Migration from Verilog based Test Flow to UVM Environment of USB -PHY

Shailesh Kumar, Tarun K. Gupta, O.P Meena and Ajay Kumar Dadoria

Department of Electronics and Communication Engineering Maulana Azad National Institute of Technology, Bhopal, (M.P.) India-

Abstract
Increase in integration of various components in single system on chip, the verification complexity for such designs become the bottleneck now a day. In the continuous pursuit of increasing the effectiveness of verification for such blocks, UVM based verification is a step ahead of the traditional directed testbench functional verification techniques. To enable faster simulations and stay time to market, we must have flexibility to reuse our environment. UVM provides scenario for CDV, which adds self-checking testbench, faster simulations, coverage matrices, and automated test generation. We propose UVM-based environment for USB-PHY (Transceiver). We have used different UVM components in aspect to make it more flexible and structural. Through constrained random stimuli we reduce time and effort for creating hundreds of tests. We have defined assertions and covergroups derived from functional specification to ensure design quality and reduce time spent on verify the designs.
Keywords:
UVM, SystemVerilog, Assertion and functional coverage

1. Introduction

The complexity of SoCs there is a need for robust design verification, especially for the analog components. Pre-silicon verification of such complex systems is a primary requirement to achieve time to market for any product. Ease of SoC integration, behavioral models is used to mimic the functionality of these analog blocks. In traditional Verilog flow, raw vectors processes from a file and use those to change values of the wire connected to the DUT, then wait for the response of the DUT and dump it into another file. This is quite good and straight approach till the design is simple but can't be reliable for very complex system.

UVM has almost become the company standard for verification now days, it supports all SystemVerilog construct and will compile on all simulators which support SystemVerilog standard. UVM environments having reusable verification components, provides strong functional verification as well randomized scenario. The methodology provides constraint random verification which can ensure avoidance of invalid driving and provide stimuli of our choice. The UVM architecture encourages modular and layered verification environments, where verification components at all layers can be reused in different environments.

The USB specifications [3] define the physical layer of high speed serial links. In USB PHY data transfer occurs through D+ and D- pads using differential signaling. The PHY therefore consists of a high speed transmitter and a differential high speed receiver. Transceiver block is the Analog front-end of the IP consisting of transmitter, receiver, bias, termination etc.
Transceiver is an analog block, which is replaced by its behavioral [1] model during SOC verification to reduce the simulation time. It must be thoroughly functionally verify before integrate into the SoC level.

2. Traditional VERILOG Directed Test Flow

In directed verilog [6] testbench approach as shown in Fig.1, TOP module includes signal declarations (Input, output and internal signal etc.), variable declarations, module instantiations, expected output generation logic and checkers etc. We can instantiate one or more module in a single top module. Stimuli are applied into the design and observed the outputs in simulation waveform window using deferent CAD tools. We can't apply enough test vectors to thoroughly verify the design, one way to achieve this via random constrained stimuli, is a great way to uncover unexpected bugs, because it give enough time and resources to allow entire state space to be covered.

Secondly there is no provision of functional coverage in Verilog based test bench, but it can be achieved by SystemVerilog or UVM. Manual inspections of output waveforms for all the test cases are quite time consuming and there may the chances of human error. Since UVM is an automated environment, it reduces the manual effort to verify functionality.

Manuscript received March 5, 2026
Manuscript revised March 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.3.16

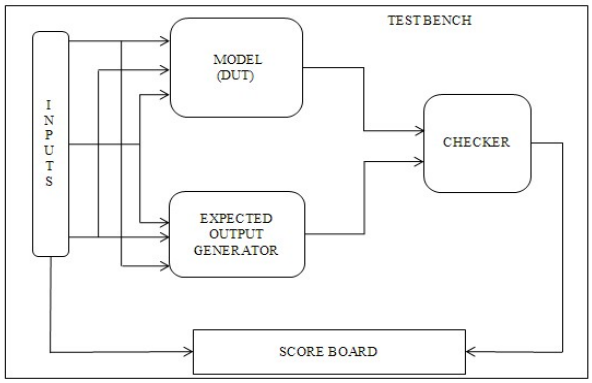

Fig.1 Traditional Verilog based verification environment

3. Migration to UVM

Verification of complex IP must not be reliant on manual inspection of waveforms and stimulus sets. Functional verification must be automated and should have coverage features; it takes time to create but improved quality of verification. UVM [8, 9] provide reusable mechanism in terms of SystemVerilog classes and uVC. Productivity can be increased by reusing verification components; it implies an important goal of UVM.

