the additional testbench attributes and the verification requirements.

Step 3: Manual Test Creation and Formulating Test Templates:

Based on the above matrix, the objective is to create a few minimal representative tests. This is used to serve as templates for the creation or generation of additional tests. For example, the test "CEA-861-D-1" from the third row of the matrix in Figure-6. This requires creating an extension of the configuration class to override the specific values of the Audio and Video configurations.

Step 3a: Creating the custom system configuration class

```
class cust_svt_hdmi_system_configuration
                              extends svt_hdmi_system_configuration;
  `uvm_object_utils(cust_svt_hdmi_system_configuration)
 function new(string name ="cust_svt_hdmi_system_configuration");
   super.new(name);
   /**
    * Enable Coverage collection for this testcase
    */
   this.src_cfg.enable_hdmi_cov = 1'b1;
 endfunction

 constraint basic_system_cst {
   /**
    * Source is configured to transmit 5 frames
    */
   src_cfg.no_of_frames == 5;
   /**
    * Source Video format is configured as FORMAT_2D
    */
   src_cfg.video_cfg.video_format_type == FORMAT_2D;
   /**
    * Source Video resolution set to 640*480p@59.94/60 Hz
    */
   src_cfg.video_cfg.video_id_code == 1;
   /**
    * Source Color depth is configured to COLOR_DEPTH_24
    */
   src_cfg.video_cfg.color_depth == COLOR_DEPTH_24;
   /**
    * Source Audio Type is configured as ASP_AUDIO
    */
   src_cfg.audio_cfg.audio_type == ASP_AUDIO;
   }

endclass : cust_svt_hdmi_system_configuration
```

Figure 7: Custom Configuration Class For CEA-861-D-1 Testcase

As seen above, the values are assigned in the extended class. In specific cases, it might be possible to avoid the extension and use the Resource mechanism if the values have to be overridden for a small number of parameters.

The custom configuration class (*cust_svt_hdmi_system_configuration*) as shown in Figure-7 extends from the base system configuration class provided by the VIP. The base system configuration class is the container for the Source and the Sink configuration objects, which are used to configure either a Source or a Sink VIP.

Each of them encapsulates lower level configuration objects such as audio and video configuration objects to define various audio/video attributes supported by the VIP in a particular configuration.
The Audio config column in the matrix shown in Figure-6 maps to the attributes of the attributes of *audio_cfg* object handle. Similarly the Video config column maps to the attributes of *video_cfg* object handle available inside Source or Sink configuration handle. The mapping is performed as a constraint on a particular attribute of the configuration handle with the values provided in the matrix. Additionally, the fourth column of the matrix in Figure-6 provides an additional constraint to restrict the number of total frames to be sent by the source and enables the default coverage model of the source.

Step 3b: Creating custom sequence
Figure-8 shows a custom sequence class created from the frame line column information provided by the matrix in Figure-6. The custom sequence extends from the base sequence available in the sequence library as a part of the HDMI UVM VIP.
All the user sequences are extended from the HDMI VIP specific base sequence. This ensures that the sequences reuse the code for raising and dropping of objections in the appropriate methods. This custom sequence adds constraints to the request before sending it to the sequencer. These constraints are primarily governed by the information provided in the frame line column of the above matrix in Figure-6. For the CEA-861-D-1 test, the frame line column expects the data island packets (audio period) to be distributed over ASP or NULL types in the probability of 0.96 and 0.04 respectively. The constraint for the same is not shown below (hidden inside the macro call in the code snippet below), but can be easily realized as a distribution constraint. Another key aspect as shown in Figure-8 is that the sequences are made 'configuration aware' through the parent sequencer handle on which they are executed. The parent sequencer retrieves the VIP configuration from the UVM configuration database. This helps in creating highly configurable sequences which can be reused much more easily.

```
class cust_svt_hdmi_source_random_sequence
                  extends svt_hdmi_base_sequence#(svt_hdmi_frame_line);
  `uvm_object_utils(cust_svt_hdmi_source_random_sequence)
  `uvm_declare_p_sequencer(svt_hdmi_source_sequencer)

  virtual task body();

   repeat(p_sequencer.src_cfg.no_of_frames) begin
     for (int i=0; i<(p_sequencer.get_vtotal());i++) begin
       `uvm_create(req)
       req.line_no = i;

       /**
        * Randomize the req transaction with specified constraints.
        */
       `uvm_rand_send_with(req,
        {
       `SVT_HDMI_SOURCE_BASE_LINE_SEQUENCE_CONSTRAINT_DI_PKTS
         })
     get_response(rsp);
     end
   end
  endtask: body
endclass: cust_svt_hdmi_source_random_sequence
```

