Advancements in UVM Test Bench Architecture for Verifying High Speed MIPI MPHY 5.0 IP

Eldhose P.M, Sagar Jayakrishnan, Suraj Vijay Shetty, Kuntal Pandya, Parag S. Lonkar
Samsung Semiconductor India Research (SSIR) Bangalore, India
(eldhose.pm, sagar.j, shetty.suraj, k.pandya, parag.lonkar)@samsung.com

***Abstract*- In recent MIPI MPHY 5.0 specification, the serial data rate is upgraded to 20Gbps per lane, increasing the verification complexity. This has risen up both the verification efforts and the pressure of meeting the time to market in mobile industry in tighter timeline. To mitigate these challenges, many advancements are done in existing MPHY test bench to achieve better quality compared to its predecessors in our journey to qualify MPHY 5.0 IP.**

***Keywords*- MPHY, Verification, Enhancements, Assertions, Coverage, FSM, Jitter**

I. Introduction

Advancements in technology node to meet rapid mobile industry requirements, along with the need to move to higher throughput designs, always demand highest quality of verification. However, for integrating the designs, with logic updates to meet node expectations along with protocol upgrades to required speed expectations, into existing Test Benches always pose a challenge to verification teams.

The challenge is majorly due to high amount of churn that existing Test Bench has to go through to accommodate the upgrades while strictly maintaining the timelines for Time to Market.

In this paper, we are going to discuss various aspects of verifying MIPI MPHY 5.0 IP to overcome some of the above challenges with scalable and reusable Test Bench environment architecture. Previously, verification environment was built focusing towards data path and multiple iteration of regressions to meet functional and code coverage requirements set for the protocol. But, in the newer scheme of things a series of enhancements are done on the top of existing infrastructure. To illustrate the same let us visit how conventional Test Bench was upgraded to help us perform qualitative verification for MPHY 5.0 protocol in a step by step fashion.

II. Conventional Test Bench Architecture

The conventional approach of verifying any serial protocol is connecting various verification components or any third party VIP across the boundaries of Design Under Test (DUT) and applying stimulus at input ports, then output response is checked against a golden reference as shown in Figure 1.

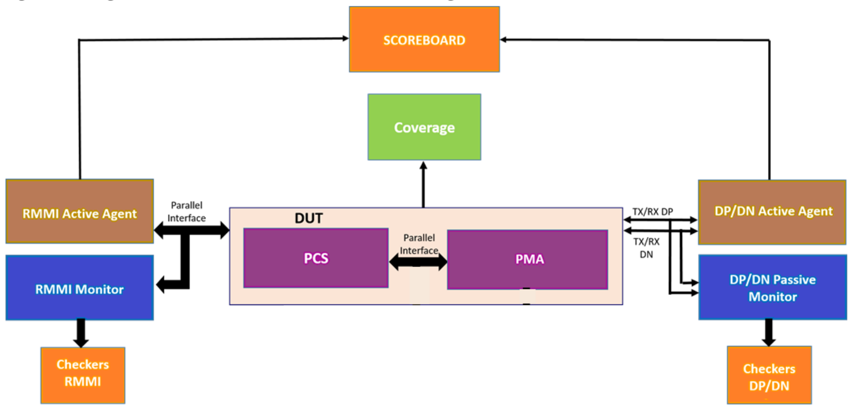


Figure 1 Conventional Test Bench Architecture

III. IMPROVED TEST BENCH ARCHITECTURE

Figure 2 illustrates enhanced Test Bench architecture to ensure Design Verification closure of ultra-high speed and low power MPHY 5.0 Protocol. A sophisticated SSC-Jitter module is connected in between Test Bench clock generation module and third party VIP to generate modulated data, which in turn is connected to receiver serial interface of Design. In addition, this figure also explains about the placements of Unknown and High impedance checker, PCS-PMA Interface and PMA Analog-Digital Interface checkers at various stages of Design.

