February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

One Compile to Rule them All: An elegant solution for OVM/UVM Testbench Topologies

by
Galen Blake
SMTS

ALTERA®

and
Steve Chappell
Solutions Architect

Mentor Graphics®

PLEASE NOTE:

- This is a ROUGH DRAFT.
- I will be providing daily updates refining and adding content.
- This ROUGH DRAFT is provided to comply with the deadline (24th in the email and 25th on the website).
- More work is planned to bring the contents and particularly the diagrams up to a more substantial level.
- Having said all that, the basic outline and information has been captured.

SOC Testbench Topology

- SOC Designs present many Verification challenges.
  - Many peripherals
  - Many peripheral configurations
  - Many peripheral connectivity options
- System Verilog and the OVM/UVM libraries offer some clever solutions.
  - Interfaces, modports, polymorphism etc.
- However it is not always easy to configure the topologies needed especially across the HDL and HVL contexts.

SOC Testbench Topology

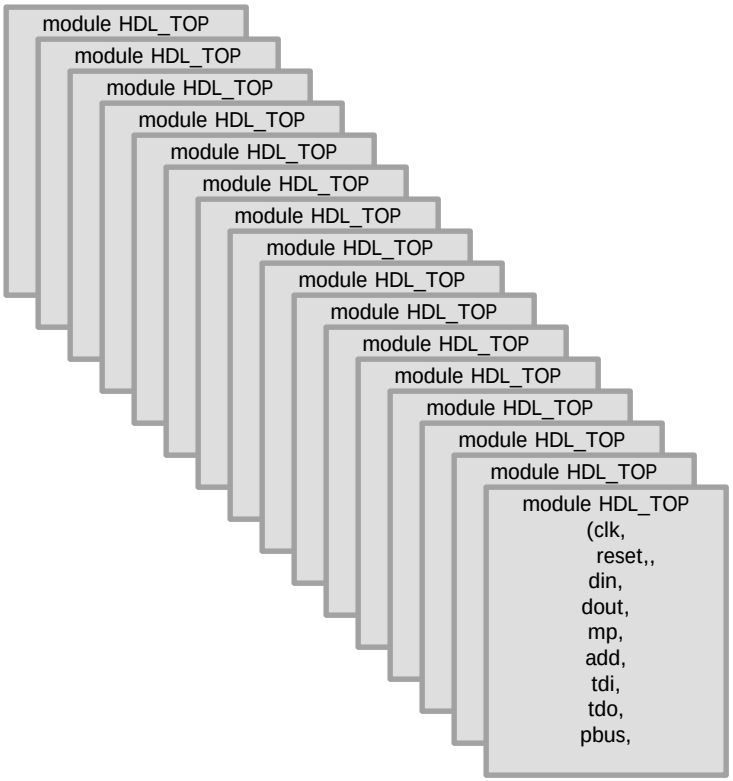
- Solutions to topology and configuration are frequently developed in an ad-hoc fashion as new test bench and DUT features are needed.
- They are rarely implemented with a consistent approach.
- This leads to clumsy solutions become difficult to enhance or maintain including:
  - A new net-list for each configuration and option
  - Pervasive reliance of the preprocessor (text macros)
  - Extensive use of parameters
  - Configuration Registers
  - Hybrid solutions mixing all of the above.

Testbench Topology Explosion

- Consider the following hypothetical SOC design
  - A central multi-core CPU
  - Ten to twenty peripherals available
  - Each peripheral has multiple modes (ie ddr2 and ddr3)
  - Each peripheral has multiple connectivity options.
- Given these characteristics the maximum theoretical number of topologies could easily reach into the millions.
- Many of the possible topologies are not useful combinations but even a small fraction of a million combinations is still a really large number.

A new net-list for Every Topology?

- There would be way to many.
- They will be really hard to maintain.
- For example one set of port updates would need to be replicated across each of these net-lists leading to a lot of error prone duplicated efforts.



Connecting VIP's to DUT Pins

- There may be many possibilities to route peripheral interfaces through an SOC to different sets of pins.
- Therefore multiple paths of connectivity from the VIP to the proper DUT pins must be supported.
- Solving this problem increases the size and complexity of the testbench.

Relying on the Preprocessor

- When used extensively, the number of text macros could easily explode making them hard to enhance or maintain.
- When text macros are used across different design teams or third party IP, name collisions occur frequently and can be difficult to correct.
- When the preprocessor is used to define topology, it becomes frozen at compile or analysis time.
- There is no flexibility to defer choices until elaboration time.
- There must be a compile or analysis and a temporary database executed and created for each topology.