Figure 8: Custom Sequence

Step 3c: Custom testcase:
Given that the custom configuration class and user defined sequence were created, they are now applied to the test environment. The custom test class (*random_test* here) extends the base test. During the build_phase of the test, it sets the custom configuration onto VIP by using *uvm_config_db::set()*. Similarly, it sets the *default_sequence* property of the source sequencer using *uvm_config_db::set()* which causes the sequencer to execute the user defined sequence in the appropriate phase. The code for the same can be referred from Figure-9.

```
class random_test extends hdmi_base_test;
  `uvm_component_utils(random_test)
  cust_svt_hdmi_system_configuration sys_cfg;

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    sys_cfg = new();
    /**
     * Set the source configuration for sub-components.
     */
    uvm_config_db#(svt_hdmi_source_configuration)::set(
                  this,"env.source_env", "cfg", this.sys_cfg.src_cfg);
    /** Apply the random sequence to the source sequencer*/
    uvm_config_db #(uvm_object_wrapper)::set
                  (this,
                   "env.source_env.source.sequencer.main_phase",
                   "default_sequence",
                   cust_svt_hdmi_source_random_sequence::type_id::get());
  endfunction: build_phase
endclass
```

Figure 9: Custom Testcase

This test can be invoked by passing the testname to the +UVM_TESTNAME runtime argument. Thus, in the context of the HDMI VIP, we have created a specific test by creating three new classes, and ensuring that they are picked up by the simulation. After executing Step 3, we essentially have a template for creating further tests. The template is basically a collection of these three classes extended appropriately. This template can now be used as the vehicle to quickly generate the required code to be inserted depending on the parameters specified in the matrix created earlier to map to the complete set of verification requirements.

4.2 ENABLING AUTOMATION

Once a structured approach or template has been identified to create new tests, the next logical step to accelerate verification closure is to provide an automation layer. For a protocol like HDMI, simple parsing and pattern matching utilities can be used to fill up the templates that serve as a framework for new tests. For this exercise, we created a 'test-gen' utility written in PERL that reads the 'matrix' which can be captured in a spreadsheet. As mentioned earlier, each column in the spreadsheet represents certain configuration descriptor or transaction descriptor. To make it easier for the end user to be aware of the attributes to select, each column has a drop-down list with the valid configuration or transaction attributes. Once the selection is made, the corresponding tests can be generated.

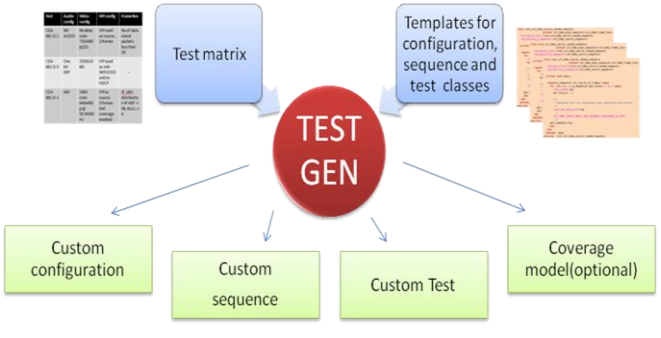

Figure 10: HDMI Test Gen Utility

The set of steps governing the automation are restated as follows:

1. The custom configuration, sequence and the test template classes are created and registered in the test generation utility. This is done once for each VIP.
2. The spreadsheet is parsed by the 'test-gen' utility to identify the attributes that are members of the VIP configuration classes. The spreadsheet is customized for each VIP so that it only lists a set of valid attributes for different configuration and transaction descriptors. This ensures that the selection is always valid.
3. In case of HDMI VIP, once the attribute and its value has been identified, a suitable constraint is created for it and dumped inside the custom configuration class for attributes in column 2,3 and 4, while it is dumped inside the sequence class for attributes in column 5. These constraints define the type of audio and video information transmitted by that frame. (This completes the custom configuration and the sequence classes and they look like the ones shown in figures 7, 8 and 9.)
   Note that mostly the constraints are regular assignments. But any SV constraint blocks would be mapped to the appropriate configuration or transaction descriptor.
4. For specific VIPs, the generator can also switch on/off different checks and coverage bins based on the chosen configuration..

