Don’t delay catching bugs: Using UVM based architecture to model external board delays

Amit Paunikar, Saurabh Arya, Vikas Makhija, Shaily Khare
Synopsys India pvt. Ltd.
apaunik@synopsys.com, saurabha@synopsys.com, vikasm@synopsys.com, shaily@synopsys.com

***Abstract*- A conventional SOC has different types of electrical delays. The cumulative result of electrical delays can cause a linear shift in each signal, which can be combined to model as board delays. This paper focuses on modeling external board delays, using the standard UVM (Universal Verification Methodology) multi-channel environment. It covers multiple approaches like fixed external board delay, configurable external board delay and randomized external board delay modeling. These approaches can help in training the device controller and modeling pre-silicon delays to catch issues at early stages of design and verification. This paper uses Low Power DRAM(LPDDR) and High Bandwidth Memory(HBM) to demonstrate modeling of external board delays.**

I. Introduction

A typical electronic system has different types of electrical delays, such as source delay, network delay, insertion delay, propagation delay, transition delays (rise time, fall time), path delay, external (wire) delays, phase delay, delays due to jitter, delays due to duty cycle distortion. It is always helpful if the impact of these delays can be resolved at the initial phase of the design. The cumulative result of these delays cause a linear shift in each signal, which can be combined to model as board delays. A typically verification environment, which is used in pre-silicon environment uses pin to pin connection via an interface and works on ideal assumption of zero transport delay between the system and external interface (Fig. 1).

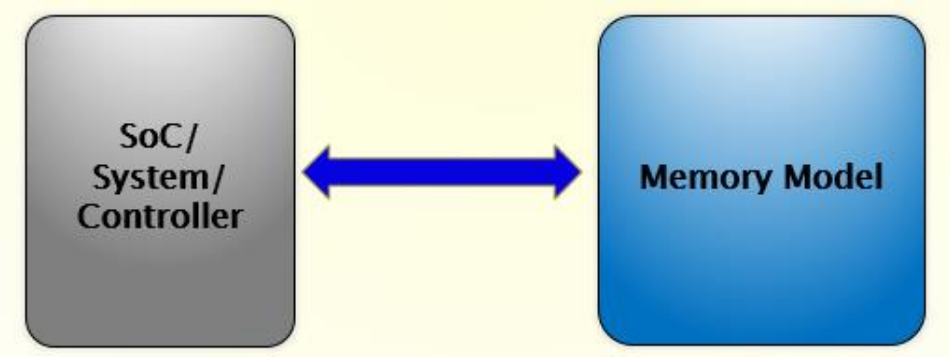

Figure 1. A Memory System

The above figure demonstrates a typical memory based system where the external interface is a DDR (Double Data Rate) memory. In a typical memory system with high speed DDR where the clock frequency can vary up to 2133 MHz with a data rate support of 4267 MBPS, the interface signals do have delays w.r.t each other as well as there is a common path delay in the board. These external board delays along with jitter (short-term variations of signals from reference clock) can cause inaccurate sampling of transferred information. In the SOC design, the impact of the delays is usually realized during gate level simulation or in static timing analysis, which usually occurs at the end of the design cycle. Even during the gate level simulation, a typical SOC system will have the delayed model but the external interface, which is usually the Verification IP and hence do not carry the board delay information.

With such high-speed interfaces, the verification of a typical SOC require a framework to induce configurable delay in I/O signals. In this paper, we are proposing one such model which we integrated with a Verification IP(VIP) (Fig. 2). In this paper, we have constrained our integration model w.r.t Memory VIP (also referred in this paper as Memory model). We have taken couple of approaches with LPDDR4 and HBM VIP. We have also demonstrated model with taking single as well as multiple instances for the Verification IP.

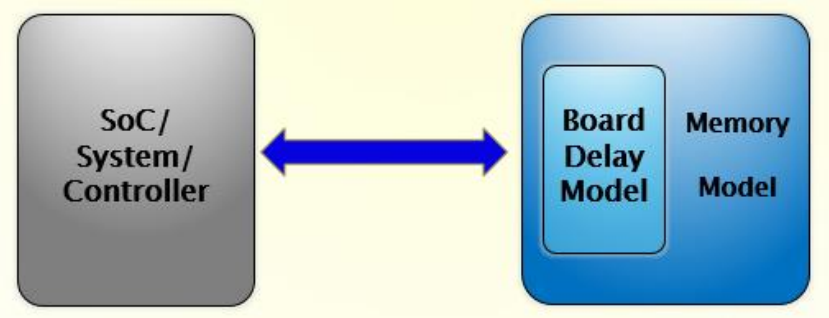

Figure 2. Memory System with encapsulated Board delay model.

A Low Power DRAM(LPDDR)/High Bandwidth Memory(HBM) memory interface contains mainly CLK (differential system clock), CS (Chip Select only for LPDDR), and CA (Command-Address) as input signals and DQS (DQ_strobe), and DQ as bi-directional signals. A device controller initiates command transaction to a memory model through CLK, CS, and CA.

These signals (CK_t/CK_c, CS, CA, DQS, DQ) can have external board delays due to interconnect delays, jitter and skew. Below are the steps for a typical DRAM operation.

