A Comprehensive High Speed Link Verification Test Bench Harnessing Scripting to Achieve Faster Functional Coverage Closure.

Piyush Tankwal (p.tankwal@samsung.com), Arnab Ghosh (arnab.ghosh@samsung.com), Piyush Agnihotri (piyush.a@smasung.com), Mukesh Gandhi (mukesh.g@samsung.com), Parag S Lonkar (parag.lonkar@samsung.com)
Samsung Semiconductor India Research (SSIR), Bengaluru, India.

***Abstract*- This paper introduces a verification framework tailored to the challenges of High-Speed Link Design Verification (DV), with a focus on improving efficiency and maintaining high-quality results. In the context of complex High Speed Link designs such as UniPro, where an extensive number of functional coverage bins must be addressed within strict time constraints, this work offers a comprehensive approach. The proposed framework incorporates a "Smart Coverage Closure Technique" that leverages test ranking and Perl script integration. This technique not only expedites the coverage closure process but also enhances its effectiveness. The script facilitates the creation of a directed test suite by extracting data on un-hit coverage bins, which precisely targets and covers specific bins with a single test run, optimizing resource utilization. Also, to find bug at the initial stage of verification some scenarios are proposed which creates simulation condition close to the real world by incorporating skew, lane mapping. Different kind of techniques, checkers and automation are also discussed to ensure robust verification framework.**

I. PROPOSED WORK AND APPLICATION

The proposed work aims to address challenges in the High-Speed Link DV (Design Verification) process by introducing a general verification framework. Our framework is designed to reduce the time and effort required for verification activities while ensuring high-quality results. Several key strategies are discussed:

The work proposes scenarios that enable the detection of bugs at an early stage to improve the efficiency of the verification process. These scenarios aim to create simulation conditions that closely resemble to real-world conditions. They incorporate elements like skew injection in between the lanes and lane mapping for cross lane connection between lanes for the IP having more than one lane, which can help uncover potential issues in the design early on.

The link designs are usually complex, in term of the features and cross scenarios they support hence the functional coverage bins turn out to be humongous. So, to meet the coverage goals for all these bins within time constraints can be challenging and to address this issue, a smart coverage closure technique is proposed. This technique involves test ranking and Perl script for coverage holes. The script is responsible for extracting statistics on un-hit bins, and then it creates a directed test suite to target those specific bins with a single test run. This approach optimizes the coverage closure process.

In summary, the proposed general verification framework aims to enhance the High-Speed Link DV process by adding robust checkers, improving functional coverage closure, and facilitating early bug detection. These strategies are geared towards streamlining the verification process while maintaining high-quality results. Though we have taken reference of UniPro IP in multiple places, these strategies are not limited to specific Link IP and can be extrapolated widely to any other High Speed Link IP's.

A. High Speed Link-PHY Test Bench Architecture

Our robust testbench encompasses a wide range of verification aspects to ensure the quality, functionality, and performance of the design under test. Here's an overview of the components we've mentioned in Fig1.

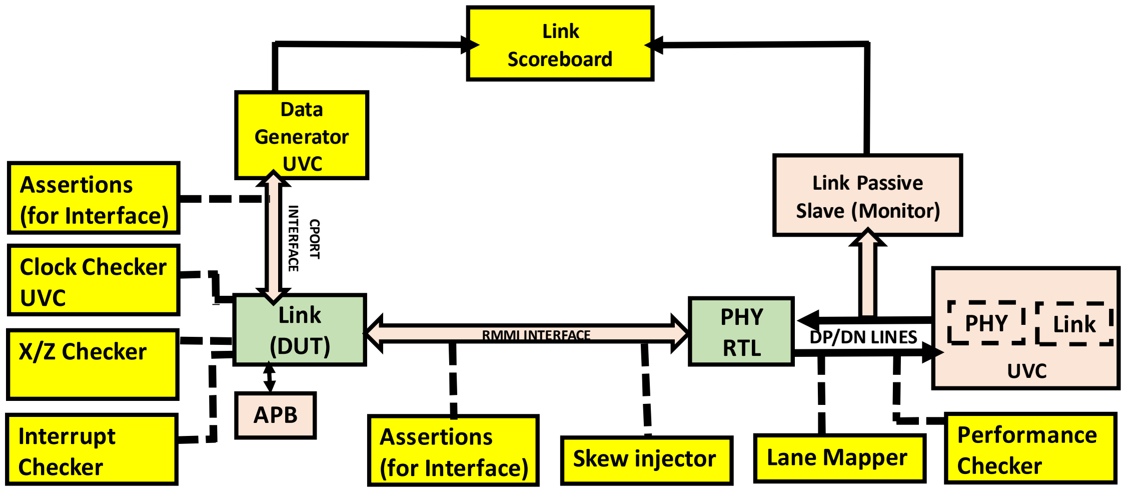


Figure 1. Configurable Robust Test bench (Reference: UniPro2.0 Link-PHY interface).

