Application Abstraction Layer: The Carpool Lane on the SoC Verification Freeway

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

Subramanian Kuppusamy
Qualcomm
skuppusa@qti.qualcomm.com

1. INTRODUCTION

Based on widely used and emerging protocols, standards-compliant third-party Verification IP (VIP) is rapidly being adopted to accelerate the development of a complete verification environment. Since the High definition Multimedia Interface (HDMI) supports a wide variety of audio and video formats, one of the major challenges in verifying it is to create all kinds of stimulus responsible to generate various types of frames. At Qualcomm we chose Synopsys HDMI VIP as it was compliant with the HDMI-2.0 specification, adheres to the latest verification methodologies such as Universal Verification Methodology (UVM) and was developed using SystemVerilog.
The main focus of this paper is to reduce the system level test generation cycle for verification of any complex protocol based DUT. Using the example of a HDMI UVM VIP based application layer, this paper discusses how an application layer can ease the process of creating application specific tests over a third party VIP without having to be very well acquainted with the VIP infrastructure and the underlying protocol. Such an approach also cuts on the learning curve associated with the usage of a third party VIP.
The HDMI VIP uses UVM-compliant classes to represent protocol activity and the characteristics of that activity. For example, a transaction object has members that define the audio and video information being transmitted. A set of base classes provide common functionality and structure to form the foundation for the entire HDMI VIP.
In a testbench, an HDMI VIP agent can be a source in active mode or as a sink with EDID enabled optionally. Stimulus is created as a UVM sequence with constrained random values for frame line values of R, G and B components during video active period. These values can be randomized either independently or could depend on certain HDMI VIP configuration parameters. A test would run many standard HDMI frames from source to sink of a certain format type.
The HDMI VIP has a default functional coverage model implemented as user extensions or callbacks to the HDMI monitor. It has a rich set of covergroups on HDMI configuration and HDMI frame line class for comprehensive functional coverage. Enhanced debug and validation is provided by allowing the user to read in audio video data, by-passing the built-in generation infrastructure. It incorporates place-holders for hooking the DUT on one side and C based model on the other thus enabling reuse of C based models for stimulus generation. To increase throughput per test, the UVM phasing mechanism is leveraged to revert to the post-configure phases to line up multiple tests in one simulation.

This paper demonstrates how a UVM compliant VIP enabled us to create a highly configurable testbench which can be re-used at multiple stages of verification, validation and prototyping.

Categories and Subject Descriptors

HDMI, UVM, Re-use philosophy

General Terms

Verification, Design, Prototyping

Keywords

HDMI, VIP, UVM, CEA-861-F, VESA, DMT, HDCP.

2. HDMI PROTOCOL OVERVIEW

The HDMI is the 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.
A HDMI source is responsible to send frames onto the Transition-minimized differential signaling (TMDS) interface, while a HDMI sink receive them. The sink never responds back to the data from source. As shown in Figure 3, an 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].

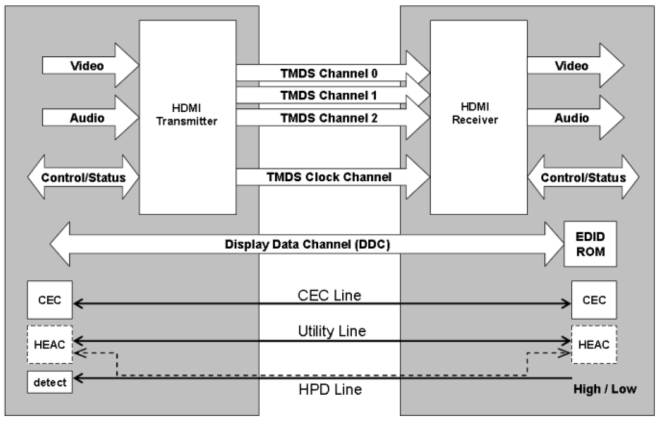


Figure 1 :- HDMI source(Tx) and sink(Rx) block Diagram

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

3. HDMI UVM VIP ARCHITECTURE

Figure-2 shows the architecture of Synopsys SVT (SystemVerilog Technology) UVM based HDMI VIP.

Here are some of the features of the UVM VIP which are in our VIP adoption guidelines.

Configuration: A protocol such as HDMI gives the flexibility of working with different parameters. For 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 the UVM Resource mechanism to provide the configurability required. The UVM VIP has a configuration class which is shared across all components. This class is randomized in the build phase and then propagated down to different individual component using the UVM Resource mechanism [6]. This sharing allows individual components to reconfigure themselves dynamically at different points in time. If a user needs to change the configuration properties for specific tests, it would require setting constraints on a derived configuration class and overriding the configuration class in the environment using factories or through UVM configuration 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 for stimulus generation. There are transaction classes for each of these layers (HDMI and CEC). These are typical UVM data descriptors which will translate to protocol specified frames.