- For a write operation, controller sends memory address and transaction data via CA and DQS-DQ signals respectively to memory model.
- For a read operation, controller requests to read data from a specific address in the memory model by sending memory address information on CA bus.
- In response read data comes from model on DQS-DQ signals.

This paper is organized as follows: In section II UVM based memory VIP architecture is explained. In section III Board delay architecture is explained. Section IV demonstrates modelling of external board delays in LPDDR and HBM VIP followed by summary in section V.

.II. UVM Architecture with Memory VIP

As discussed above we took an approach for inducing board delay within a memory VIP. A typical LPDDR VIP architecture that we used to integrate our model is as follows (Fig. 3).

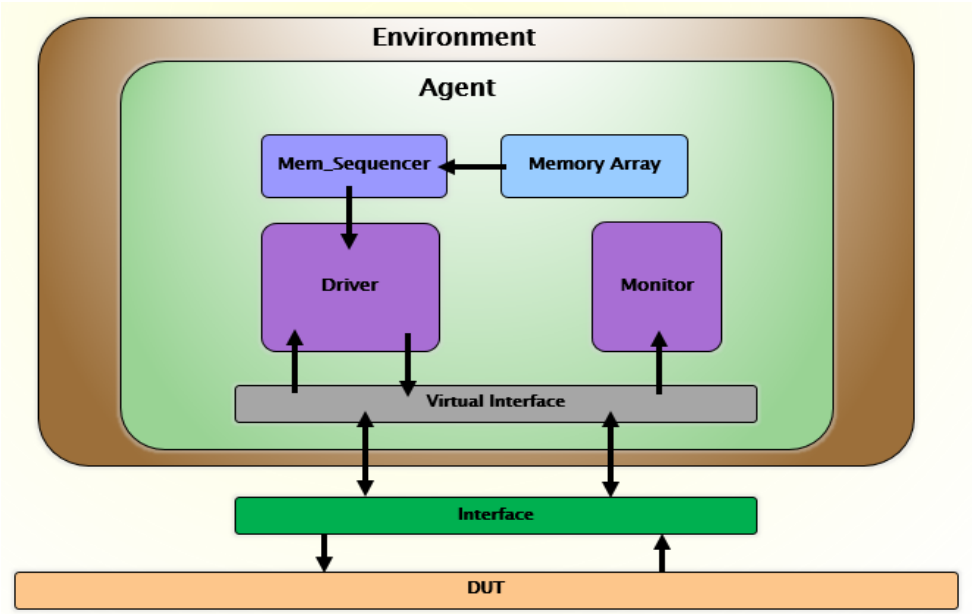

Figure 3. VIP Architecture

As explained a typical UVM Memory architecture has following components:

- **Mem Sequencer**: A typical memory is a reactive component and mem sequencer here plays a special role than a conventional sequencer. Mem sequencer used here is also a reactive component. It has its own inbuild sequences which would be called by driver based on transaction(write/read) request. When write transaction is received from the driver, role of reactive mem sequencer in this case is to write the data in physical memory by getting write transaction information from driver then calling its inbuild sequence to write the data and during a read transaction, the data from the memory is taken into the mem sequencer as a sequence-item and passed to the driver and driven on the interface. For a static memory array, user might not need a mem sequencer, but if user wants to use its own custom memory data array, mem sequencer is very helpful. Advantage of having mem sequencer is to allow user to override mem sequencer's inbuild sequences to initialize/update memory data according to the requirement or to reroute its write/read access towards user's custom memory array.

- **Driver**: A driver sends stream of data/transactions coming from sequencer to DUT (Device Under Test) by converting them into pin level activity. Drivers are used to send controlled stimulus to DUT.

- **Monitor**: A monitor captures the data/transaction information by monitoring the pin-level activity on the bus and converting them streams of data/transaction which can be used for comparison of data/transaction mismatch. Monitor is a passive component means it can't affect the functionality/operation of the DUT in any way.

- **Agent**: An agent encapsulates sequencer, driver, and monitor.

- **Environment**: It's the top-level component of the testbench architecture. It provides access to the hierarchy construction and testbench execution. The class-based environment has virtual interface information and it allows its other subordinate component to use it.

- **Interface**: An interface is a collection of tasks that are used to interact with the classes. Virtual interface is basically a reference to an interface used by UVM. Since virtual interface is a reference to the actual interface to be used inside a class, the classes containing such virtual interface utilizes these virtual interfaces to have direct access to actual bus interface.

UVM Configuration

We provide a typical configuration class to maintain a database of objects and variables to move and share parameters across various testbench components. With this, objects can share their variables, data, methods etc. with other objects. One testbench component can access the object of other component without knowing where its declared in the testbench hierarchy. As a typical UVM method the two main function *uvm_config_db set()* and *get()* are used to share the configuration between different components

III. BOARD DELAY ARCHITECTURE WITH UVM APPROACH

The board delay approach that we are proposing takes the current VIP UVM architecture as explained in the above section. We have modelled these delays in couple of Memory VIP's LPDDR4 and HBM. For inducing board delays, we have added two more components

- External Board Delay Interface
- Intermediate Interconnect class

External Board Delay Interface

This interface named as external board delay interface have exactly same signal information as original device controller signals to be hooked to memory interface.