1) *Performance Analysis:* Our testbench goes beyond functional verification by including performance checks. Specifically, we're evaluating two key performance metrics:

Throughput: This metric measures the rate at which data is processed or transmitted through the design. It helps ensure that the system can handle data at the expected rate. Few factors like Link protocol overhead at different message, frame size and PHY protocol overhead in different power mode and gear can reduce the actual throughput of the DUT than the theoretical maximum.

Latency: Latency measures the delay or time taken for data to traverse the design. Low latency is often crucial in real-time systems, so verifying this aspect is important.

2) *Save-Restore*: In High-Speed Link, initialization itself can take more than half of the simulation time that delays the verification process. So, to avoid that we can save the repetitive process of Link Up (Link Enable and Training) and re-store it when required. Though it is not a novelty but for the completeness and general awareness of DV engineers we would like to highlight the significance of this step. We have achieved closed to 50-60 % efficiency improvement using it.

3) *Skew Injector:* Inter-lane skew is a common phenomenon at the serial lanes and standards are mentioned in the PHY specification. This skew is passed to the upper layer and it is responsibility of the Link layer to merge the data correctly. So, skew injection at Link-PHY interface is required considering all possible scenario and adequate randomization for proper verification sign-off, which is achieved by configuring the Clock UVC appropriately.

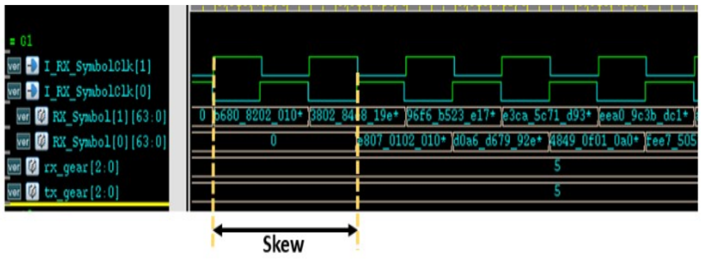
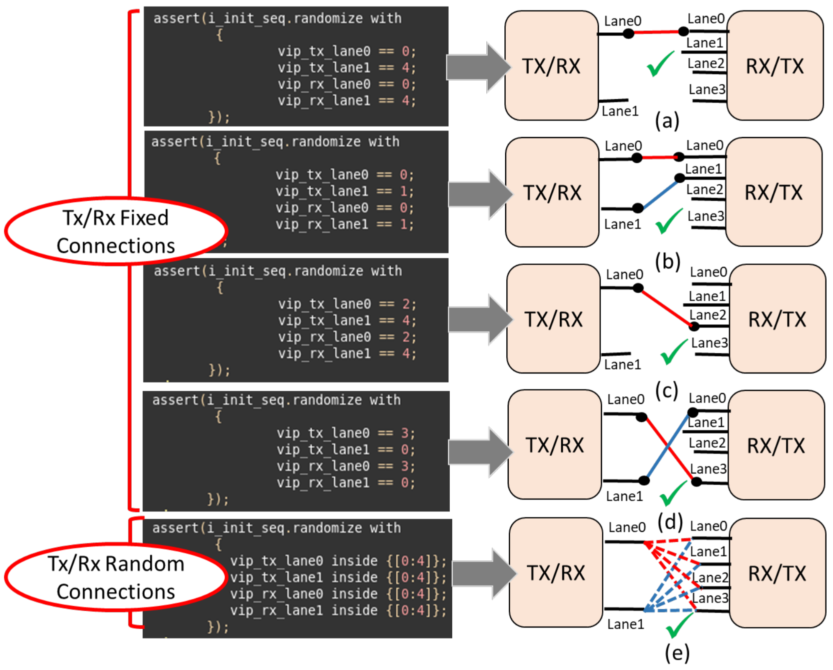


Figure 2. Skew insertion between Lane0 and Lane1 at RMMI for HS-G5 in Rx symbol (Reference: UniPro2.0 Link-PHY interface).