Transaction level Interfaces: UVM analysis ports broadcast the required parameters to the coverage and scoreboard models.

Extension points: The VIP provides a rich set of UVM based callbacks across the different layers so we can add in project or test specific extensions.

Data Exceptions: The extension points can also be used for changing the default stimulus and generate appropriate conditions for negative tests. A number of exception data classes are 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: Many UVM events are provided so users can synchronize their testbench with transition of data or states within the 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 the HDMI VIP. These can be readily leveraged in 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.

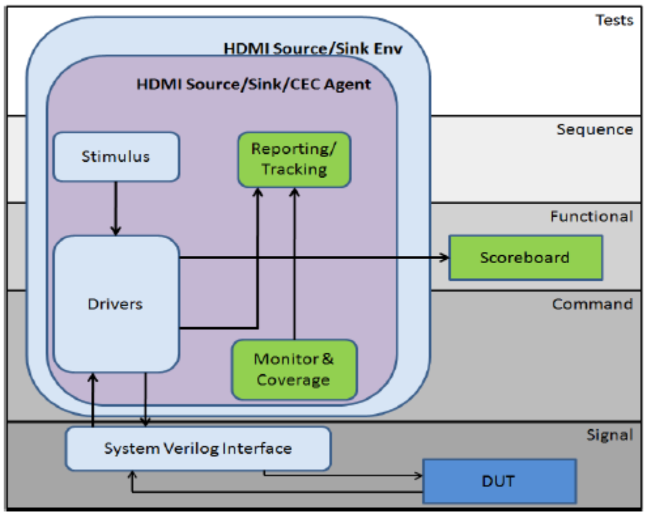


Figure 2 :- Synopsys HDMI VIP Block Diagram

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 HDMI VIP can be configured either as source or sink. This requires 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 encapsulate audio and video configuration objects to support various types of audio and video attributes such as ASP audio or 24bit color video etc. The complete list of attributes can be referred from [3]. Additionally, these objects have parameters to control a host of features such as 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 then propagated to either the Source or Sink Environments using the UVM Resource mechanism to configure the individual VIP components. The various modes of operation and the complete feature list can be referred from [3].

4. The Testbench

The structural testbench integration was done as shown in the figure 3 and 4 below. It uses the Synopsys UVM compliant VIP and rest of it is Qualcomm proprietary testbench component to verify the audio and video data path of HDMI based design.

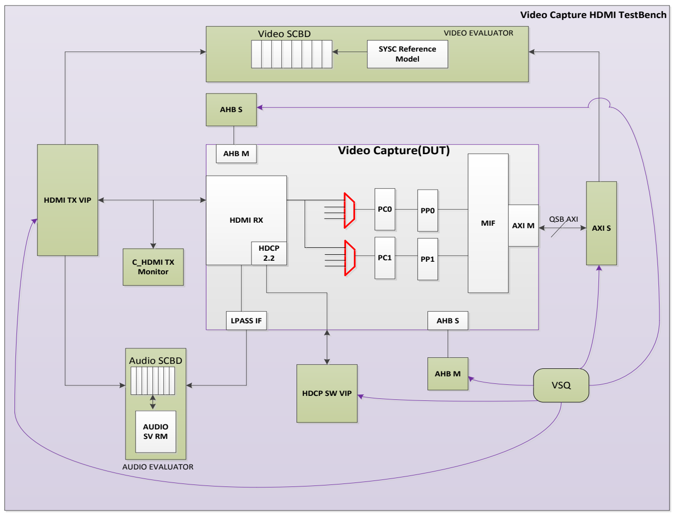


Figure 3 :- Synopsys HDMI VIP integrated in Qualcomm testbench.

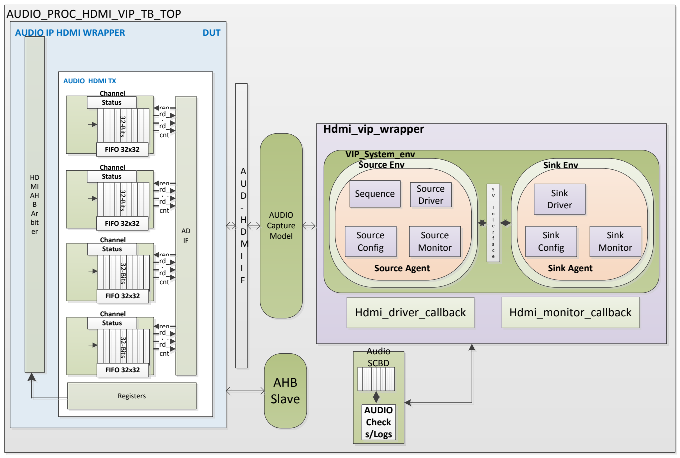


Figure 4 :- Synopsys HDMI VIP integrated in Qualcomm testbench.