In a conventional UVM memory architecture the interface that binds memory and controller is configured as virtual interface as explained previously. This interface acts as a medium to pass information from controller to memory model and vice versa (Fig. 4).

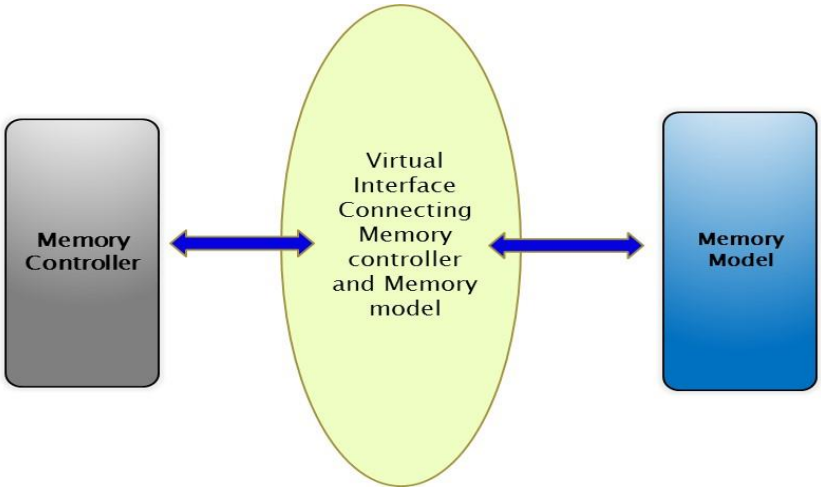

Figure 4 – Virtual Interface between Memory and Controller.

The above architecture has ideal set of signals where the signal transmitted by memory controller reaches the memory model without any delay. To mimic the board delays as per the use case scenarios of real world, another interface is introduced within the memory model itself so that the signals to memory should reach after a delay (Fig. 5).

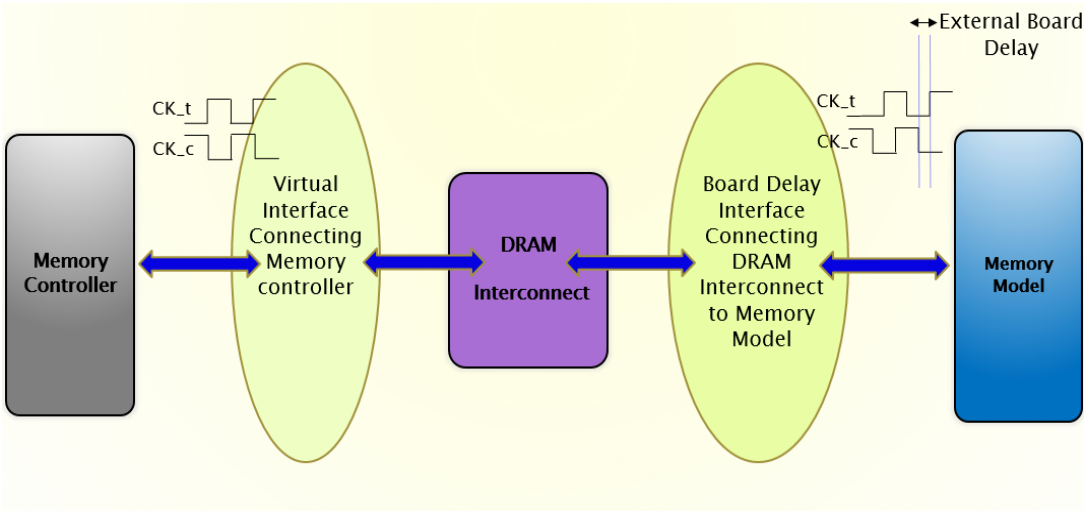

Figure 5. Board Delay components Interface and Interconnect

The board delay interface is a pin compatible interface with original memory interface and is instantiated inside original memory interface. Figure 6 shows the snippet of a board delay interface instantiated within HBM memory interface.

```
interface hbm_ext_board_delay_if();

//---- Interface Ports ---- //

wire ck_t;

wire ck_c;

logic cke;

wire [127:0] dq;

.

.

.

monport drv_port();

monport mon_port();

endinterface:hbm_ext_board_delay_interface
```

```
interface hbm_if();

//---- Interface Ports ---- //

wire ck_t;

wire ck_c;

logic cke;

wire [127:0] dq;

.

.

.

monport drv_port();

monport mon_port();

`ifdef HBM_EXT_BOARD_DELAY_IF

  hbm_ext_board_delay_if ext_board_delay_if();

`endif

endinterface:hbm_if
```

Figure 6. Board Delay interface instantiation inside the original interface

From a user perspective, the interface visible to the memory controller is original interface and board delay interface is completely hidden (Fig. 7).