Since every verification component in UVM having its own intended work and responsibilities, and accordingly they are configured and used. Any component can be hierarchical access, overwritten or extend. Transaction level modeling. To validate the functionality of the design an exhaustive UVM-based environment of USB-PHY (Transceiver), shown in Fig 2, has been created. UVM offers a structured verification environment. We have Derived different UVM classes from parent components.

3.1 Implemented UVM architecture for USB-PHY (Transceiver)

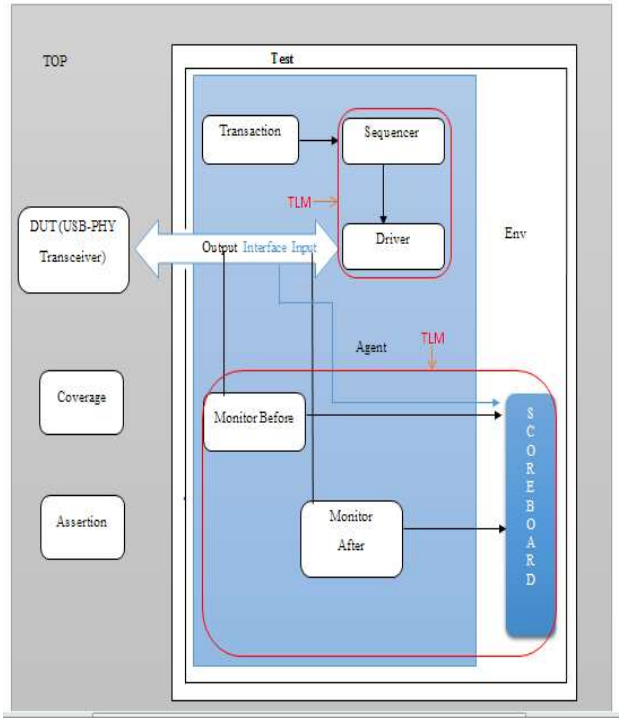

Fig.2 UVM environment for USB-PHY (Transceiver)

We have added covergroups and assertions to improve the quality of verification. Transaction level modelling provides communication between different components of UVM.

A. Transactions class which extended from class uvm_sequence_item, represent the input and output to the DUT. For HSTX mode of operation INHSDATAN/P_1V8 is the input data which may be fully randomized or constrained. Transaction class keeps the records of all the transactions between the components.
To avoid invalid driving in INHSDATAN/P_1V8, defined constrained randomizations shown in Fig.3. Only signals defined in transaction class are recorded by this transaction component.

```
Constraint inp{INHSDATA_1V8 !=
INHSDATA_1V8 ;}
```

Fig.3 Constraints inputs

B. We have derived Sequence class from base class uvm_sequence, it controls the flow of input data to the Transceiver. Upon request from the driver, sequencer provides the sequences to the driver. More the sequences much better the randomizations means more the test cases and thorough the verification.

C. Driver class extends from base class uvm_driver, driver gets the sequences from sequencer and convert the transaction into wiggles. Driver sends these wiggles to the system interface since there is another component (monitor) to observe the response, driver only drives the limited input to the Transceiver, and wait for Transceiver to execute.
For the above case, we have used complete random scenario, describe in Fig.4, in which signals going into the high speed transmitter blocks are randomized.

```
task HSTX_RANDOMIZED();
dut_vif.INHSTERMENABLE_VDDX     <= req.INHSTERMENAB
dut_vif.INHSDRIVERENABLE_VDDX   <= req.INHSDRIVERENA
dut_vif.INHSRECENABLE_VDDX      <=req.INHSRECENABLE
dut_vif.INHSENVENABLE_VDDX      <=req.INHSENVENABLE
dut_vif.INLOOPENABLE_VDDX       <=req.INLOOPENABLE_V
endtask
```

Fig.4 Fully randomized input condition

We have defined a virtual clock to synchronize the input signal applied to the Transceiver (since Transceiver is clock less). The actual simulation always take place in run phase, so whatever the input to be applied is derived in run phase.

D. Monitor act like a watch dog, as it samples the input/output of DUT from system interface. Here we have defined two monitor class- monitor before and monitor after, both classes are extends from uvm_monitor.

