C through UVM: Effectively using C based models with UVM based Verification IP

Chris Spear, Kevork Dikramanjian,
Abhisek Verma
Synopsys
spear@synopsys.com
kevork@synopsys.com,
abhiv@synopsys.com

Senay Haile
Qualcomm
shaile@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-1.4b specification, adheres to the latest verification methodologies such as Universal Verification Methodology (UVM) and was developed using SystemVerilog.

The challenge for this project was to integrate the legacy C models/testbench mainly used for HDMI stimulus generation with the HDMI VIP UVM code effectively. The HDMI VIP would then hook onto the design under test (DUT). With careful planning and execution we were successfully connected these three pieces. This ensured complete reuse of the C-based models and efficient usage of the HDMI VIP. The DUT is a video capture and processing subsystem used as a front-end to various video interfaces, including HDMI.

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.

The project was divided into the following milestones:

- Define the communication between the C testbench and the UVM based VIP
- Develop the SystemVerilog Direct Programming Interface (DPI-C) code to transfer data from C to SystemVerilog side of testbench
- Transfer data between the C and SystemVerilog parts of the testbench with DPI-C
- Establish the connectivity between the C testbench, the UVM based VIP (source) and the DUT (sink)
- Configure the HDMI VIP to run tests

The C testbench acts as a stimulus generator and creates the frame as per the HDMI protocol. The frame data is passed to the UVM based VIP, which drives them on signal interface as per the protocol.

The flow below outlines how we established handshaking to synchronize the flow between the C test and the SV testbench:

- The UVM VIP enters build phase, deliberately skips the randomization of the configuration class and then enters the connect and run phases where it polls the *test_done* flag, set by the C model.
- The C model initializes and then calls test library functions to set high-level knobs. DPI-C functions are used to set fields which eventually go into SystemVerilog constraint blocks within the configuration class. Constraints include video, audio and packet mode and traffic profile.
- The C model calls a SystemVerilog function which builds and randomizes the configuration class with the applied constraints
- The C model reads back low-level constraints which were solved by HDMI VIP and then configures the DUT with same constraints.
- C model starts HDMI traffic sequence in the VIP. In the SystemVerilog testbench or the UVM VIP, the sequence generates N transactions and sends it to driver and then to the DUT

Categories and Subject Descriptors

HDMI, UVM, Re-use philosophy

General Terms

Verification, Design

Keywords

HDMI, VIP, UVM, CEA-861-D, 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].

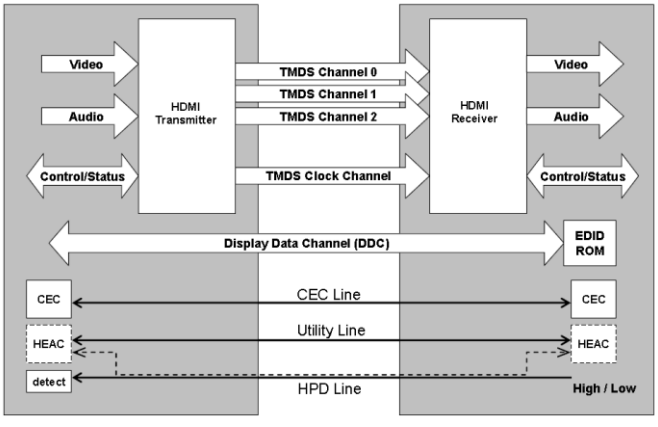

Figure 1 :- HDMI source(Tx) and sink(Rx) block Diagram

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.

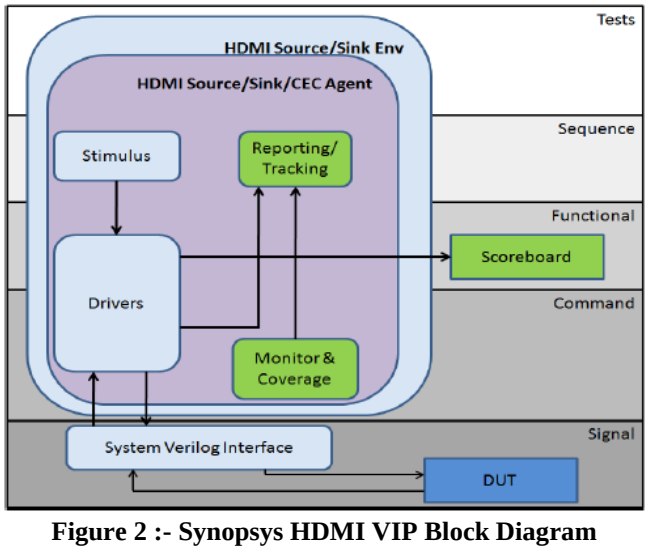

Figure 2 :- Synopsys HDMI VIP Block Diagram

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 below. The components shown in purple color are Synopsys UVM compliant VIPs. The components shown in orange are the Qualcomm legacy C testbench while the DUT is the light blue box.