The DUT is a video capture and processing subsystem that is a front-end to various video interfaces, including HDMI. The DUT consists of a set of DSP-intensive video processing blocks which process incoming analog/digital video streams and stores them into system memory.

The video processing blocks within the subsystem are modeled using SystemC reference models which are simulated within a C/C++ reference testbench as shown by the orange components in Figure 3. The SystemC reference models are used to generate reference memory dumps from a given video frame(s). The C/C++ reference testbench is a pure software testbench which simulates only the SystemC reference models and generates reference memory dump files used for comparison.

In the other configuration (blue components in Figure 3) the C/C++ RTL testbench instantiates the HDMI Source VIP and the DUT, and the same video frame(s) is passed to the HDMI Source VIP which transmits it to the DUT. The DUT processes the video stimuli(s) from the HDMI VIP and generates a memory dump which is then compared to the memory dump generated by the reference testbench.

Since the reference testbench is used to validate the results from the RTL testbench, the video configuration and the video stimuli needs to be kept consistent between the C/C++ reference testbench and C/C++ RTL testbench.

4.1 Application layer

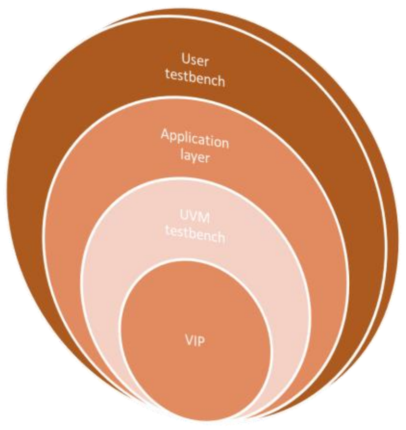


Figure 5 :- Application layer concept

The application layer comprises code which translates the high level scenarios to be tested into the low level configurations and stimulus to be provided to the VIP as shown in Figure 5 above.

A typical test would run different numbers of standard HDMI frames from source to sink of a certain format type land the respective Video ID codes. Thus, based on the test flows, a distinct set of the permutations can appropriately be sequenced. The application layer 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. At the same time, the covergroups either on HDMI configuration or HDMI frame line class are interpreted to generate a suitable coverage and scoreboard model, and the respective ports (HSYNC/VSYNC/DE/VD) made available for debug as required. In the case of image processing, an enhanced debug and validation experience can be provided by allowing user intervention in reading in images by-passing the generation infrastructure. To increase throughput per test, the UVM phasing mechanism needs to be leveraged to revert to the post-configure phases to line up multiple tests in one simulation.

Thus, with minimal user involvement, the user is able to create and control the required testcases that are desired and thus concentrate on converging in completing the verification tasks efficiently. Though, the UVM based HDMI VIP was defined to demonstrate this flow, the various approaches and 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 end users.

4.2 Test Generation Cycle

One of the focus of this paper is to reduce the test generation cycle for verification of any complex protocol based DUT. The typical verification cycle of a DUT is depicted in figure 6 below. Once a robust UVM testbench is created the test generation is one of the major road-block as it is a very time consuming process especially when using a third party VIP. 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.