Relying on parameters

- Parameters work well for configuring topology for HDL (modules).
- However, parameters do not work as well for HVL (parameterized classes).
- Since they work well on the HDL side and not as well on the HVL side, how could they be used across both contexts?

Configuration Registers

- Provides for dynamic reconfiguration.
- Registers must be programmed and readable.
- This may require adding wiring to the net-list or API's to the registers.
- The bigger problem is that as new features are added that impact testbench topology, new register bits must be allocated and added and API access (from the HVL class side) must also be enhanced.

Hybrid Solutions

- In most real world cases, some combination of these solutions are implemented.
- This may be due to legacy code, multiple IP providers or teams or just different levels of experience on a design team.
- Hybrid solutions are usually difficult to maintain and enhance since there are so many different ways to do the same types of things.

Topology in HDL & HVL contexts.

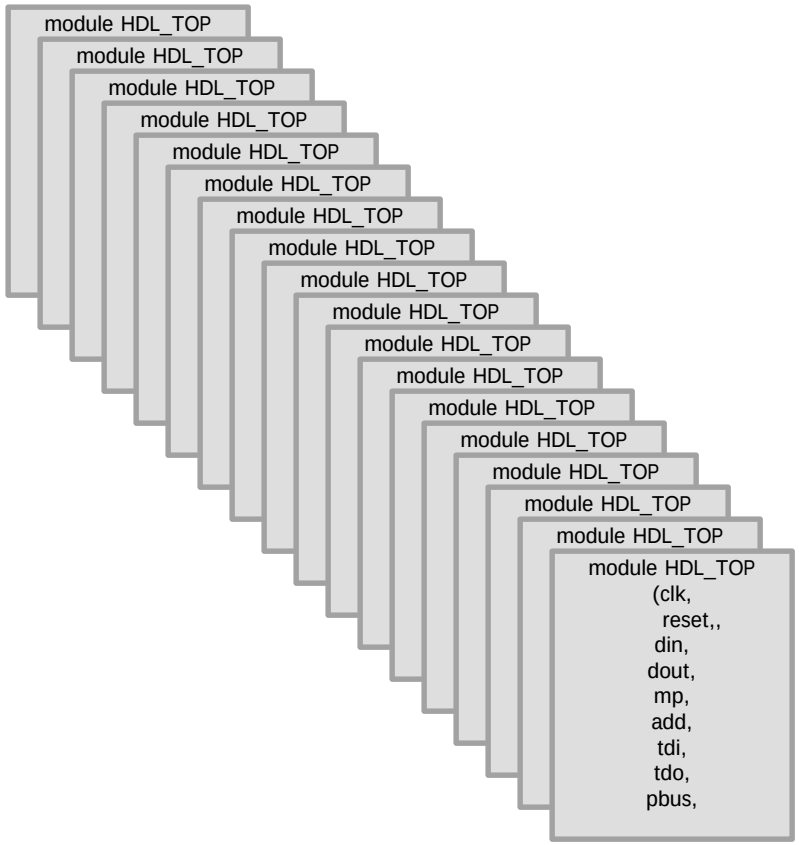
- Any solution for dealing with the configuring the topology of the testbench must consider both the HDL (module) and HVL (class) contexts.
  - If separate HDL net-lists are used, they must have equivalent HVL objects.
  - If separate text macros
  - Parameters
  - Configuration registers
  - Hybrid solutions

Requirements for an Elegant Solution.

- One and only one HDL net-list and HVL object.
- Consistent and clean method to configure topology.
- Encapsulate the complexity of testbench connectivity leaving simple connections for the net-list and object.
- A single access and control point to configure HDL and HVL.
- Only one compile or analysis is needed with all possible models are compiled.
- Topology choices are deferred until elaboration time.
- HDL net-list options are synchronized with appropriate HVL objects.

One netlist it all that is needed.

- Creating a net-list for every topology is clearly not a pratical solution and is very hard to maintain.
- Some keep the number of net-lists to a minimum.
- But even having only two would still invite difficulties keeping them maintained.