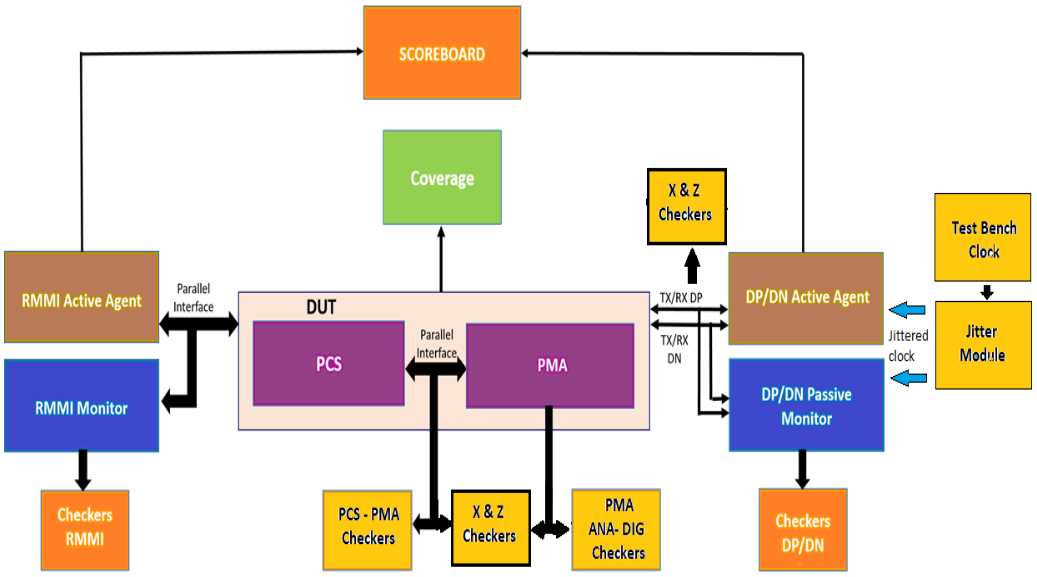


Figure 2 Advanced Test Bench Architecture

This paper also focuses on test sequence enhancements based on MPHY specification FSM, providing flexibility to user for controlling the Test Bench, approaches to achieve faster coverage, increasing regression throughput and debugging assertions.

IV. ENHANCEMENTS AND RESULTS

In this section, we will discuss about the enhancements done in Test Bench and results/improvements achieved based on implementation.

A. FSM Inspired Scalable Test Bench Architecture

In standard Test Bench, test sequences are usually planned keeping focus only towards data path transactions. For example Figure 3(a) below represents MIPI MPHY Tx FSM state diagram, which represents various paths to send the data/bursts. Coming to implementation part, as seen in Figure 3(b), each state machine is coded by using conditional statements. The difficulty lies when additional settings, applicable for each state are required to be incorporated. Hence, this approach makes Test Bench more complex and considerable amount of time is required on enhancements. This inefficiency reduces scalability and reusability of Test Bench.

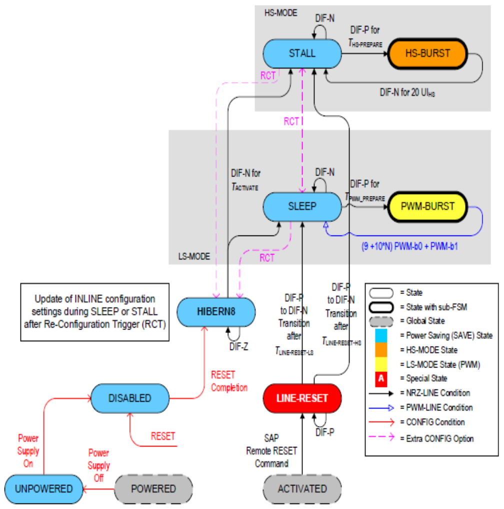
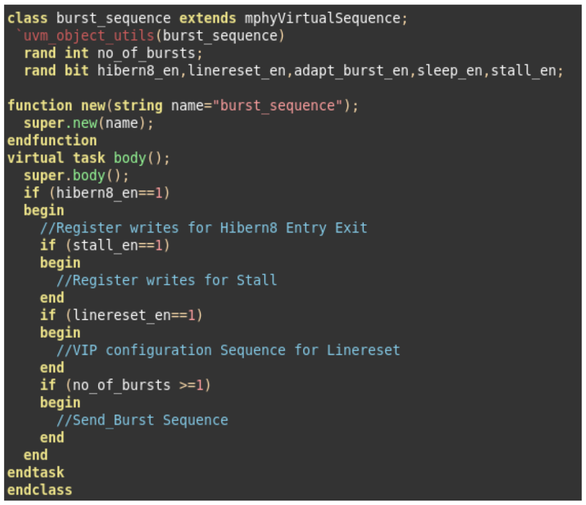


