2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

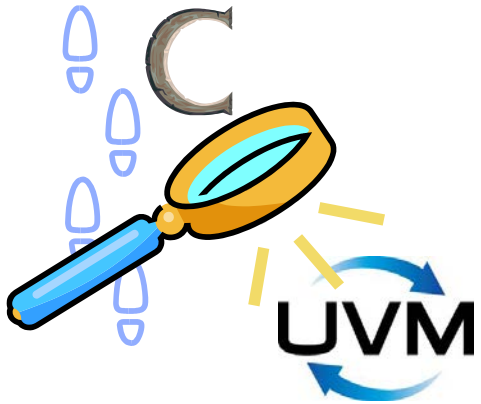
Application Abstraction Layer: The Carpool Lane on the SoC Verification Freeway

Abhisek Verma
Varun S
SYNOPSYS®
Accelerating Innovation

Subramanian Kuppusamy
QUALCOMM®

Agenda:

- VIP usage Challenges
- Application layer to deal with it
- CASE STUDY : The Testbench
- CASE STUDY : The Test Flow
- CASE STUDY : Stimulus Generation
- Interrupt-based C++ interaction
- Re-use @ Silicon Validation
- VIP coding guidelines
- Take Away!



Challenges With Verification IP Integration

Analyze Coverage & Results

Run with different seeds

Debug Protocol Activity

Develop Test Plan

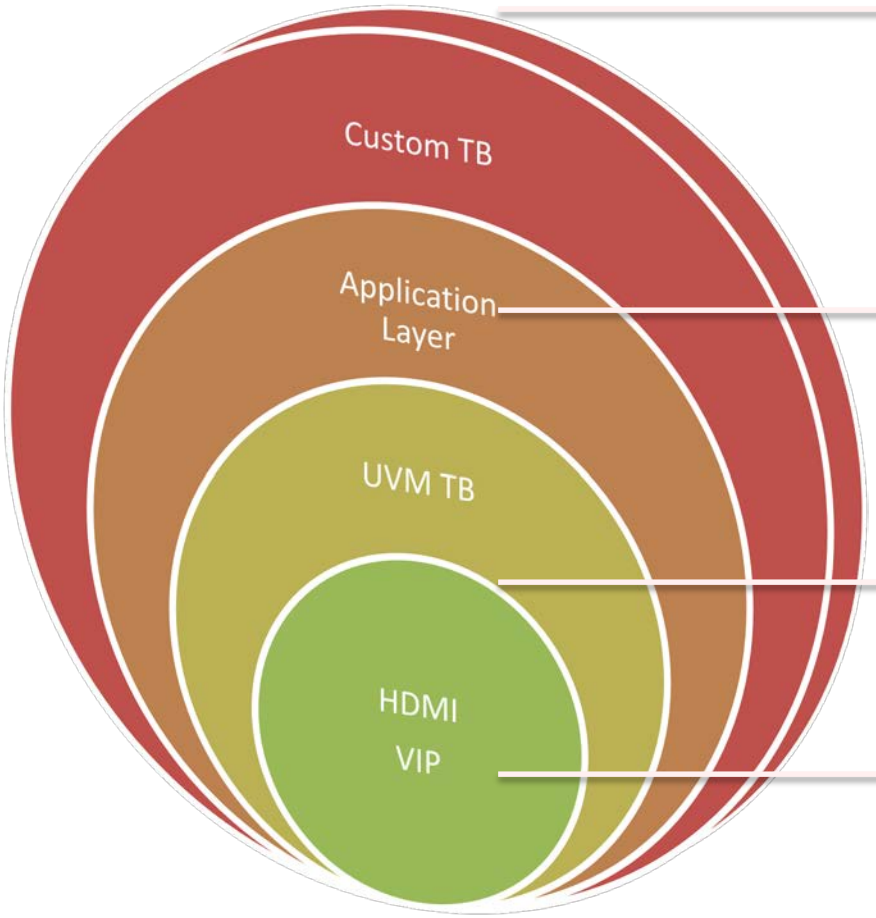
Configure VIP

Multiple iterations of test creation till coverage closure.

Writing methodology specific test sequences.

Mapping the umpteen configuration variables with hardware specifications.

How to deal with them our way?