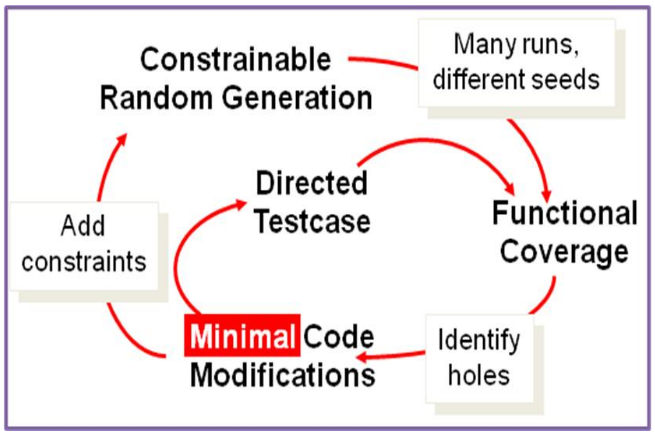
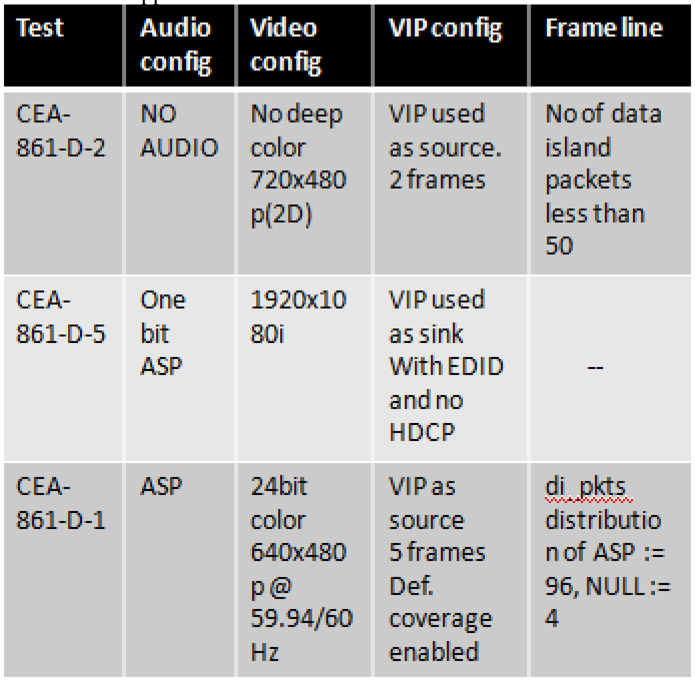

Constrainable Random Generation
Many runs, different seeds
Directed Testcase
Functional Coverage
Add constraints
Minimal Code Modifications
Identify holes

Figure 6 :-Typical Verification cycle of a DUT.

4.3 Application Scenarios

The typical application scenarios to test using the HDMI VIP are listed in Figure 7 below. It is a subset of all possible scenarios but in line with the application software.



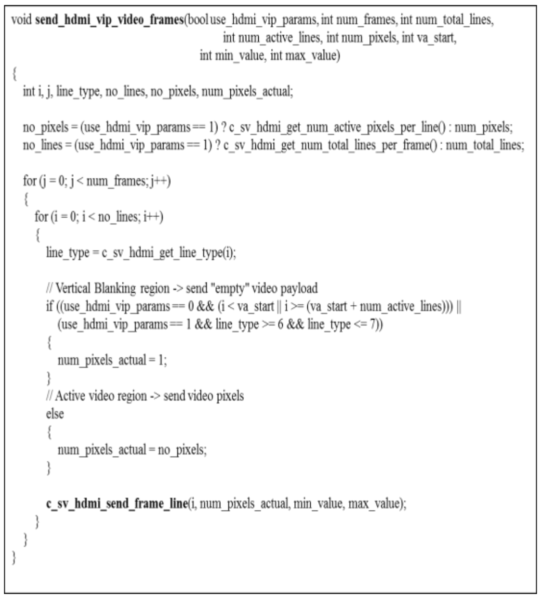
Test	Audio config	Video config	VIP config	Frame line
CEA-861-D-2	NO AUDIO	No deep color 720x480 p(2D)	VIP used as source. 2 frames	No of data island packets less than 50
CEA-861-D-5	One bit ASP	1920x1080i	VIP used as sink With EDID and no HDCP	--
CEA-861-D-1	ASP	24bit color 640x480 p @ 59.94/60 Hz	VIP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

Figure 7 :-Application scenarios for HDMI protocol verification.

4.4 Stimulus Generation

Because the C/C++ reference testbench was used to generate golden memory dumps for comparison, identical video frame stimuli were applied to both the SystemC reference models and the DUT. Though the HDMI VIP has the capability of generating random video frames, it was required that the video frame stimuli needed to be generated from C/C++ test library and sent to the HDMI VIP via DPI-C functions/tasks. A DPI-C export task and a DPI-C import function were used for passing video horizontal frame lines from C/C++ test library to HDMI VIP. The DPI-C export task *sv_hdmi_send_frame_line()* is invoked from C/C++ test library, and within the task a DPI-C import task *c_hdmi_get_frame_line_pixels()* is invoked from HDMI VIP to retrieve the frame line from C/C++ test library and pack it to the HDMI VIP sequence item for transmission.