Figure 3(a) MPHY Tx FSM State Diagram

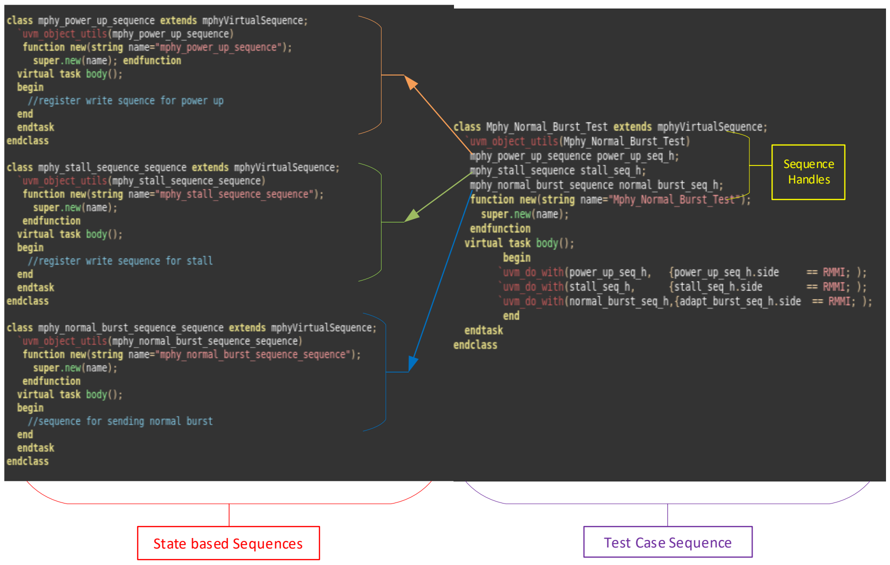


```
class burst_sequence extends mphyVirtualSequence;
  `uvm_object_utils(burst_sequence)
  rand int no_of_bursts;
  rand bit hibern8_en,linereset_en,adapt_burst_en,sleep_en,stall_en;

function new(string name="burst_sequence");
  super.new(name);
endfunction
virtual task body();
  super.body();
  if (hibern8_en==1)
  begin
    //Register writes for Hibern8 Entry Exit
    if (stall_en==1)
    begin
      //Register writes for Stall
    end
    if (lineresetv_en==1)
    begin
      //VIP configuration Sequence for Linereset
    end
    if (no_of_bursts >=1)
    begin
      //Send_Burst Sequence
    end
  end
endtask
endclass
```