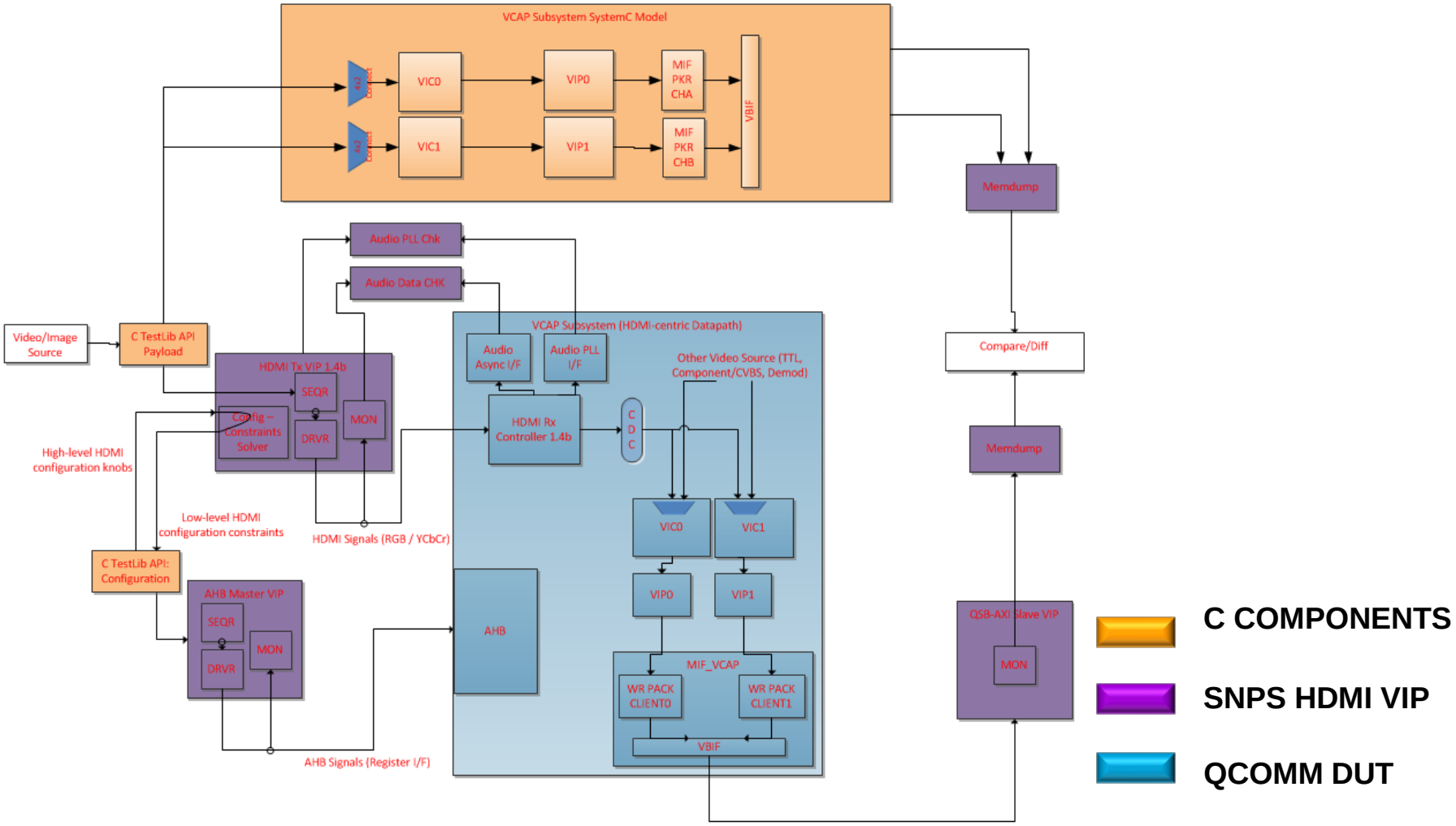
Highly configurable and scalable custom testbench component created as a result of application layer. C/C++ for re-use at firmware.

Provide the user with an application layer to create tests, application specific sequences and VIP configuration classes as extensions of the base HDMI VIP classes through a utility. Can be C/C++ for re-use at firmware.

Compliance to UVM enables a highly configurable testbench template and ease of integration of the VIP

Industry wide rapid adoption of 3rd party VIP based on widely used and emerging protocols, to accelerate the development of a complete verification env.