As shown in Fig.5, Monitor before samples the response from transceiver and write to the analysis port. This analysis ports connected to the analysis port of the agent.

```
begin
my_tx.DP = vif.DP;
my_tx.DN = vif.DN;
my_tx.HSDIFFRECOUT_VDDX = vif. HSDIFFRECOUT_VDDX
end
ap_monitor_before.write(my_tx);
```

Fig.5 Observing output response and writing to analysis port

Since at block level there is many rigorous test cases apply rather than system level, predicting the output according to the condition as well as monitoring the respected output from the DUT is quite a complex job.
The Fig.6 shows that Monitor after samples input INHSDATAP/N_1V8 and predict output according to input and converted into the transactions, it has analysis port ap_monitor_after where it writes the sampled inputs and predicted output.

```
begin
my_tx.INHSDATAP_1V8 = vif. INHSDATAP_1V8;
my_tx.INHSDATAP_1V8 = vif. INHSDATAP_1V8;
predictor();
ap_monitor_after.write(my_tx);
end
```

Fig.6 Sampled inputs and predicted output write to analysis port

E. agent class derived from uvm_agent, comprises of driver, sequencer and monitor before or after, emulates and verify the Transceiver. For Transceiver we have defined different four agent for all for mode of operation since verification component can have more than one agent. The analysis ports of monitors have been connected to the analysis ports of agent.

F. Scoreboard [10] takes the signals from monitor and performs operation on these. Scoreboard class derived from base class uvm_scoreboard, it compares, as shown in Fig.7, the predicted outputs with the sampled outputs of the transceiver. If predicted and actual outputs DP/DN and HSDIFFERECOUT are matched then it print the message that test case PASSED otherwise FAILED.

```
begin
    before_fifo.get(transaction_before);
    before_fifo.get(transaction_before);
    compare;
end
```

Fig.7 comparison of actual and expected output

G. Environment (env) is top level component of verification, it contains configuration properties that enable to customized topology and behavior and make it reusable. We have defined env from parent class uvm_env, env connects the scoreboard and agent's analysis ports. For every test we have defined different environment since the entire component are not usable for particular mode of operation.

H. Test class enables us to apply specific test cases (stimulus to the DUT), organize our test bench in structural manner. We are having four modes of operations LFSTX, LFSRX, HSTX, HSRX, therefore defined four test class. The particular test to be run is been given in command line, it call its env and simulate it in run phase of the execution.

I. Testbench TOP connects between DUT and the test class using virtual interface. Virtual interface has been registered into the UVM factory via uvm_config_db while calling set method. Virtual clock has been generated here and initialized the default value of the signals.

4. FUNCTIONAL COVERAGE

UVM is advantageous to find functional coverage of the design. There are two ways to analyse the functional coverage, first is to define assertion and secondary by writing the covergroups.

4.1 SystemVerilog assertion

SystemVerilog concurrent assertion [2] has been added to improve the quality of verification. The property of High speed transmitter is evaluated at every positive edge of virtual clock. Fig.8 shows SystemVerilog assertion for high speed transmitter mode, where the property defines that DP/DN must follow INHSDATAP/N_1V8

```
`ifdef assertion_HSTX
 property XCVR_HSTX1;
@(posedge dut_if1.vir_clk) if (dut_if1.VBGIN &&
dut_if1.IREXT50U1 &&
dut_if1.INBIASENABLE_VDDX &&
dut_if1.INPROTENABLE_VDDX
&& !dut_if1.INSERECENABLE_VDDX
&& !dut_if1.INLFSRECEIVERENABLE_VDDX &&
dut_if1.INHSDRIVERENABLE_VDDX &&
!dut_if1.INLFSDRIVERENABLEP_VDDX
&& !dut_if1.INLFSDRIVERENABLEN_VDDX &&
dut_if1.INHSTERMENABLE_VDDX
&& !dut_if1.INHSRECENABLE_VDDX
&& !dut_if1.INHSENVENABLE_VDDX
&& !dut_if1.INLOOPENABLE_VDDX
&& !dut_if1.INHSDISCONENABLE_VDDX &&
dut_if1.DP !== 1'bx && dut_if1.DN !== 1'bx )
((dut_if1.INHSDATAP_1V8 !== 1'bz) ||
(dut_if1.INHSDATAN_1V8 !== 1'bz) ) |-> (dut_if1.DN
=== dut_if1.INHSDATAN_1V8 && dut_if1.DP ===
dut_if1.INHSDATAP_1V8 );
endproperty
assertion_XCVR_HSTX1 : assert property
(XCVR_HSTX1) else $warning("PROPERTY NOT
SATISFIED, ASSERTION FAILED ");
`endif
```