Figure 3(b) Pseudo Conventional Sequence Code

To overcome this challenge in verification, sequences are extensively enhanced and modularized based on MPHY states. Each state requirements has been defined as a separate sub-sequence, which in turn becomes part of the more complex main sequence. Figure 3(c) represents the method where FSM states are mapped to individual class sequences and respective sequence can be called from test cases.



```
class mphy_power_up_sequence extends mphyVirtualSequence;
  `uvm_object_utils(mphy_power_up_sequence)
   function new(string name="mphy_power_up_sequence");
     super.new(name); endfunction
  virtual task body();
  begin
    //register write squence for power up
  end
  endtask
endclass

class mphy_stall_sequence_sequence extends mphyVirtualSequence;
  `uvm_object_utils(mphy_stall_sequence_sequence)
   function new(string name="mphy_stall_sequence_sequence");
     super.new(name);
   endfunction
  virtual task body();
  begin
    //register write sequence for stall
  end
  endtask
endclass

class mphy_normal_burst_sequence_sequence extends mphyVirtualSequence;
  `uvm_object_utils(mphy_normal_burst_sequence_sequence)
   function new(string name="mphy_normal_burst_sequence_sequence");
     super.new(name);
   endfunction
  virtual task body();
  begin
    //sequence for sending normal burst
  end
  endtask
endclass
```

```
class Mphy_Normal_Burst_Test extends mphyVirtualSequence;
  `uvm_object_utils(Mphy_Normal_Burst_Test)
   mphy_power_up_sequence power_up_seq_h;
   mphy_stall_sequence stall_seq_h;
   mphy_normal_burst_sequence normal_burst_seq_h;
   function new(string name="Mphy_Normal_Burst_Test");
     super.new(name);
   endfunction
  virtual task body();
        begin
        `uvm_do_with(power_up_seq_h,   {power_up_seq_h.side     == RMMI; );
        `uvm_do_with(stall_seq_h,      {stall_seq_h.side        == RMMI; );
        `uvm_do_with(normal_burst_seq_h,{adapt_burst_seq_h.side  == RMMI; );
        end
  endtask
endclass
```

Figure 3(c) Pseudo Code for State based Sequences

With this approach, the test scenarios can be easily implemented with much lesser efforts. This improves the scalability of Test Bench and sequences became highly reusable.

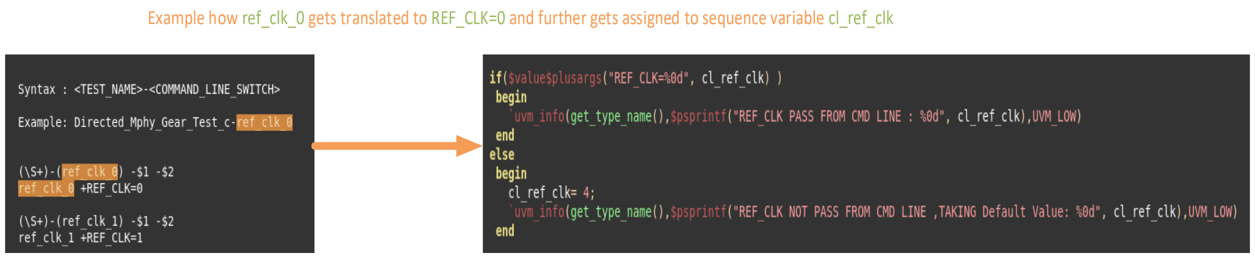
The added advantages for this method are:

1. Easy accommodation of future updates as per specification.
2. Debug for failing cases becomes much easier with modular sequences as compared to single complex sequence.

B. *Parameterized Test Sequence*

High Speed MPHY protocol uses highly sensitive PLL, CDR, BIST designs with different modes of operations, which requires verification with various design settings. Previously, the verification was done by creating multiple compilation builds for each feature. However, maintenance and functional coverage closure with such Test Bench architecture is difficult where quick sign off with coverage is required.

To verify different modes supported within each features with minimal changes in Test Bench, as an improvement, **$value$plusargs API** is extensively used instead of **`define macros**. This helps in generating directed scenarios without changing test sequence manually and hence targeted coverage scenario can be achieved without compiling the Test Bench. Figure 4 below shows the example where switch is passed from command line and how it was translated to desired value and assigned to variable in sequence.