The Testbench



The Test Flow

- The C application layer 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 application layer APIs are generic to either hook to simulation VIP or synthesizable transactor or final firmware.

The Test Flow (2)

- The C application layer calls a SystemVerilog function which builds and randomizes the configuration class with the applied constraints
- The C application layer reads back low-level constraints which were solved by HDMI VIP and then configures the DUT with same constraints.
- C application layer starts HDMI traffic sequence in the VIP. In the UVM VIP, the sequence generates N transactions and sends it to driver and then to the DUT

Stimulus Generation

C-side

```
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);
    }
  }
}
```

```
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
```

SV/UVM-side

Stimulus Generation (2)

- The video frame stimuli needed to be generated from C test library and sent to the HDMI VIP via DPI-C tasks.

- The DPI-C export task *sv_hdmi_send_frame_line()* is invoked from 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 test library and pack it to the HDMI VIP sequence item for transmission.

Interrupt-based C++ interaction

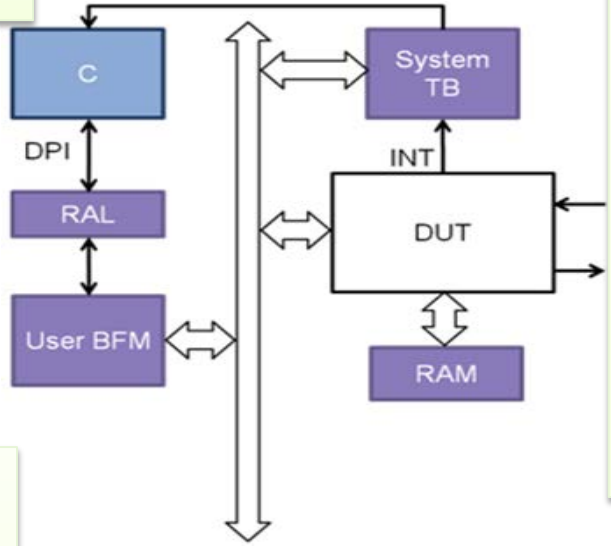
C++ test entry taking context as an input

```
extern "C" int
usb_dev_isr_entry(int context)
{
   usbdev_t usb(context);
   return usb_dev_isr(usb);
}
```

Device driver accepting reference of the reg model

```
reqs=snps_reg::regRead(usbdev.status());

snps_reg::regWrite(usbdev.intMask(),0xFFFF);
```




```
import "DPI-C" context task dev_drv(int ctxt);

class cpp_test extends uvm_test;
   int context_val;
   `uvm_component_utils(cpp_test)

   virtual function void connect_phase(…);
      super.connect_phase(phase);
      context_val =
         snps_reg::create_context(env.model);
   endfunction: connect_phase

   virtual task run_phase(uvm_phase phase);
      super.run_phase(phase);
      phase.raise_objection(this);

      // C++ device driver called via DPI-C
      dev_drv(context_val);

      phase.drop_objection(this);
   endtask: run_phase

endclass: cpp_test
```

The C++ Device driver being used in simulation test

```
void slave_driver::dev_drv(slave_t dev)
{
    uint32 mode_status;
    regWrite(dev.SESSION.SRC(), 0x0000FA);
    regWrite(dev.SESSION.DST(), 0x000E90);
    regRead(dev.MODE_STATUS);
  switch ( mode_status )
  {
   case 0x0001:   regWrite(dev.IDX(), 0x5aa5);
                  break;
   case 0x0080:   regWrite(dev.IDX(), 0xa55a);
                  break;
   default:       regWrite(dev.IDX(), 0x0000);

   }
};
```

C++ device driver code

```
static slave_t Sys("Sys", 0);

int
main(int argc, char* argv[])
{
    return slave_driver::dev_drv(Sys);
}
```