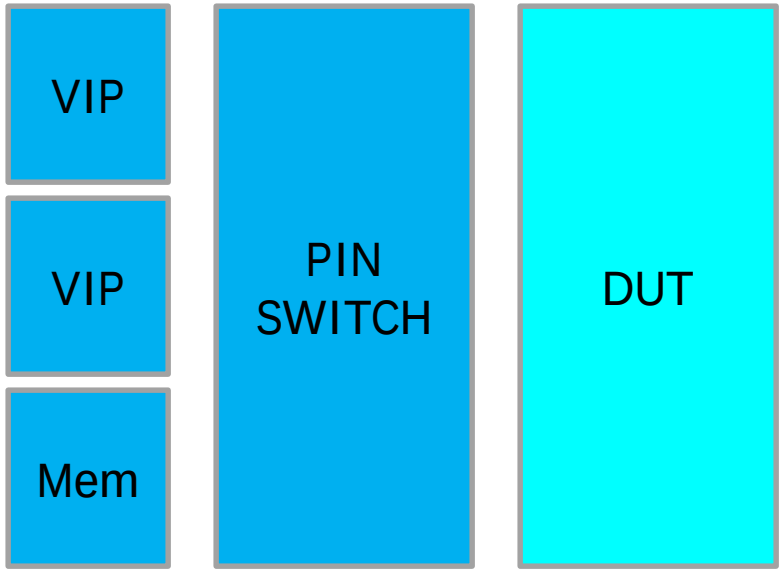


Topology Configuration.

- One set of topology parameters should be all that is necessary.
- The preprocessor should not be used for topology:
  - `define
  - `ifdef
  - `elsif
  - `endif

Encapsulate Complexity.

- Keep the top level net list very clean and simple.
- Define and maintain one set of simple connections for each DUT port connection, VIP and other testbench components.
- Switching connections from VIPs to DUT pins or internal nets may be complicated.
- Consolidate and reuse switching solutions.
- Avoid implementing different sprawling solutions which leads to creeping complexity and is difficult to maintain.



Control HDL and HVL topology.

- Parameters work well in HDL but not as well in HVL.
- Parameters used with **`generate`** is very effective in HDL.
- A generate statement could connect one of two UVM driver/monitor choices in the topology.
- But how would a corresponding agent and sequencer get selected in the HVL object hierarchy?
- How could this be done without requiring parameters in the class which is problematic?
- The answer is to pass HDL topology parameters to HVL by assigning them to signals in an interface.
- Access them from a virtual interface in the HVL object.

Only One Compile is needed.

- The flexibility of the generate statement is highly leveraged.
- It allows all possible models to be compiled at compile or analysis time.
- The decision as to which model to use for a particular test is deferred until elaboration time.
- This prevents recompiling for every possible topology saving many wasted hours for a regression run.
- The parameters that define which of the generate statements to follow can be passed from parameter settings from files or even on the simulator command line.

Topology Defined at Elaboration.

- The choices of which testbench VIPs or models will be used are determined by the topology parameters.
- These parameters can be set from default values or passed on the simulator command line that performs elaboration.
- Deferring topology choices until elaboration time gives maximum flexibility.