4) *Lane Mapper:* As per MIPI Standards, the Lane Mapping functionality is used to discover the physical lanes on the high-speed link. Most of the times, it is seen that this feature is not tested rigorously as compared to other features and mostly testing is limited to fixed connections.



```
assert(i_init_seq.randomize with
        {
                  vip_tx_lane0 == 0;
                  vip_tx_lane1 == 4;
                  vip_rx_lane0 == 0;
                  vip_rx_lane1 == 4;
        });
```

```
assert(i_init_seq.randomize with
        {
                  vip_tx_lane0 == 0;
                  vip_tx_lane1 == 1;
                  vip_rx_lane0 == 0;
                  vip_rx_lane1 == 1;
```

```
assert(i_init_seq.randomize with
        {
                  vip_tx_lane0 == 2;
                  vip_tx_lane1 == 4;
                  vip_rx_lane0 == 2;
                  vip_rx_lane1 == 4;
        });
```

```
assert(i_init_seq.randomize with
        {
                  vip_tx_lane0 == 3;
                  vip_tx_lane1 == 0;
                  vip_rx_lane0 == 3;
                  vip_rx_lane1 == 0;
        });
```

```
assert(i_init_seq.randomize with
        {
             vip_tx_lane0 inside {[0:4]};
             vip_tx_lane1 inside {[0:4]};
             vip_rx_lane0 inside {[0:4]};
             vip_rx_lane1 inside {[0:4]};
        });
```

Figure 4. Random lane connection example for UniPro 2.0 DUT (2 Lane) and Peer Device (4 Lane) (a) Single-Lane Straight Connection (b) Dual-Lane Straight Connection (c) Single-Lane Cross Connection (d) Dual-Lane Cross Connection (e) Random Lanes Connection

5) *Interrupt Checker:* The interrupt checker component is designed to verify the functionality of interrupt handling within the design. It validates if interrupts are triggered, processed and cleared correctly and timely, which is especially important in systems where timely responses to events are critical.

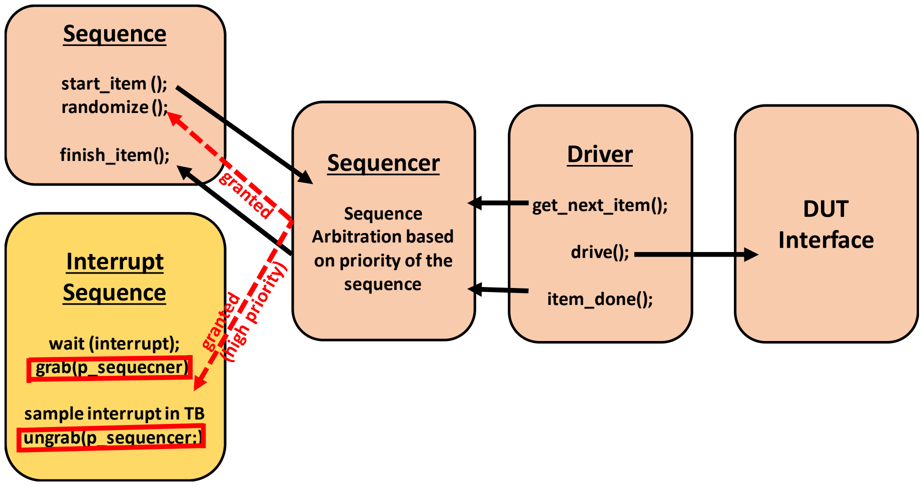


Figure 3. Interrupts handler sequence in UVM based test bench.

6) *Clock UVC*: Clock UVC (Universal Verification Component) is responsible for checking the behavior of clock signals within the design. It ensures that clocks are generated, distributed, and gated correctly, which is crucial for synchronization and proper operation of the design. Automation script is used to dump the clock UVC which generates appropriate clock for the DUT and also check the output clock with respect to different parameter like frequency, phase, division factor, jitter, skew etc.

7) *Assertions for Protocol Compliance:* Assertions are used to verify that the design adheres to specific protocols or standards. In our case, these assertions are employed to ensure that the design follows the specified communication protocols correctly, which is essential for interoperability and data integrity.

Overall, our testbench seems well-rounded, addressing both functional and performance aspects of verification, which is essential for thoroughly testing a design's quality and capability. Incorporating performance analysis into our testbench is valuable because it ensures that our design not only functions correctly but also meets performance requirements.
It demonstrates a comprehensive approach to verification that goes beyond mere functionality to evaluate real-world performance, making it robust and suitable for rigorous testing of your design.