Example how ref_clk_0 gets translated to REF_CLK=0 and further gets assigned to sequence variable cl_ref_clk

```
Syntax : <TEST_NAME>-<COMMAND_LINE_SWITCH>

Example: Directed_Mphy_Gear_Test_c-ref_clk_0


(\S+)-(ref_clk_0) -$1 -$2
ref_clk_0 +REF_CLK=0

(\S+)-(ref_clk_1) -$1 -$2
ref_clk_1 +REF_CLK=1
```

```
if($value$plusargs("REF_CLK=%0d", cl_ref_clk) )
 begin
   `uvm_info(get_type_name(),$psprintf("REF_CLK PASS FROM CMD LINE : %0d", cl_ref_clk),UVM_LOW)
 end
else
 begin
   cl_ref_clk= 4;
   `uvm_info(get_type_name(),$psprintf("REF_CLK NOT PASS FROM CMD LINE ,TAKING Default Value: %0d", cl_ref_clk),UVM_LOW)
 end
```

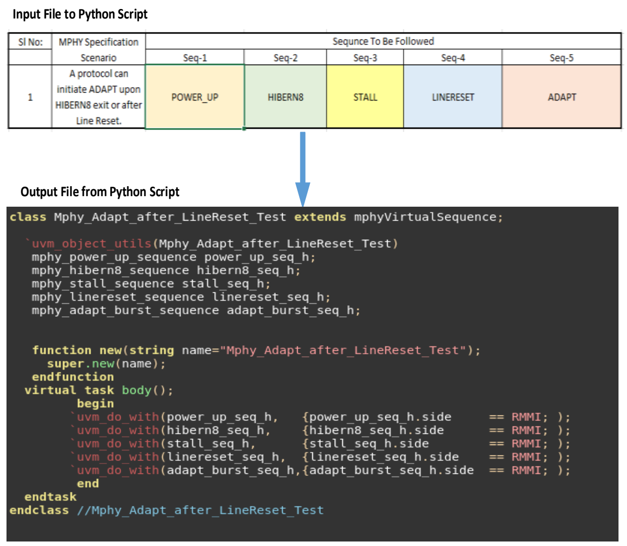
Figure 4 Command line Argument

C. *Automated Python Script for Test Case Generation*

As, MIPI MPHY standard specification has progressed from version 4.1 to version 5.0, complexity of design also becomes double fold compared to its predecessor. Hence, this mandates additional protocol feature verification leading to various new tests and their combinations. Manually writing test cases from scratch for each new scenario identified from specification is time consuming considering the product timelines.

To mitigate the efforts, a test generation setup has been designed using Python Script and Microsoft Excel. Since, the Test Bench Architecture is based on FSM the identified test cases gets translated to respective FSM based sequence.

Python script takes excel derived file as an input where each FSM state is mapped to corresponding DUT level sequence. Once the sequence order is finalized, the python script needs to be evoked to create the required test case. Figure 5 refers to input excel format and sample output file from python script. This approach has come very handy and efficient in creating multiple scenarios in quick time.


Input File to Python Script

Sl No:	MPHY Specification Scenario	Sequnce To Be Followed				
		Seq-1	Seq-2	Seq-3	Seq-4	Seq-5
1	A protocol can initiate ADAPT upon HIBERN8 exit or after Line Reset.	POWER_UP	HIBERN8	STALL	LINERESET	ADAPT

Output File from Python Script

```
class Mphy_Adapt_after_LineReset_Test extends mphyVirtualSequence;

  `uvm_object_utils(Mphy_Adapt_after_LineReset_Test)
   mphy_power_up_sequence power_up_seq_h;
   mphy_hibern8_sequence hibern8_seq_h;
   mphy_stall_sequence stall_seq_h;
   mphy_linereset_sequence linereset_seq_h;
   mphy_adapt_burst_sequence adapt_burst_seq_h;


   function new(string name="Mphy_Adapt_after_LineReset_Test");
     super.new(name);
   endfunction
  virtual task body();
         begin
        `uvm_do_with(power_up_seq_h,   {power_up_seq_h.side     == RMMI; );
        `uvm_do_with(hibern8_seq_h,    {hibern8_seq_h.side      == RMMI; );
        `uvm_do_with(stall_seq_h,      {stall_seq_h.side        == RMMI; );
        `uvm_do_with(linereset_seq_h,  {linereset_seq_h.side    == RMMI; );
        `uvm_do_with(adapt_burst_seq_h,{adapt_burst_seq_h.side  == RMMI; );
         end
  endtask
endclass //Mphy_Adapt_after_LineReset_Test
```

Figure 5 Automated Test Generation

D. Approach for faster Functional Coverage Closure

Regression with constraint randomization is an integral part of Design Verification process as 100% functional coverage is an important milestone for closure. If cross coverage groups involve several cover bins, the amount of time it takes to meet the 100% functional coverage is much higher. Figure 6 describes the usual method of functional coverage closure.