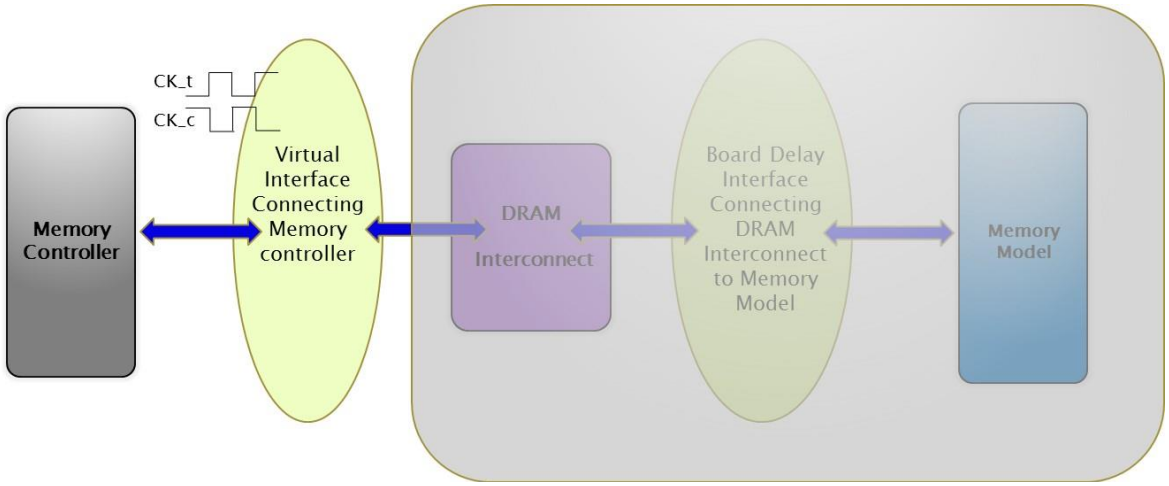

Figure 7. Board Delay interface – Memory Controller perspective

In the same test bench setup user can enable the board delay then memory model will auto configure itself to take board delay interface instead of non-delayed interface. The HBM snippet (Fig. 8) explains that enabling of board delay is controlled by a macro ***"HBM_EXT_BOARD_DELAY_IF"***. If the board delay is enabled, virtual interface used by memory is ***hbm_if.ext_board_delay_if***. So, the user configures the VIP with passing virtual interface (hbm_if). When board delay is enabled HBM VIP picks the board delay interface(ext_board_delay_if) which is instantiated inside the hbm_if.

```
`ifdef HBM_EXT_BOARD_DELAY_IF
  typedef virtual hbm_ext_board_delay_if DEVICE_IF;
  typedef virtual hbm_ext_board_delay_if.mom_port HBM_MON_PORT;
  typedef virtual hbm_ext_board_delay_if.drv_port HBM_DRV_PORT;
`else
  typedef virtual hbm_if DEVICE_IF;
  typedef virtual hbm_if.mom_port HBM_MON_PORT;
  typedef virtual hbm_if.drv_port HBM_DRV_PORT;
`endif
```

```
`ifdef HBM_EXT_BOARD_DELAY_IF
  virtual hbm_ext_board_delay_if vif;
  vif = cfg.hbm_if.ext_board_delay_if;
`else
  virtual hbm_if vif;
  vif = cfg.hbm_if;
`endif
```

Figure 8. Original memory interface replaced by board delay interface

When the board delays are enabled, the only interface which is visible to memory is external board delay interface (Fig. 9).

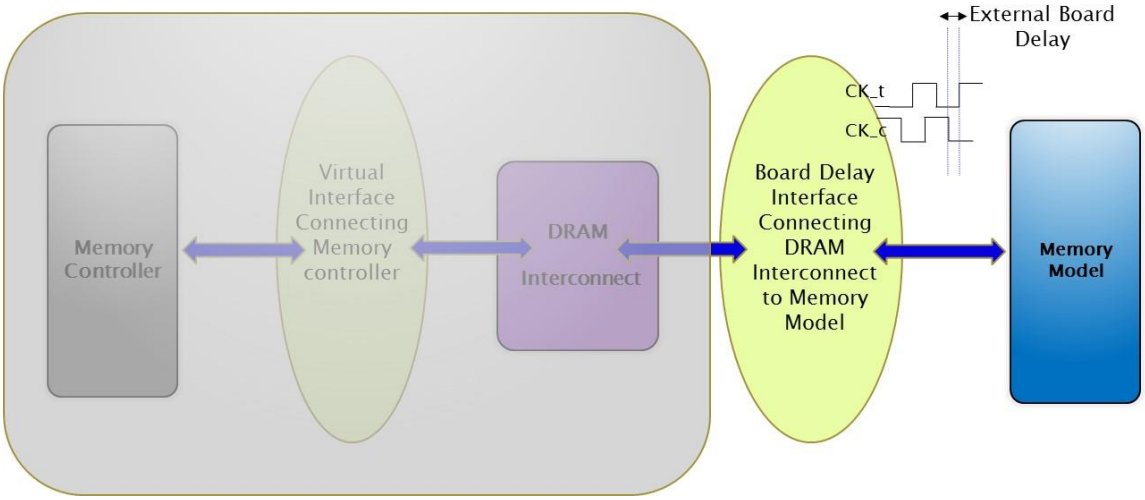

Figure 9. Board Delay interface – Memory Model perspective

Intermediate Interconnect class

As the name suggests, this interconnect class connects board delay interface (visible to memory) to the original interface (visible to controller) as shown in Figure 5. It's a class which delays original input signals from controller to model and similarly signals from model to controller interface. It provides a mechanism to introduce user desired random delays on each pin independently. It senses any change at the interface pins to or from the model and then drives information after some random delay in the required direction. As this class is a UVM component encapsulated inside agent, so this approach provides direct flexibility to the user to calibrate delays during anytime in simulation. For this reason, we preferred this approach rather than using delays directly at the interface level.