Device driver scheduled for execution as software

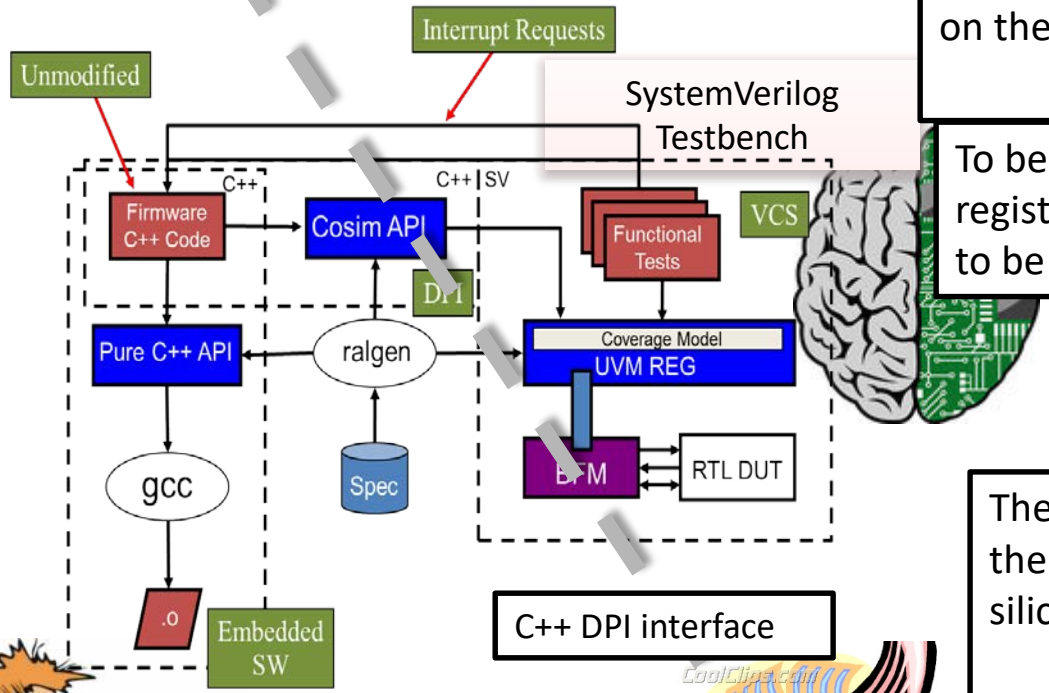
Environment for an interrupt-driven C++ interaction

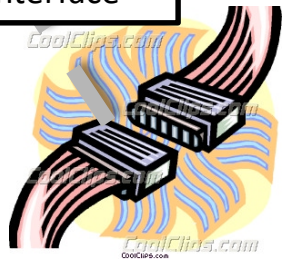
Re-use @ Si Validation

To be compiled and executed as a standalone C++ code on the target processor

To be interfaced to the SystemVerilog register model using DPI-C to be simulated on a HDL simulator

The need of the hour is to ensure that the sequences can be reused in post-silicon validation from RTL simulation.





Guidelines for VIP's

Adhere to the UVM guidelines and best practices. Use UVM features such as configurations, factory overrides & callbacks.

Provide constraints within a VIP defining valid ranges for all configuration and data parameters.

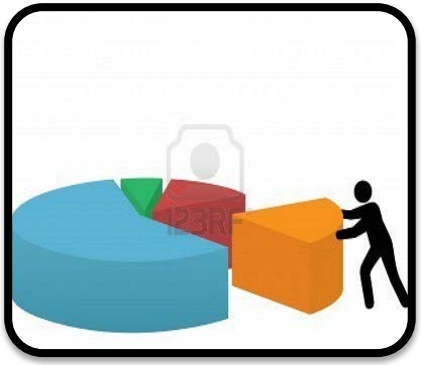
UVM-VIPs should provide a rich set of sequences to enable user run many tests with very minimal effort.

Make the VIP components configuration aware.

Guidelines for VIP's (2)

Provide validity checks for data & configuration within the Verification IP.

Use strongly typed set() and get() while using uvm_config_db.



Provide a default coverage model with a possibility for user extensions.

Take Away!

- Advanced methodology provides appropriate hooks for VIP adoption
  - Can be leveraged to create simple application layer for a Verification IP user.
  - VIP users can focus on verification requirements.
- The simulation performance was measured as a tradeoff between the relaxed System Verilog constraint solver efforts and overhead for DPI-C calls.
- Though the UVM based HDMI VIP was used to demonstrate this flow, the approach can be well leveraged with other VIPs and methodologies across various constrained random verification environments to increase the verification productivity

Thanks for your time!!