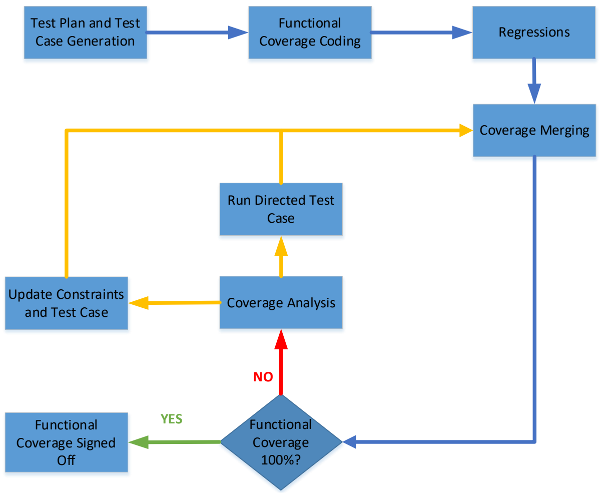

Test Plan and Test Case Generation
Functional Coverage Coding
Regressions
Coverage Merging
Run Directed Test Case
Update Constraints and Test Case
Coverage Analysis
NO
YES
Functional Coverage Signed Off
Functional Coverage 100%?

Figure 6 Functional Coverage Sign Off Process

Running same regression suites for multiple times may not help in achieving 100% coverage. Moreover, the effort for updating random constraints and reconfiguring test cases for hitting uncovered bins consumes more work hours.

While random regressions run in background to make sure no new failures are seen, following measures have been taken to overcome this issue and to achieve maximum functional coverage with minimal efforts:

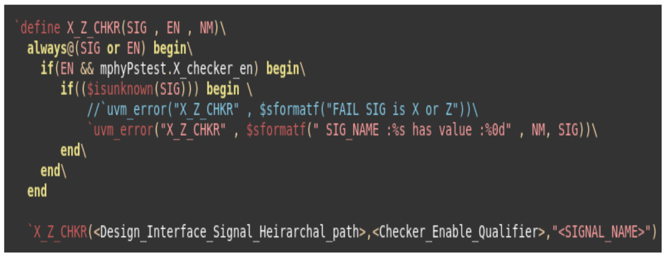
1) The test cases contributing to maximum coverage are ranked and custom regression suite is generated based on the results. With this method, redundant test cases contributing to same bins are discarded.
2) Weighted distribution on cover bins is added wherever possible for quick convergence of complex cross coverages, resulting in faster coverage closure.

E. *Interface Connectivity and Timing Checks*

MPHY Design consists of Physical Coding Sub-Layer (PCS) and Physical Media Attachment Layer (PMA). In addition, PMA further deals with analog and digital logic within. Therefore, connectivity and protocol related checks are applied at different levels of design.

Different ways to verify the interface logic are:

1) **Third Party Formal Connectivity tool:** To ensure the physical connections between interface signal, the tool takes comma-separated-values (CSV) file and register transfer logic (RTL) as input and automatically generates stimulus to verify the connection. As it provides interactive debug environment, the results are faster and easy to debug missing connections.
2) **Interface Timing Verification:** As discussed above the PCS-PMA interface and PMA Analog-Digital interface are crucial part of such designs and ensuring their intactness is critical for quality of IP. We ensure enough checks, on value and signal timings, are applied at these interfaces to ascertain signal exchange between two complex logics (PCS-PMA or PMA Analog-Digital) always happens as expected.
3) **Unknown (X) and High Impedance (Z) checker:** Having a big Analog logic may at times result in intended or unintended X/Z being driven on interface signals for certain cases, which can potentially affect digital simulations. So having such checks is crucial at these interfaces. These checks are clock independent and hence are able to flag any unexpected transitions (X or Z) at any point of simulation. Figure 7 is pseudo code to detect unknown and high impedance.



```
`define X_Z_CHKR(SIG , EN , NM)\
 always@(SIG or EN) begin\
   if(EN && mphyPstest.X_checker_en) begin\
      if(($isunknown(SIG))) begin \
          //`uvm_error("X_Z_CHKR" , $sformatf("FAIL SIG is X or Z"))\
          `uvm_error("X_Z_CHKR" , $sformatf(" SIG_NAME :%s has value :%0d" , NM, SIG))\
      end\
   end\
 end

  `X_Z_CHKR(<Design_Interface_Signal_Heirarchal_path>,<Checker_Enable_Qualifier>,"<SIGNAL_NAME>")
```

Figure 7 Macro for Unknown and High Impedance Checker

F. *Better Approach in Handling Assertions*

- Assertions based on PCS-PMA Interface and PMA Analog-Digital Interface helps to verify timings and overall functionality of design in different states. However, it comes with the cost of increased simulation time and file size especially when reproducing failures with waveform dumps. In order to mitigate this below mentioned steps have been taken:

  i. Runtime switch for enable and disable logic is added w.r.t. each assertion. It has in-fact helped in reducing simulation time up to 30% in certain cases

  ii. There is mostly always the case where certain assertions fail almost every clock cycle or regular intervals resulting in huge and unusable log file. For such cases a disable condition window is added to keep log file cleaner and simulation time unaffected.
- Another aspect is having coverage based on assertions. The presence of uncovered assertions could mean a particular scenario or mode of operation may not have been verified. It can also lead to missing bugs in the design. In order to confirm the same, coverage has been coded for each assertion to ensure it has triggered. This can be then easily analyzed with a coverage analysis tool.

G. *Jittered Clock Generation Module for CDR Stress Testing*

MPHY protocol follows a typical SerDes architecture with a serial transmitter and a serial receiver. The transmitted serial data then passes through the channel and gets exposed to various distortions say ISI, Noise, Jitter etc., resulting in mild to severe modulation w.r.t. actual data. This makes the task of CDR (Clock and Data Recovery), at receiver side, much complex to faithfully reproduce the transmitted data. Hence, it becomes a crucial test scenario to be added in verification suite to ascertain Design works well within tolerance limits set by Protocol.

A specific SSC-Jitter module is hooked in Test Bench, having the capability to modulate any serial data stream, based on input parameters, before feeding the same to receiver logic of DUT. Figure 8 is graphical representation of Jitter Tolerance regression results achieved against specification settings.

Regressions were triggered with around 5000+ seeds with various combinations of analog parameters like ppm offset, jitter frequencies, jitter amplitude, SSC applied on input clock. Then for each jitter frequency, jitter amplitude is swept in small steps in order to determine the failing/tolerance point of Design in terms of data integrity.

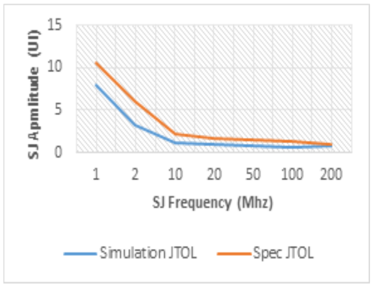


Figure 8 Jitter Tolerance Curve

H. *Increasing Regression Throughput using EDA Tool Option Save-Restore*

In majority of test cases initial configuration and power-up sequence is common. As highlighted in Figure 9, design need to exercise the power supply sequence. Moreover, MIPI MPHY specification mandates a warm up wait time of 1.5ms on Hibernate exit.