Alternate way to introduce delay here could be to use clocking blocks but using them will restrict delays to static values only and therefore cannot be controlled by user. To avoid such restrictions, we have used class based object. With that, now user have control over these board delays as class based declaration will allow these delays to be overridden at run-time.

A snippet of a HBM based interconnect class is shown in Figure 10.

These delays are enabled according to the type of signal that could be categorized in three groups- Input, Output and Inout signals.

- *Input Signals:* For input signals, interconnect class provides the delay to ext_board_delay_if from the hbm_if. For example, the clock(ck_t) signal which is input to the memory model, is delayed by tck_t_fly_by_ps value when it reaches board delay interface (Fig 10).

- *Output Signals:* In case of output signals the flow of information is from memory model to controller side. Here hbm_if is getting delayed information from ext_board_delay_if as depicted in the example snippet (Fig. 10).

- *Inout Signals:* For inout signals the current infrastructure provides two different delays for different data paths. If we take an example of dq signal, which is bidirectional in nature so during read the data path from memory to controller is enabled and hbm_if.dq becomes a delayed version of ext_board_delay_if.dq while during write operation ext_board_delay_if.dq becomes a delayed version of hbm_if.dq. These delays are controlled through internal enable signals which control read or write operation (Fig. 10).

As mentioned, the user configured delays play an important role in defining board delay setup. As described in the setup (Fig. 10) the delay is mentioned as **<signal_name>_fly_by_delay_ps**. Both the data paths have separate delays. For e.g. for read data path delay is dq_rd_fly_by_delay_ps while during writing, delay is described as dq_wr_fly_by_delay_ps. Each signal and corresponding individual pin can be assigned with independent delay. For e.g., in a HBM system where dq signal has a width of 128 bits, the current architecture supports delaying each pin individually. This delay is supported through a UVM configuration and all the delay parameters can be accessed by user through a single configuration object.

```
class hbm_dram_interconnect;
  task hbm_dram_interconnect::run();
    hbm_dram_interconnect_memory_ext_board_delay_connection();
  endtask

  task hbm_dram_interconnect::hbm_dram_interconnect_memory_ext_board_delay_connection();
    begin
       // ------------------- Connecting an input signal -------------------------//

       //*** Connecting ck_t from Controller to Model ***//
       forever @(cfg.hbm_if.ck_t)
         cfg.hbm_if.ext_board_delay_if.ck_t <= #(cfg.timing_cfg.tck_t_fly_by_delay_ps*1ps)
(cfg.hbm_if.ext_board_delay_if.ck_t_en_d===0) ? cfg.hbm_if.ck_t : 'z;

      //*** Connecting ck_c from Controller to Model ***//
      forever @(cfg.hbm_if.ck_c)
        cfg.hbm_if.ext_board_delay_if.ck_c <= #(cfg.timing_cfg.tck_c_fly_by_delay_ps*1ps)
(cfg.hbm_if.ext_board_delay_if.ck_t_en_d===0) ? cfg.hbm_if.ck_c : 'z;

      // -------------------- Connecting an output signal -------------------------//

      //*** Connecting aerr from Model to Controller  ***//
      forever @(cfg.hbm_if.ext_board_delay_if.aerr)
       cfg.hbm_if.aerr <= #(cfg.timing_cfg.taerr_fly_by_delay_ps*1ps)
cfg.hbm_if.ext_board_delay_if.aerr;

     // ------------------ Connecting a bi-directional signal ---------------------//

     //*** Connecting dq , considering DQ_WIDTH = 128 ***//
     for(int i=0; i<128; i++) begin
        automatic int j = i;
        fork
          forever @(cfg.hbm_if.ext_board_delay_if.dq[j])
            //Model to Controller
            cfg.hbm_if.dq[j] <= #(cfg.timing_cfg.tdq_rd_fly_by_delay_ps[j]*1ps)
(cfg.hbm_if.ext_board_delay_if.dq_en[j]===1) ? cfg.hbm_if.ext_board_delay_if.dq[j] : 'z;

          //*** Delaying enable signal dq_en_d ***//
          forever @(cfg.hbm_if.ext_board_delay_if.dq_en[j])
            //Model to Controller
            cfg.hbm_if.ext_board_delay_if.dq_en_d[j] <=
#(cfg.timing_cfg.tdq_rd_fly_by_delay_ps[j]*1ps) (cfg.hbm_if.ext_board_delay_if.dq_en[j]===1) ?
'1 : '0;

          forever @(cfg.hbm_if.dq[j])
            //Controller to Model
            cfg.hbm_if.ext_board_delay_if.dq[j] <=
#(cfg.timing_cfg.tdq_wr_fly_by_delay_ps[j]*1ps) (cfg.hbm_if.ext_board_delay_if.dq_en_d[j]===0) ?
cfg.hbm_if.dq[j] : 'z;
        join_none
     end
endtask

endclass
```

Figure 10. dram_interconnect_class of snippet

IV. EXPERIMENTS and RESULTS

As discussed we demonstrated the above explained board delay architecture with Verification IP's LPDDR and HBM.