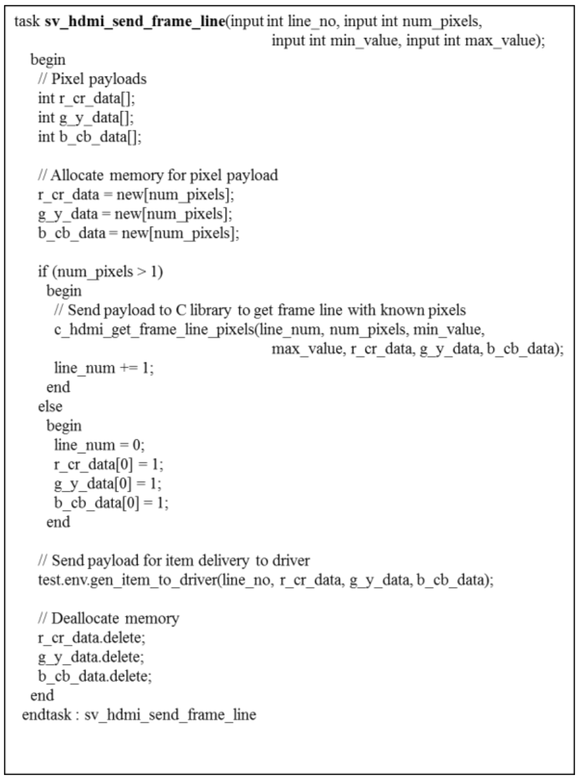
By allowing the HDMI VIP to retrieve video frames from the C/C++ test library, the same video frame stimuli can be applied to both testbenches for producing golden memory dumps.



```
void send_hdmi_vip_video_frames(bool use_hdmi_vip_params, int num_frames, int num_total_lines,
                                            int num_active_lines, int num_pixels, int va_start,
                                        int min_value, int max_value)
{
  int i, j, line_type, no_lines, no_pixels, num_pixels_actual;

  no_pixels = (use_hdmi_vip_params == 1) ? c_sv_hdmi_get_num_active_pixels_per_line() : num_pixels;
  no_lines = (use_hdmi_vip_params == 1) ? c_sv_hdmi_get_num_total_lines_per_frame() : num_total_lines;

  for (j = 0; j < num_frames; j++)
  {
    for (i = 0; i < no_lines; i++)
    {
      line_type = c_sv_hdmi_get_line_type(i);

      // Vertical Blanking region -> send "empty" video payload
      if ((use_hdmi_vip_params == 0 && (i < va_start || i >= (va_start + num_active_lines))) ||
        (use_hdmi_vip_params == 1 && line_type >= 6 && line_type <= 7))
      {
        num_pixels_actual = 1;
      }
      // Active video region -> send video pixels
      else
      {
        num_pixels_actual = no_pixels;
      }

      c_sv_hdmi_send_frame_line(i, num_pixels_actual, min_value, max_value);
    }
  }
}
```

Figure 8 :- The C Side

Figure 8 and 9 show the code snippet used for interaction between the C stimulus generation and SystemVerilog HDMI VIP.



```
task sv_hdmi_send_frame_line(input int line_no, input int num_pixels,
                                         input int min_value, input int max_value);
  begin
    // Pixel payloads
    int r_cr_data[];
    int g_y_data[];
    int b_cb_data[];

    // Allocate memory for pixel payload
    r_cr_data = new[num_pixels];
    g_y_data = new[num_pixels];
    b_cb_data = new[num_pixels];

    if (num_pixels > 1)
      begin
        // Send payload to C library to get frame line with known pixels
        c_hdmi_get_frame_line_pixels(line_num, num_pixels, min_value,
                                         max_value, r_cr_data, g_y_data, b_cb_data);
        line_num += 1;
      end
    else
      begin
        line_num = 0;
        r_cr_data[0] = 1;
        g_y_data[0] = 1;
        b_cb_data[0] = 1;
      end

    // Send payload for item delivery to driver
    test.env.gen_item_to_driver(line_no, r_cr_data, g_y_data, b_cb_data);

    // Deallocate memory
    r_cr_data.delete;
    g_y_data.delete;
    b_cb_data.delete;
  end
endtask : sv_hdmi_send_frame_line
```

Figure 9 :- The SV side

4.2 HDMI VIP and Reconfiguration

The correct configuration for the VIP was known sometime during the *run_phase*. On the other hand, the source model needed the correct configuration to start the model during *build_phase*. This called for a reconfiguration of the HDMI VIP during run phase as shown in Figure [10]. This enables the integration of the HDMI VIP into non-UVM testbenches. Of course some of the configuration attributes are static and cannot be changed during run_phase but most of them being dynamic enabled a robust reconfiguration.

```
begin
   new_sys_cfg=new();
   // wait for C-side to push the config values
   wait_for_cfg_from_c(new_sys_cfg);
   //randomize the system configuration
   new_sys_cfg.randomize();
   //re-configure the model
   env.source_env.reconfigure(new_sys_cfg.src_cfg);
   // wait for all the components to sync-up
   repeat (2) @(posedge env.source_env.hdmi_if.tmds_clk);
 end
```