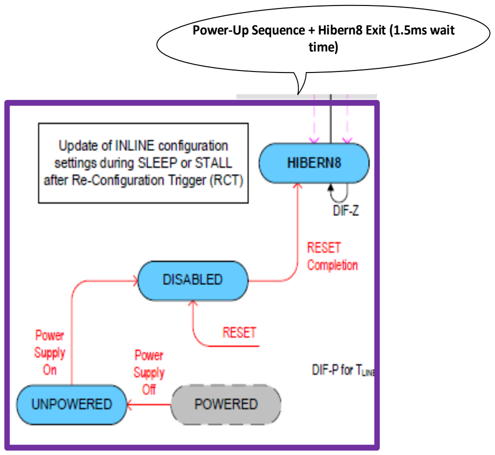

Power-Up Sequence + Hibern8 Exit (1.5ms wait time)
Update of INLINE configuration settings during SLEEP or STALL after Re-Configuration Trigger (RCT)
HIBERN8
DIF-Z
RESET Completion
DISABLED
RESET
Power Supply On
Power Supply Off
DIF-P for T_LINE
UNPOWERED
POWERED

Figure 9 Region of Save-Restore method applied in Advanced Test Bench

On analysis of majority test scenarios in the verification environment, this common sequence consumes approximately one-third of total simulation time. Hence, if we can pre-load our simulation to reach this design state quickly it provides a huge possibility of saving critical simulation time for all applicable scenarios. The simulator has an option, from where the saved snapshots can be reused to start a new sequence from the point of last saved data. This feature is harnessed in the Test Bench over multiple test seeds in regression suite and thus increases the throughput by saving good amount of regression time.

I. *Usage of High Precision Analog Models*

For typical AMS designs like MPHY, for digital simulation purpose, analog behavior is modelled at a high level to mimic the functionality. However, for an efficient testing of analog features, it always helps to have models having real data types and related logic, taking them much closer to real analog behavior. Such models were added as final verification signoff criterion and multiple features like Eye Monitoring, Jitter Tolerance, calibration etc. were qualified using these models. For example, Eye Monitoring test is performed by sweeping sampling clock phase shifter in horizontal axis and voltage reference in vertical axis. For each of their values, Pass/Fail criterion is captured and later used for viewing the Eye Opening in graphical format.

FUTURE SCOPE

The current enhancements have significantly improved the scalability and reusability of the test bench while helping in quicker coverage signoff. On top of these, we are working on some more enhancements to improve coverage collection even further. Changes (minor or major) in existing designs to accommodate derivatives and customer requests are quite common, though the design impact is less but accommodating these features into coverage framework requires considerable effort. A flexible coverage setup capable for providing mechanism to easily add/drop relevant cover bin is always an added advantage.

The few such identified areas are :-

1) Introducing a coverage closure tracking sequence with mapping of all possible combinations of cover bins and its cross coverages. In UVM pre-randomization phase, add controls for enabling and disabling the randomization of variables based on tracking the progress of respective coverage closure knobs.
2) Another method is by mapping all cover bins to an associative array. Record each unique randomized value into the array as covered or not covered during each randomization call. In UVM post-randomization phase, manipulate the current randomized value if the coverage is already hit by walking through an array.

The enhancement works related to these improvement items are in progress.

CONCLUSION

In this paper, we have tried to demonstrate various steps taken and enhancements done in our existing Test Bench to make it much more efficient as well as effective in fulfilling the design challenges with higher speeds and much complex logics. For ex. Interface assertions helped us to unearth some really critical bugs, which could have led to excess power consumption, regressive CDR testing ensured high quality jitter tolerant receiver logic while at the same time with runtime switches, save n restore, quicker turn around on coverage closure helped us big time in closure of various verification activities.

REFERENCES

[1] mipi_M-phy_specification_v5-0.pdf, by MIPI Alliance
[2] SystemVerilog 3.1a Language reference Manual
[3] https://www.academia.edu/30128384/SERDES_Rx_CDR_Verification_using_Jitter_Spread_spectrum_clocking_SSC_stimulus