B. Smart Coverage Closure using Scripting

The challenges like the constrained random verification and directed test case verification are effective methodologies but can be time-consuming and demanding in terms of DV engineer effort. These standard methodologies involve creating and running numerous tests to achieve comprehensive coverage, which may not always be feasible within project timelines. As a solutions coverage hole script is introduced which ensures efficient verification processes in a dynamic environment. Let us go a bit in details:

1) *Coverage Hole Perl Script Agent:* To address the challenge of coverage holes and streamline the verification process, the coverage hole script agent plays a crucial role. It helps by:

a) Obtaining the latest coverage database.
b) Extracting information about un-hit coverage bins.
c) Creating a directed test suite to target the remaining uncovered bins with a single test run.

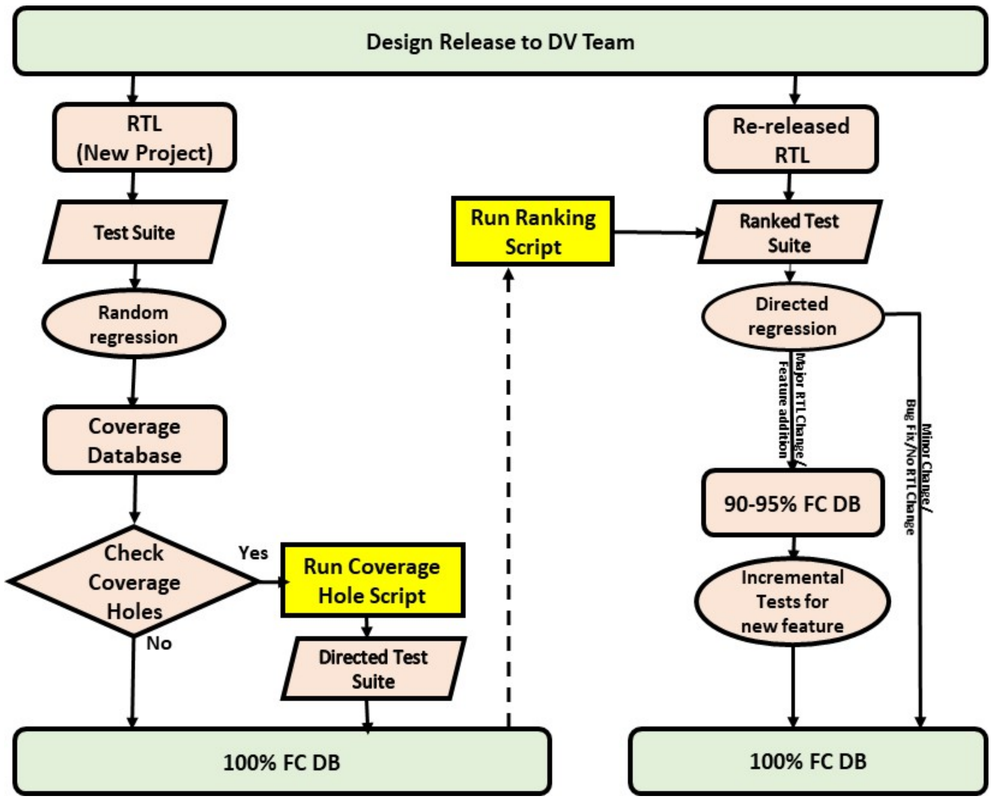


Figure 5. Test bench integrated with Coverage Hole Perl and Ranking script for coverage.

Benefits: We have well-structured process for achieving and maintaining coverage in our RTL verification as shown in Fig 5. By running random regression and then using our coverage hole script to generate directed test suites to target un-hit bins, we can efficiently approach 100% coverage. Also, at times there are few legacy cover groups which may not be needed and impossible to hit so to avoid overworking of script for the invalid bins, there is provision that the user can pass the refine file to script as an input and the script will ignore those bins. Additionally, the ranking script helps prioritize tests based on their contribution to overall coverage and generate a ranked test suite. For minor RTL changes, the ranked test suite can help in achieving nearly 100% coverage without much effort. However, for major changes user may need to write new test to cover the newly added features which is common practice to ensure comprehensive verification.