Figure 10 :- reconfiguring the model

The new configuration was randomized to ensure all the configuration parameters obey the reasonable and valid constraints as part of the VIP. These constraints ensure that the protocol specification is not violated. For example, in Digital Visual Interface (DVI) mode the VIP ensures there are no data island packets by constraining the attribute *no_of_di_pkt* of the frame line class to zero. The user-constraints added for this testbench ensured that some of the configuration values which came from C-side got applied to the system configuration of the HDMI VIP.

5. Feature Verification

Hooking up of the Protocol Analyzer helped a lot in debugging purposes as shown in Figure [11]. It provided a GUI based view of the transaction and its synchronization with the waveforms helped a lot in debugging at the transaction level.

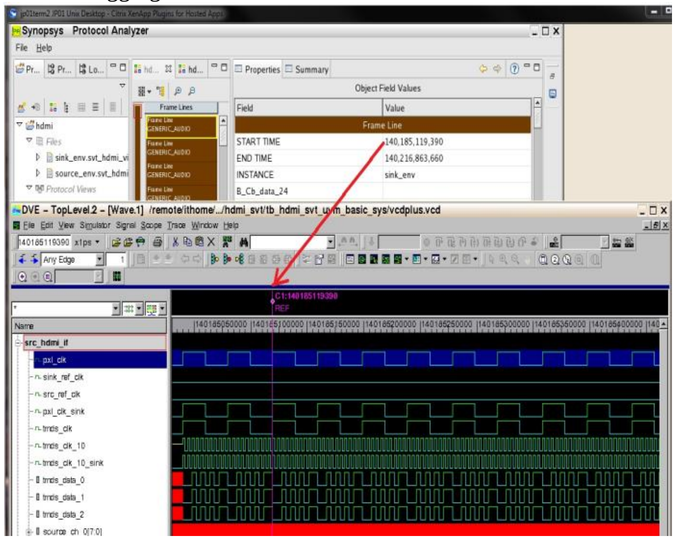


Figure 11 :- Hooking of the Protocol Analyser with the HDMI VIP

5.1 HDCP Authentication

The HDCP (High-bandwidth Digital Content Protection) authentication for the source model can be enabled by using a configuration attribute. The authentication in terms of reading and writing registers happen on the DDC (digital down-converter) or the i2c bus connected to the sink. Once the authentication is successful, the encrypted frames appear on the TMDS interface. As per protocol we also needed to apply a default pull-up on the sda and the sclk lines.

EVENT_HDCP_AU_DONE event is triggered by the source driver once the authentication is done. The default keys for the cipher process have been taken from the appendix A of the HDCP 1.4 spec [7].The user had the flexibility to use his own defined set of Key Selection Vectors (KSV), Key set and AN through a callback task [pre_hdcp_keyset] in the source driver and the monitor. There is a minimum requirement for a frame to have at least 508 pixels (which is equal to `SVT_HDMI_HDCP_KEEP_OUT_START) when run with HDCP enabled. This will lead to errors from the model if not met.

There is an array in source configuration "hdcp_seq", which the Source used to read/write from/to registers. Check the reasonable constraint shown in figure 12 below –

```
constraint reasonable_hdcp_seq {
   foreach(hdcp_seq[i])
   {
     if(sink_is_repeater)
     {
      (i == 0) -> hdcp_seq[i] == svt_hdmi_configuration::RD_BSTATUS;
      (i == 1) -> hdcp_seq[i] == svt_hdmi_configuration::RD_BCAPS;
      (i == 2) -> hdcp_seq[i] == svt_hdmi_configuration::WR_AINFO;
      (i == 3) -> hdcp_seq[i] == svt_hdmi_configuration::WR_AN;
      (i == 4) -> hdcp_seq[i] == svt_hdmi_configuration::WR_AKSV;
      (i == 5) -> hdcp_seq[i] == svt_hdmi_configuration::RD_BKSV;
      (i == 6) -> hdcp_seq[i] == svt_hdmi_configuration::RD_RI;
      (i == 7) -> hdcp_seq[i] == svt_hdmi_configuration::RD_KSV_FIFO;
      (i == 8) -> hdcp_seq[i] == svt_hdmi_configuration::RD_HASH;
     }
     else
     {
      (i == 0) -> hdcp_seq[i] == svt_hdmi_configuration::RD_BSTATUS;
      (i == 1) -> hdcp_seq[i] == svt_hdmi_configuration::RD_BCAPS;
      (i == 2) -> hdcp_seq[i] == svt_hdmi_configuration::WR_AINFO;
      (i == 3) -> hdcp_seq[i] == svt_hdmi_configuration::WR_AN;
      (i == 4) -> hdcp_seq[i] == svt_hdmi_configuration::WR_AKSV;
      (i == 5) -> hdcp_seq[i] == svt_hdmi_configuration::RD_BKSV;
      (i == 6) -> hdcp_seq[i] == svt_hdmi_configuration::RD_RI;
     }
   }
 }
```