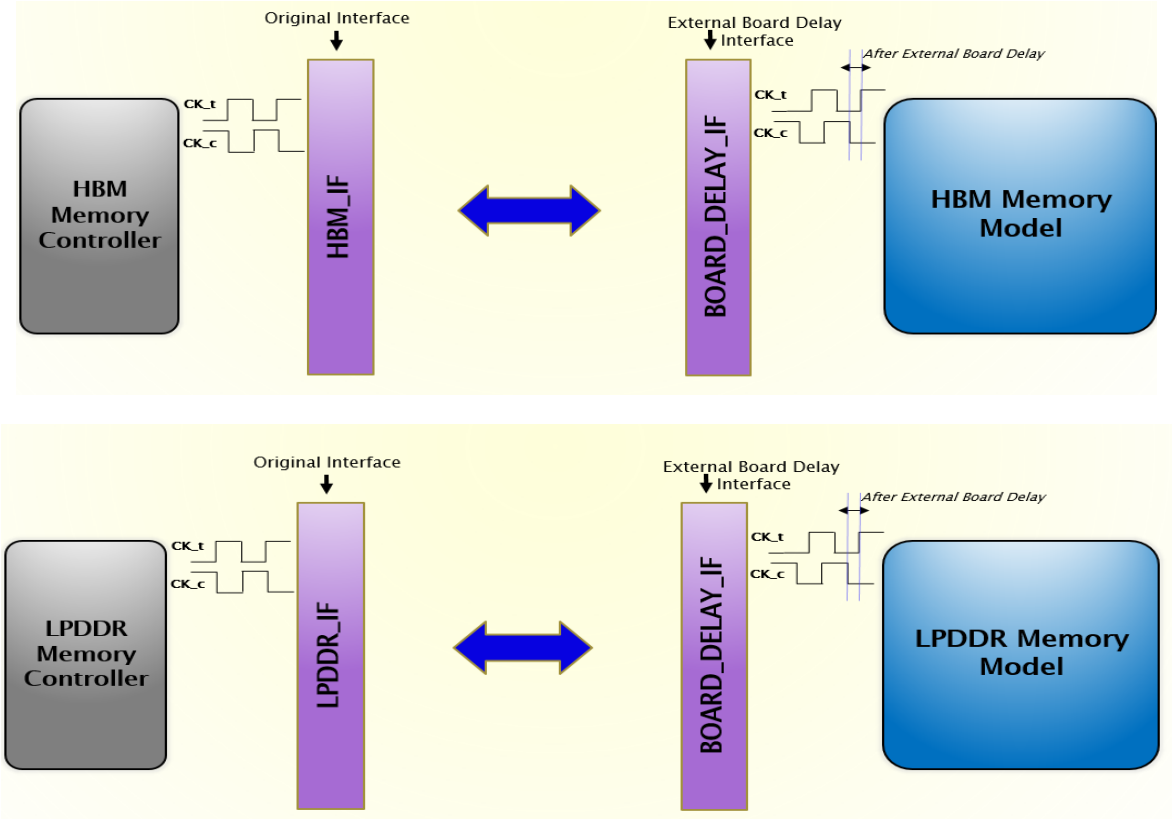

Figure 11. HBM and LPDDR Board Delay Architecture

We focused on multiple random delays as well as fixed delays which helped in more robust verification. We have collected the waveforms with multiple skews and delay models by using board delay architecture.

Differential signals skew: Differential signaling is a method for electrically transmitting information using two complementary signals. This is done to avoid any unwanted electrical noise or crosstalk to maintain the data integrity. In a high frequency system, there is substantial skew available between these differential signals. The current experiment provides a skew of 0.25UI (Unit Intervals) between differential signals, ck_t/ck_c, wdqs_t/wdsq_c and rdqs_t/rdqs_c.(Fig. 12).

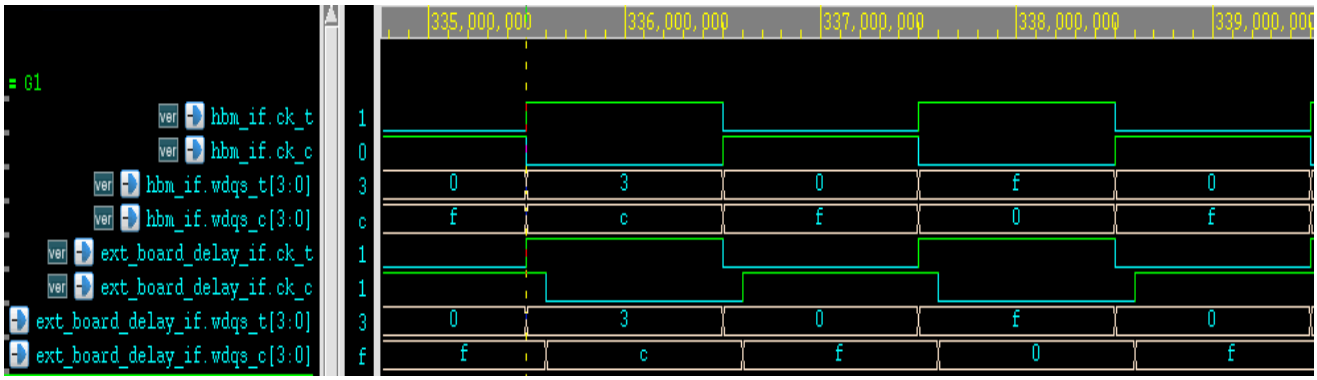

Figure 12. Differential signal skew in HBM VIP

Per transaction delays: As mentioned these delays are re-configurable, so in our experiments we provided different delays during different transactions by reconfiguring these delays in every transaction. Below is the example we took:

- Set *tdq_wr_fly_by_ps[n]* as 700 ps at 0 time.
- Activate any random bank.
- Perform Write transaction.
- Set *tdq_wr_fly_by_ps[n]* as 300ps and reconfigure the device.
- Perform another Write transaction with the new delay value.
- Precharge the opened bank.

The waveform shown describes the different delays that are configured in dq pins in multiple writing transactions (Fig.13).

Figure 13. Per transaction delays in HBM VIP

DQ skew: As mentioned earlier, there is possibility to configure delay on each signal and each corresponding pin independently. So, we monitored the data sampling on both reading and writing by applying data skew between individual DQ pins (Fig. 14).

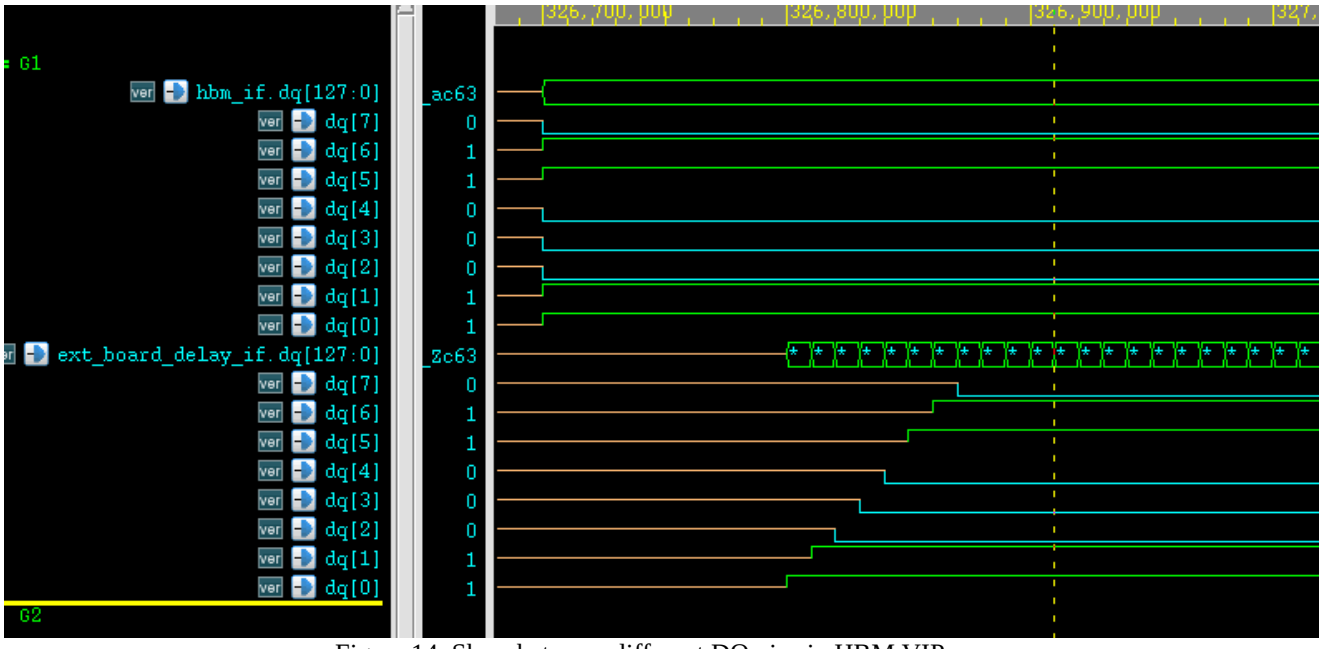

Figure 14. Skew between different DQ pins in HBM VIP

DQ to DQS Skew: A typical DDR memory samples the data on strobe signal and ideal positioning of data is centrally aligned to the strobe. However in real scenario due to induced delay on high frequencies the position of DQ change w.r.t

DQS and Model/Memory are expected to sample the data with given range. We implemented this skew board delay in LPDDR model which is demonstrated in below waveform (Fig. 15).