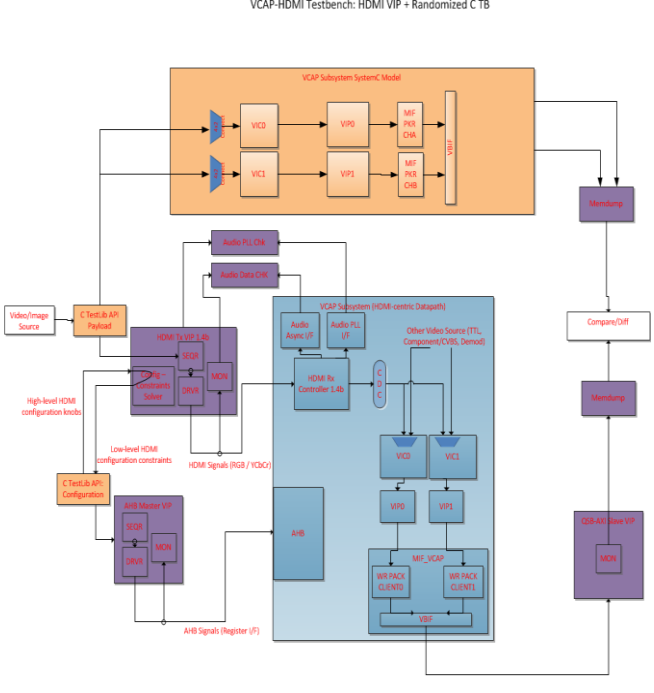

Figure 3 :- Synopsys HDMI VIP with Qualcomm C based testbench.

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 Test flow

The HDMI is a SystemVerilog UVM-compliant VIP which contains the built-in standard phases for test flow. These include the *build_phase(uvm_phase phase)*, *connect_phase(uvm_phase phase)* and *run_phase(uvm_phase phase)* phases which are invoked from the UVM scheduler when the *run_test()* method is called in the testbench.
The C/C++ testbench follows its own test flow that is independent from the UVM test flow. Since the C/C++ RTL testbench invokes the initialize/reset functions, DUT/VIP configuration functions and checking functions it was decided that the C/C++ RTL testbench would dictate the test flow for the tests instead of the HDMI UVM-compliant VIP. In order to allow the C/C++ RTL testbench to control the test flow and to prevent the UVM test flows from colliding with the C/C++ test flows, the HDMI VIP was effectively made a slave to the C/C++ testbench. This was done by adding a semi-infinite loop to the *run_phase(uvm_phase phase)* method of the **uvm_test** class such that the HDMI VIP would never exit the loop until invoked by the C/C++ RTL testbench. The HDMI VIP would only be active when it accepts commands from C/C++ test library via DPI-C export functions for the purpose of configuration and stimulus generation. That way the HDMI VIP can gracefully go through all the UVM phases at the end of the test as dictated by the UVM scheduler.

4.2 Configuration

The HDMI VIP contains a configuration class which has several fields used to configure the type of video, audio and packet data it can generate. The fields are fully constrained-random and are used in several SystemVerilog constraint blocks that are included as part of the HDMI VIP package. This contrasts with the more directed nature of the C/C++ testbench, since C/C++ does not lend itself easily to constrained-random testing.
A mechanism needed to be developed where the high-level video mode could be controlled from the C/C++ test library and passed into the HDMI VIP, effectively bypassing some of the constraint blocks for relevant configuration fields. Only the fields required for high-level video mode would be set from C/C++, whereas other fields not required by the reference testbench would be randomized within the HDMI VIP. The fields required to be controlled from the C/C++ test library includes video frame height, frame width, scan type, timing information (blanking/synchronization), pixel encoding type and color depth. These fields are passed in from C/C++ test library because they are used by the C/C++ reference testbench to configure the SystemC reference models and generate the golden memory dumps.
DPI-C was chosen as the method of communication between C/C++ test library and HDMI VIP, since SystemVerilog has built-in support for DPI-C with little overhead. Configuration information that is determined from the C/C++ test library is passed into the HDMI VIP via DPI-C export functions as shown in Figure ?. They get passed into the configuration class and ultimately end up as inputs to custom constraint blocks of the configuration class. Once all the configuration is passed in, the C/C++ test library invokes the