Figure 12:- HDCP Authentication sequence by using SystemVerilog constraints.

It was a set of reads and writes to various registers in the sink thru the ddc interface. This was the first authentication process, once this goes thru without any errors, the source will start sending encrypted frames on the HDMI tmds interface from the subsequent frames.

The above read and write to the registers was observed on the DUT interface as shown in Figure 13 below.

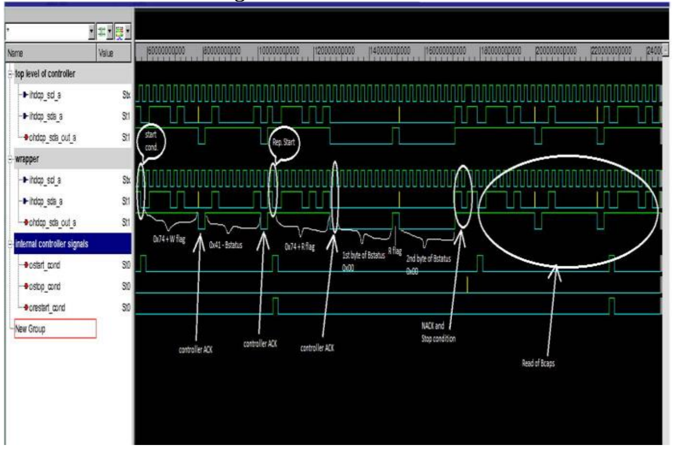


Figure 13: HDCP sequence as viewed on the DUT ddc interface

5.2 VESA DMT frame generation in DVI mode

The HDMI source model was configured to generate the VESA DMT format frames during DVI mode apart from the HDMI CEA frames. The configuration to switch the mode from HDMI to DVI came from the C-side via an API. This was then used to constrain the *op_mode* attribute of the source configuration to DVI_MODE and switch off the *reasonable_op_mode* constraint which sets the *op_mode* to be in HDMI_MODE by default. The *video_standard* attribute of the current frame video configuration was also obtained from the C-Side and set either to VESA or CEA as shown in Figure[14] below.

```
function new(string name="dvi_vesa_dmt_hdmi_system_configuration");
  .....
  this.src_cfg.reasonable_op_mode.constraint_mode(mode_from_c);
  .....
  this.src_cfg.video_cfg.video_standard = vdo_std_from_c
  .....
endfunction :new

constraint basic_system_cst{
  .....
  this.src_cfg.op_mode == mode_from_c;
  .....
}
```

Figure 14: Configuration of SV model from C

5.3 Non-standard frame generation

The concept discussed in Section 5.3 was extended to generate non-standard HDMI frames. The typical parameters present in the HDMI database object that define a frame are shown in Figure [14]. The values in the figure is for VIC=1 extended resolution format. . The *set_format_field* and *get_video_id* APIs of the HDMI database were overridden to generate a frame of user choice i.e the frame parameters were redefined using the *set_format_field* API as shown in Figure[15] and subsequently the *get_video_id* API needs to map the new values of the frame parameters to the video id code.

```
hfront              = 176;
hsync               = 88;
hback               = 296;
vfront_1            = 0;
vfront_2            = 0;
vsync               = 10;
vback_1             = 72;
vback_2             = 0;
vpol                = 1'b1;
hpol                = 1'b1;
hactive             = 3840;
vactive             = 2160;
vtotal              = 2250;
pxl_clk_type        = 29700;
vic_pxl_clk         = PIXEL_CLOCK_297;
min_pxl_repeat      = 0;
max_pxl_repeat      = 0;
limited_quant_range = 0;
intl_even_vblank    = 0;
```