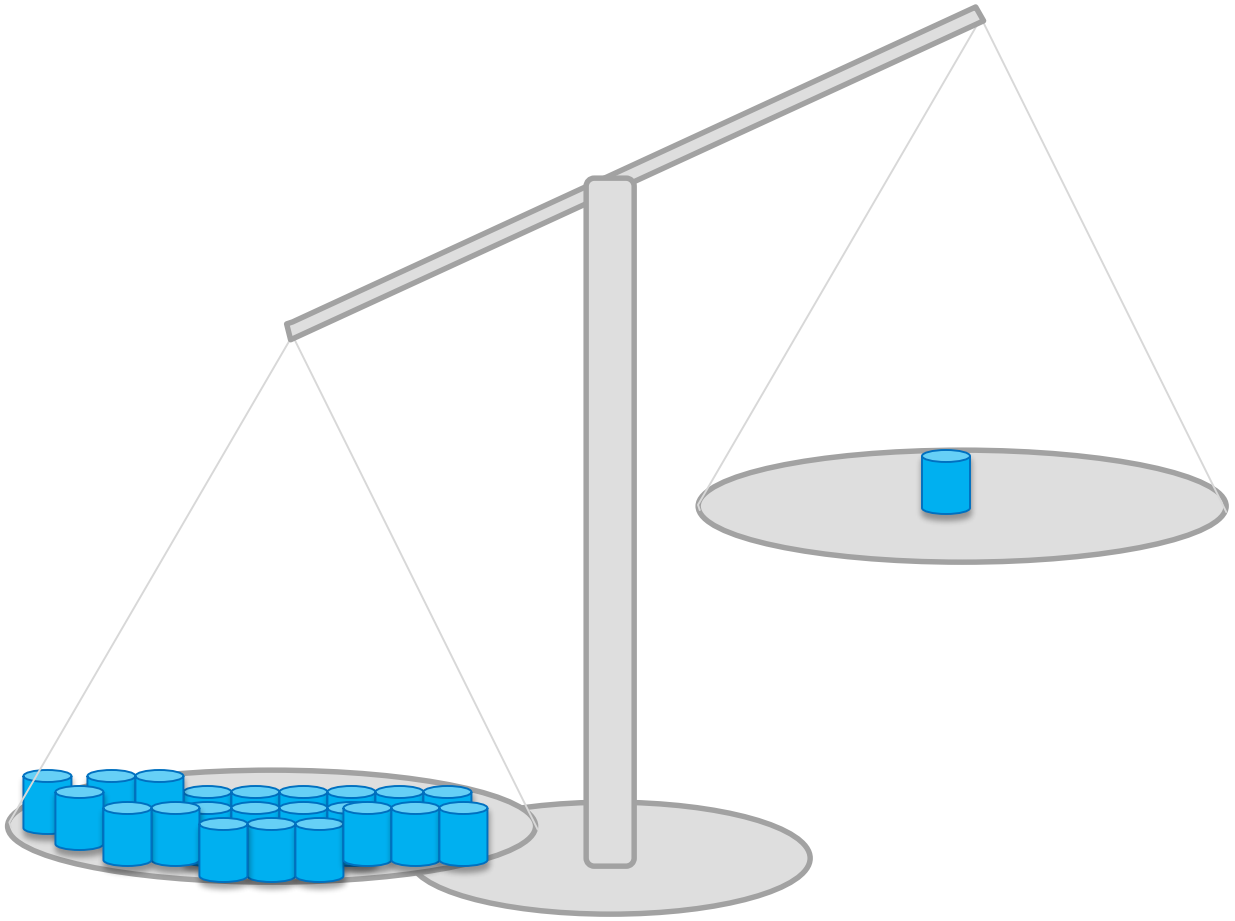
Common Control for HDL & HVL

- The parameters that control the generate choices are also assigned to signals in an interface and accessed through a virtual interface in the HVL objects.
- The values detected in the virtual interface in the HVL object are used to instantiate the correct agents in the HVL object matching up with the corresponding drivers or monitors chosen in the HDL generate statements.

More Advantages

- Suppose a large SOC has a total of 72 useful topologies.
- Traditional approaches using multiple net-lists or text macros would require 72 compile operations.
- If each operation took 10 minutes that would require 720 minutes or 12 hours of CPU time for a full regression run just to prepare to run simulation.
- If each compiled DB required 7Giga-bytes of storage then it would take 500 Giga-Bytes of storage just to prepare for simulation.
- In contrast, if all 72 topologies were controlled with the elegant solution there would only be one compile taking 10 minutes and 1 7 Giga-byte of storage.

Faster Compile time; Smaller



HDL Example

FROM THE HDL CONTEXT:

```
. . .
  parameter USE_L3_ACP_AXI_VIP                  = 0;
. . .
  // This interface is used to store and distribute parameter settings
  // for which TB components and parameters to use.
  config_tb_if cfg_tb();
  assign cfg_tb.use_l3_acp_axi_vip              = USE_L3_ACP_AXI_VIP;
. . .
  //// AXI Tranactor (replaces peripheral connections for testing).
  generate if (USE_L3_ACP_AXI_VIP) begin : inst_acp_axi_vip_mst
    axi_monitor_module #( .ADDR_WIDTH             (`ACP_ADDR_WIDTH),
                          .RDATA_WIDTH            (`ACP_RDATA_WIDTH),
                          .WDATA_WIDTH            (`ACP_WDATA_WIDTH),
                          .ID_WIDTH               (`ACP_ID_WIDTH),
                          .LEN_WIDTH              (4),
                          .AXI_ID                 ( "acp_axi_monitor" )
                        )  acp_axi_monitor (.mif(acp_axi_vip_mst));

    end
  endgenerate
```

HVL Example

FROM THE HVL CONTEXT:

```
. . .
virtual config_tb_if cfg_tb;

. . .
    if ( cfg_tb.use_l3_acp_axi_vip) begin
      ovm_report_info("build: ", "ACP_AXI_VIP agent will be enabled ...");
      l3_acp_axi_a = axi_agent #(ACP_axi_p)::type_id::create("l3_acp_axi_a",
this);
      l3_acp_axi_a.set_config_id("master_acp", axi_pkg::MASTER_MONITOR);
      l3_acp_axi_a.register_agent(TBMB_PERIF_ACP_ID);
    end
. . .
```

Conclusion

- Sprawling complexity and inconsistent hybrid solutions for defining the testbench topology create serious maintenance issues making it difficult to develop or enhance tests for complex SOC designs.
- Consolidating topology definition of both HDL net-lists and HVL object hierarchies to a single point source simplifies maintenance while preserving significant flexibility with a very large variety of topologies available.
- Encapsulating complex connectivity choices within a specialized switching model provides the necessary connectivity options while keeping the top level HDL net-list simple to define and maintain.