Fig.8 SystemVerilog assertion for high speed transmitter

4.2 Covergroups for Functional Coverage

Fig.9 describes that we have added covergroups [5] for high speed transmitter mode where INHSDATAN/P are the inputs and DP/DN are the outputs.

```
covergroup cg @ (posedge dut_if1.vir_clk);
out_INHSDATAN : coverpoint dut_if1.INHSDATAN_1V8;
out_INHSDATAP : coverpoint dut_if1.INHSDATAP_1V8;
out_DP : coverpoint dut_if1.DP;
out_DN: coverpoint dut_if1.DN;
endgroup
cg cg_int = new() ;
```

Fig. 9 covergroups defined for HSTX mode

5. RESULTS AND DISCUSSION

The below waveforms, shown in Fig.10, of high speed transmitter mode shows the randomization of input stimuli, race free simulation covering maximum test cases. We have gone through fully randomized input conditions of high speed transmitter mode of transceiver

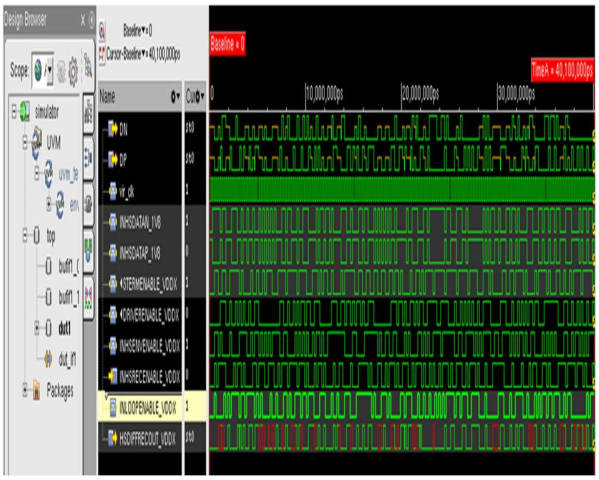

Fig.10 Input-output waveform of high speed transmitter

Transaction level communication [4] provides transaction between different verification components to increase the efficient verification. Input data INHSDATAP_1V8 and INHSDATAN_1V8 should not be high or low simultaneously to maintain the differential swing; it means constraints are defined in terms to avoid invalid driving.

5.1 Functional coverage

The below snippet of functional coverage report in Fig.11 shows the cover points defined are 100% triggered, raised our verification quality.

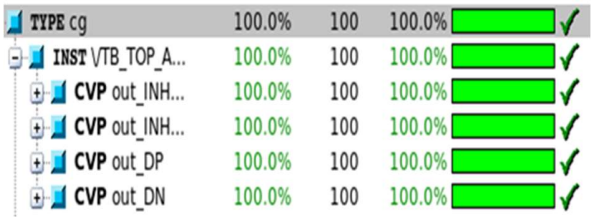

Fig11. Functional coverage of high speed transmitter mode

6. CONCLUSION

In this paper, we have shown departure from Verilog based testbench to UVM environment. Previously there had been a directed testbench used for functional verification of USB-PHY transceiver.

Verilog test flow	UVM
Take care Race between testbench and design	Each component in synchronization
Manual display and monitoring	Auto reporting (Ease in debugging)
Every new test case need compilation	Run multiple test without overhead of compilation
No provision for CDV	CDV

References

[1] M.H. Zaki et al, “Formal verification of analog and mixed signal designs: a survey”, Microelectronics Journal, pp. 1395-1404, **39** (2008).

[2] Tao, Yunfeng. "An introduction to assertion-based verification." In *ASIC, 2009. ASICON'09. IEEE 8th International Conference on*, pp. 1318-1323. IEEE, 2009

[3] Universal Serial Bus 2.0 Specification, USB Implementers Forum, (2000).

[4] C. Ming song and P. Mishra, “Assertion-Based Functional Consistency Checking between TLM and RTL Models”, in 26th International Conference on VLSI Design and the 12th International Conference on Embedded Systems, pp. 320-325, 2013

[5] Mehta Ashok B., “SystemVerilog Assertions and Functional Coverage”, Newyork, Springer, 2014

[6] Language reference manual, IEEE Standard for Verilog® Hardware Description Language, IEEE Std. 1364-2005.

[7] Roman Wang, Yu Peng, Tang Jin, “Best practices of simplifying code coverage unreachability analysis with IEV in OVM/UVM metric-driven verification”, CDN Live Cadence user conference, Beijing, china, 9 aug 2012.

[8] www.verificationacademy.com

[9] UVM 1.2 Class Reference, Accellera Systems Initiative Inc.

[10] Clifford E. Cummings, “OVM/UVM Scoreboards - Fundamental Architectures”, SNUG (2013)