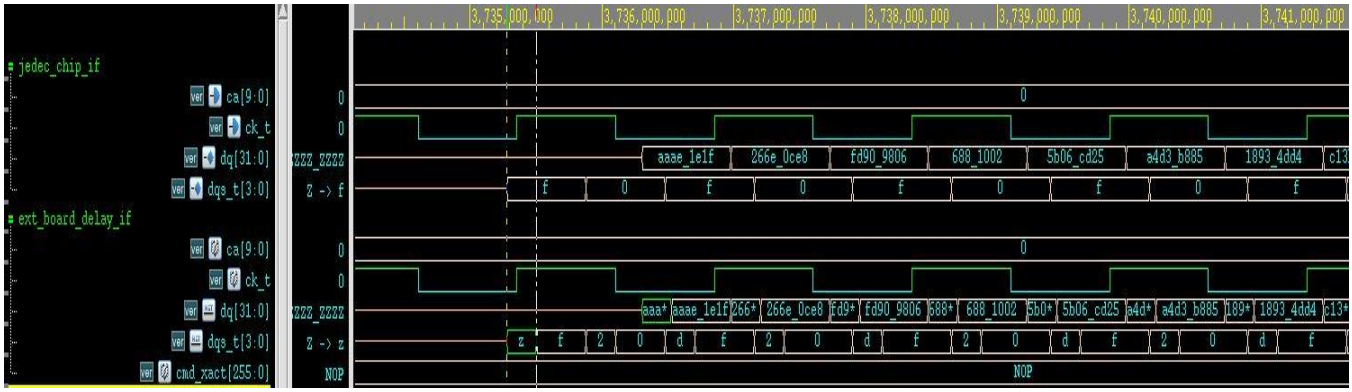

Figure 15. DQ to DQS skew in LPDDR VIP

DQS to CK Skew: In systems, such as LPDDR "fly-by" topology is frequently used, which causes skew between CK and DQS signals. This skew can be different for different DQSx signals. This is shown in the figure below. This requires to adjust each DQSx line separately through some mechanism (e.g. Write Leveling) (Fig. 16).

Figure 16. DQS to CK skew in LPDDR VIP

Address to CK Skew: Controller sends a command by accessing command address pins w.r.t. clock signal. We tested in HBM VIP, a skew delay between address to CK of 0.25 UI using the board delay model and checked if the device reliability is not affected. Random delays were also applied on each command bus(ca) pin independently.

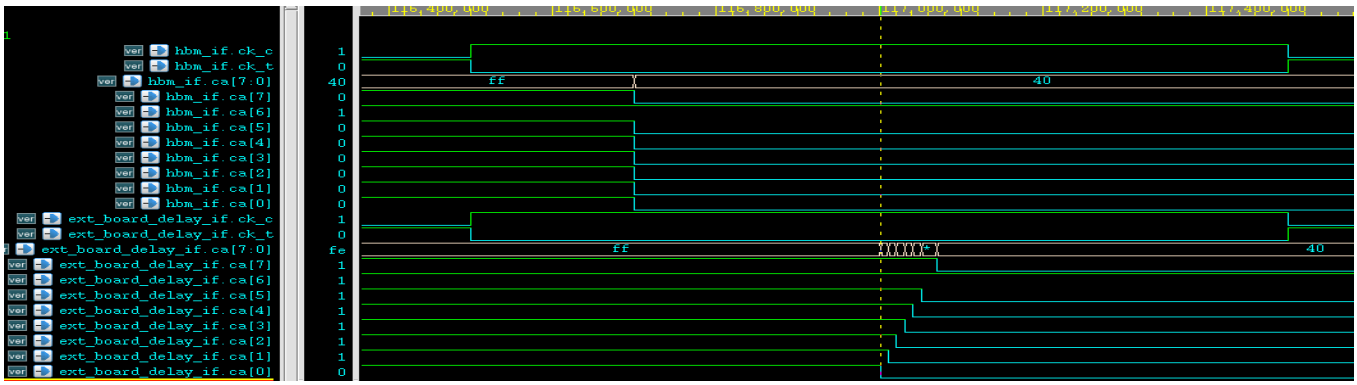

Figure 17. Address bus to CK skew in HBM VIP

IV. SUMMARY

In this paper we have demonstrated a UVM based model to generate random board delays on each pin of a memory oriented interface. These delays are independent and entirely user configurable. It has also been highlighted that the user interface for this architecture remains unchanged as the controller still interact through the original interface. Different skew results were generated to describe the robustness of the proposed design. These methods could be extended to multi-rank and multi-channel environments as well.

We have examined this model on a parallel memory protocol, but this approach can be extended to other parallel as well as serial protocols as future work. Some typical parallel protocol systems such as AMBA (Advanced Microcontroller Bus Architecture) and serial protocol such as PCIE (Peripheral Component Interconnect Express), USB (Universal Serial Bus) etc. can exercise this architecture using the UVM methodology.

V. REFRENCES

[1] *IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language, IEEE Std 1800-2012*

[2] *UVM Documentation* *http://www.accellera.org/downloads/standards/uvm*

[3] *K. Raahemifar , M. Ahmadi, "A design-for-testability technique for detecting delay faults in logic circuits", Feb 1998*

[4] *P. Bose, "Pre-Silicon Modeling and Analysis: Impact On Real Design", Aug 2006*.

[5] *D.S. Gao ; A.T. Yang ; S.M. Kang, "Modeling and simulation of interconnection delays and crosstalks in high-speed integrated circuits", Jan 1990*

[6] *https://blogs.synopsys.com/vip-central/2017/10/03/lpddr4-the-total-package-for-mobile-soc-ram/*

[7] *https://blogs.synopsys.com/vip-central/2017/09/05/lpddr4-what-makes-it-faster-and-reduces-power-consumption/*