randomize() method of the configuration class via DPI-C to solve the applied constraints.
Configuration fields in the HDMI VIP which are not constrained from C/C++ test library are randomized and returned to the C/C++ test library for DUT configuration, such as audio-related configuration and control-related information. This way the C/C++ RTL testbench can take full advantage of the randomization features of SystemVerilog while maintaining control over basic configuration required for the C/C++ reference testbench.

Abhisek, insert code here:

-

4.3 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 4 :- The C Side

Figure 4 and 5 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 5 :- 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 [6]. 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 6 :- 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.

4.3 Audio data Scoreboard

Unlike video data, audio data was entirely generated and checked within the context of the HDMI Source VIP. An external audio scoreboard was used for audio data checking. A callback function within the UVM based HDMI Source monitor was used to extract reference audio samples and queue them up in the scoreboard for future comparison. There was little reliance of the C side since audio data passes through the DUT relatively untouched. A monitor was used in the audio output interface of the DUT for protocol checking, and upon reception of an audio packet the monitor sends the packet to the scoreboard for comparison.

```
function void post_hdmi_di_pkt_trans ( svt_hdmi_monitor hdmi_monitor,
                                       bit [3:0][7:0] di_pkt_header,
                                       bit [31:0][7:0] di_pkt );

  // Process audio ASP/HBR/DST etc and use it for scb.
  case (di_pkt_header[0])

      svt_hdmi_frame_line::ASP:
        begin
        end
      …..
      …..
  endcase
endfunction
```

Figure 7 :- Audio scoreboard callback in source monitor

4.4 Video data Scoreboard

The video data scoreboard relies on the C reference testbench for reference video frames. The C/C++ reference testbench contains SystemC models which emulate the video processing functions found in the DUT. Once a SystemC simulation completes the memory output file is used to validate the C RTL simulation results. In this way the video data scoreboard acts as a post-process checker for video data.

4.5 Control data Scoreboard

Similar to audio data, control/Info-frame data was entirely generated and checked within the context of the HDMI Source VIP. Since the packet data remains untouched within the DUT, packet data scoreboard and callback functions were used to extract reference control packet data from the HDMI stream for comparison with DUT output packet data.

5. Feature Verification

Hooking up of the Protocol Analyzer helped a lot in debugging purposes as shown in Figure [8]. It provided a GUI based view of the transaction and its synchronization with the waveforms helped a lot in debugging at the transaction level.

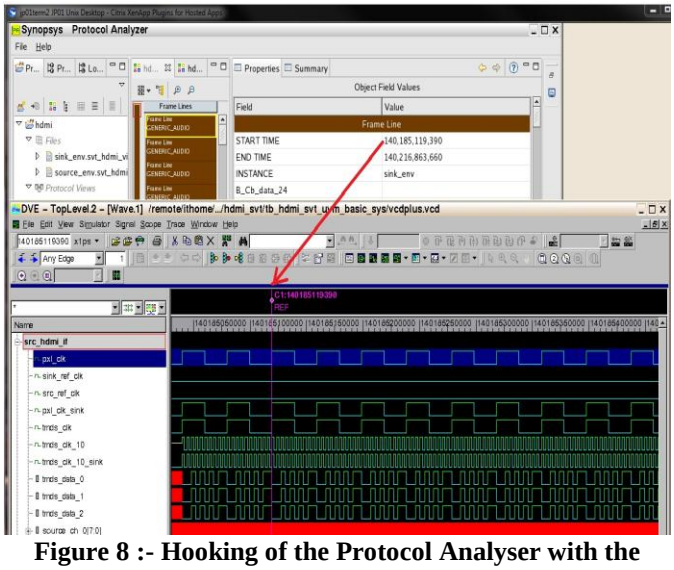

Figure 8 :- Hooking of the Protocol Analyser with the HDMI VIP

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 9 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 9:- 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 10 below.