Figure 15: Frame parameters

6. Guidelines for Generic VIP architecture to enable application layer.

Although, we created application layer on top of HDMI UVM VIP the philosophy 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 the VIP should enable UVM features such as configurations, factory overrides, callbacks as these enable a lot of re-use and scalable testbenches
2. VIP code must have a rich set of constraint defining the valid ranges of all the parameters inside the configuration or the transaction data classes. This helps in coverage convergence as well as faster debug, in case of invalid values being driven.
3. VIPs should provide a rich set of sequences to enable user run a many tests with very minimal effort
4. Most of the VIP components should be made configuration aware. This ensures that the VIP configuration can be made available to data objects as well. For eg – Once the sequencer is made configuration aware, the sequences running on the same can request the same via 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 test-gen was the is_valid check done by the VIP on the transaction and the configuration data object before using it. Primarily, these checks available inside the configuration and the transaction classes ensure that these objects do not assume in-valid 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 codes are automatically dumped. Whenever the VIP gets a value via uvm_config_db it provided a check to ensure that the get was correctly executed.
7. A default coverage model implemented as user extensions or callbacks to a monitor. This ensures ease of adding new cover-groups to the monitor by extending the coverage model adding new cover group and appending it to the correct model.

7. Results and Conclusion

A challenge in the verification of a modern SoC is in creating traffic patterns that are of relevance to the end application. It is the task of a SoC verification engineer to create these traffic patterns using Verification IPs. But the programming interface to these VIPs is complicated with complex protocols such as HDMI, PCI Express, USB etc. This challenge can be mitigated to some extent by adding an abstraction layer on top of the Verification IP.

Creating system level tests using a Verification IP can be a tricky problem to solve. Generally such users want a quick way to create a test and do not want to delve into the details of programming each attribute of the Verification IP whereas a typical UVM based Verification IP requires a bunch of configuration attributes to be programmed either through the configuration class of the VIP or the transaction object flowing through the VIP. This is integral to a VIP as they provide enough flexibility to test any scenario. In this paper we talk about a novel approach of having an application layer on top of any VIP to be able to translate between the system level tests and the VIP level configuration settings etc.

Thus, with minimal user involvement, we were able to create and control the desired application level scenarios, re-use them for validation or prototyping and thus concentrate on converging in completing the verification tasks efficiently.

Though the UVM based HDMI VIP was used to demonstrate this flow, the various approaches techniques and guidelines can be well leveraged with other VIPs and methodologies across various constrained random verification environments to increase the verification productivity.

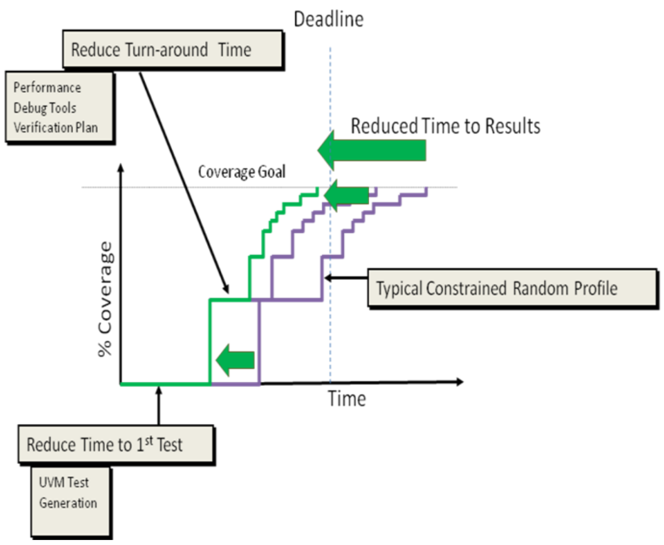


Figure 16 :- Coverage Vs Time for various test gen strategies.

The graph in Figure 16 shows the reduced turn-around time by using an application layer such as ours. This provides coverage convergence much ahead of the project deadline. This is achieved by significantly reducing the time to adapt the third party VIP and reducing the time spent to write VIP specific code to configure the VIP or write new sequences and tests.
Such an approach reduces the stimulus generation time using a third party VIP and the efforts on VIP specific code development. One can argue that an exhaustive set of tests supplied with the VIP can solve the same purpose, but with protocols like HDMI which has too many possibilities of audio and video types for example, it is an overkill to run all possible sequence of the protocol.

7. REFERENCES

[1] Accellera, *Universal Verification Methodology (UVM) 1.1 User's Guide*, 2011
[2] HDMI-1.4 Specification
[3] Synopsys HDMI UVM VIP User Guide
[4] CEA-861-F Specification
[5] UVM Reference Guide
[6] Mark Glasser, Mohamed Elmalaki, *Advanced Testbench Configuration with Resources*, DVCon 2011
[7] HDCP 1.4 Specification
[8] VESA DMT version 1 revision 12