The steps described above are generic steps and the generation infrastructure can suitable be enhanced for more complex protocols.

5. GUIDELINES FOR A GENERIC VIP ARCHITECTURE TO ENABLE TEST AUTOMATION

We can see that a testcase generation infrastructure can be extended to any VIP given that it adheres to a certain set of guidelines as outlined below:

1. One of the most important guideline is to have the VIP code strictly adhere to the UVM guidelines and best practices. Each component inside VIP should enable UVM features such as configurations, factory infrastructure and callback. These are the cornerstones of reusable and configurable testbenches.
2. The VIP configuration and transaction descriptors must have the right set of constraints which are closely tied to the specification. They should govern the valid ranges of all the parameters inside the configuration or the transaction data classes.
3. For more complicated scenarios, VIPs should provide a rich set of sequences to enable user to use them as building blocks.
4. Most of the VIP components should be made aware of the configuration. This ensures that the VIP configuration can be made available to data objects as well. For example, once the sequencer is made configuration aware, the sequences running on the same can request the same through the handle of the parent sequencer and use them accordingly as shown in Figure-8.
5. One of the interesting features of the HDMI UVM VIP which helped us debug while using the test generation infrastructure was the *is_valid* check done by the VIP on the transaction and the configuration data object before using it.
   Primarily, these checks are available inside the configuration and the transaction classes to ensure that these objects do not assume invalid values, either not supported by the VIP or illegal as per the protocol.
6. Use of strongly typed 'set' and 'get' while using *uvm_config_db* by the VIP ensured correct assignments especially if such testbench codes is automatically generated. Whenever the VIP retrieves a value through *uvm_config_db::get*, it is provided a check to ensure that the '*get'* was correctly executed.
7. The default coverage model should have the coverage parameters initialized through *uvm_config_db::get()*. Also, the covergroups should be enabled/disabled through configuration parameters. This helps to ensure that only the relevant covergroups are enabled for different simulations and the covergroups are also made configurable given that the SystemVerilog language does not provide coverage extensions.
8. As a general guideline to VIP architecture, a consistent language and methodology gives the best testbench methodology support and performance. Mixed language VIP can often do the job but results in unwieldy language translations that hamper methodology support and limit performance. This will have a continual drag on productivity. With SystemVerilog having gained acceptance as an industry standard, a pure SystemVerilog based VIP would produce the best results.

6. RESULTS & CONCLUSION

We introduced Figure-2 earlier as our objective of creating a set of guidelines and infrastructure for a more efficient adoption of third Party VIPs. By following a set of steps and leveraging the UVM Compliant HDMI VIP, we were able to put down specific guidelines which helped us to create a test generation infrastructure. This helped to generate tests targeting a big percentage of tests aimed at verifying DUT interfaces compliant to the protocol. A structured approach towards planning the verification requirements and mapping them to the capabilities of a VIP upfront would help us create the appropriate framework for accelerating the process of adding incremental tests. Such an approach reduces the time to use a third party VIP and the overall effort spent on VIP specific code development. Thus, with minimal user involvement, the user is able to create and control the required testcases that are desired and thus concentrate on converging and completing the verification tasks efficiently. Though, the UVM based HDMI VIP was defined to demonstrate this flow, the various approaches, guidelines and techniques described above can be well leveraged with other VIPs and methodologies across various constrained random verification environments to increase the verification productivity of the end users.

7. REFERENCES

[1] UVM user guide
[2] HDMI-1.4 Specifications
[3] Synopsys HDMI UVM VIP user guide
[4] CEA-861-D Specifications.
[5] UVM Reference Guide
[6] Advanced Testbench Configuration with Resources, Mark Glasser, Mohamed Elmalaki