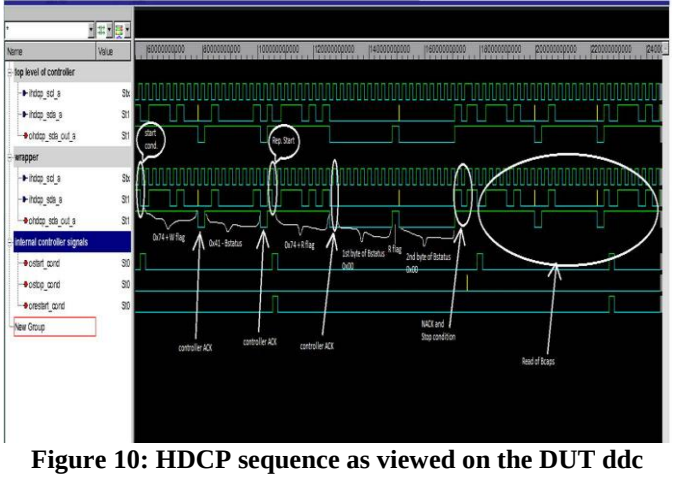

Figure 10: HDCP sequence as viewed on the DUT ddc interface

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[11] 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 11: Configuration of SV model from C

5.3 Enabling faster simulation

The HDMI VIP provides for a mechanism to reduce the number of video active lines (denoted as *vactive*) in a given HDMI frame leading to faster simulation. The number of pixels in each *vactive* line (denoted by *hactive*) can also be reduced to further shorten the frame size and increase simulation speed.

For example, according to the CEA-861D specification, the total number of lines to construct a frame in 2D format with video id code (VIC) 1 is 525, out of which 480 lines are *vactive* lines. Each *vactive* line contains 640 active pixels (or *hactive*). Hence the frame is of size 640x480. The same frame when shortened to 16x12 improves simulation performance drastically, while still keeping the control lines same as the CEA-861E spec.

It was achieved by using the UVM factory override of the HDMI database object which contains the information as available in Table 2 and 3 of CEA-861E spec. The HDMI database object of the VIP provides *set_format_field* and *get_video_id* APIs to be overridden by the user to supply the attributes such as vactive/hactive of a frame
As shown in figure [12] below.

```
class cust_svt_hdmi_hdcp_database extends svt_hdmi_hdcp_database;


 virtual function void set_format_fields(int video_format_id_code,
            svt_hdmi_video_frame_configuration::video_format_type_enum format_type =
            svt_hdmi_video_frame_configuration::FORMAT_2D);

   case(video_format_id_code)
    1        :  begin
                  vactive = 16;
                  hactive = 12;
                  vtotal   = 61;
                end
    default  :  super.set_format_fields(video_format_id_code,format_type);
   endcase
 endfunction

 virtual function int get_video_id( bit [7:0] scan_frequency, bit [12:0] active_pxls,
                                    bit [11:0] hblank, bit [11:0] active_lines,
                                    bit interlaced,  bit [2:0] aspect_ratio = 0,
              svt_hdmi_video_frame_configuration::video_format_type_enum
              format_type = svt_hdmi_video_frame_configuration::FORMAT_2D);

  case({scan_frequency, active_pxls, hblank, active_lines,
      interlaced, aspect_ratio})

    {8'd60, 13'd12, 12'd160, 12'd16, 1'b0, 3'b000}:
     begin
      get_video_id = 1;
     end

    default : super.get_video_id(scan_frequency, active_pxls, hblank, active_lines,
                                 interlaced, aspect_ratio, format_type);
  endcase
 endfunction

endclass : cust_svt_hdmi_hdcp_database
```

Figure 12: set_format_fields and get_video_id APIs to enable faster simulation

The extended HDMI database class can be then used to override the blueprint of the HDMI database with the model by using the UVM factory override function as shown in Figure [12] below.

```
virtual function void build_phase(uvm_phase phase);

  super.build_phase(phase);

   /** override the hdcp database class from the test class*/
  set_type_override_by_type(svt_hdmi_hdcp_database::get_type(),
                            cust_svt_hdmi_hdcp_database::get_type());

endfunction: build_phase
```

Figure 13: Overidding the HDMI database thru test class

5.4 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[12] 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 14: Frame parameters

6. Results and Conclusion

This challenging project was a good example of how to integrate a legacy testbench with new technology. By integrating our C/SystemC testbench with the HDMI VIP which was developed using UVM/SV we have reused our testbench code and have benefitted from using state of the art verification IP which complies with the HDMI standards, compared to developing the VIP ourselves. The simulation performance is measured as a tradeoff between the relaxed System Verilog constraint solver efforts and overhead for DPI-C calls.

Thus, with minimal user involvement, we were able to create and control the desired testcases 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.

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-D 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