Limitation: The script has a limitation that it will generate the directed test cases targeting specific un-hit bins only. So, a word of caution here that it should not be taken up as a first step in closing the functional coverage.
To find the hidden bugs and hit corner scenarios, random regression is essential and should not be bypassed. So, in other word, though this script can be run for most of the coverage but it is not recommended as randomness of DV test bench will be lost. This script is more suitable where a big covergroup or cross coverage is targeted by randomizing or constraining one single testcase with multiple inputs.

Coverage Hole Script Working Flow: This script is designed to focus on stimulus coverage, making it easily adaptable to different complex IP's. We have a sequential script process consisting three scripts as shown Fig. 6. Initially, the main script is responsible for extracting essential information from the user's passed input data like project name, coverage model, refine file, coverage database and these user's passed input will be changed project to project. This refines file includes identifying invalid cases for dependent random variables. Additionally, this main script triggers the coverage tool to generate a coverage report, as illustrated in Fig. 9. This coverage report has the details about both hit and un-hit bins. Following this, the second script examines the user's provided inputs, which contain the information about the for which cover group user want to extract un-hit bin information and this script determines which generic random test cases these extracted unhit bins should be passed to. These generic random test cases need to be written according to the targeted cover groups. As evident, described script is very generic and can be used for any stimulus coverage closure for any kind of IP or Sub-system. The required input files and testcases will only change based on the use case scenario.

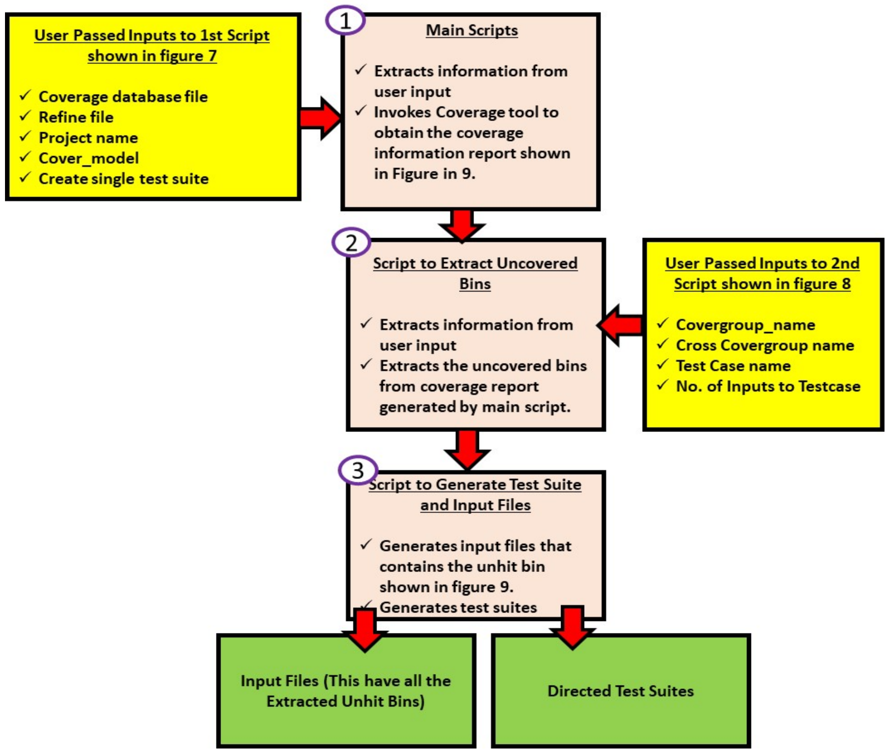
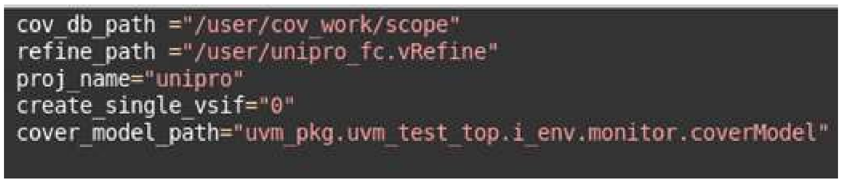

User Passed Inputs to 1st Script shown in figure 7
✓ Coverage database file
✓ Refine file
✓ Project name
✓ Cover_model
✓ Create single test suite
1
Main Scripts
✓ Extracts information from user input
✓ Invokes Coverage tool to obtain the coverage information report shown in Figure in 9.
2
Script to Extract Uncovered Bins
✓ Extracts information from user input
✓ Extracts the uncovered bins from coverage report generated by main script.
User Passed Inputs to 2nd Script shown in figure 8
✓ Covergroup_name
✓ Cross Covergroup name
✓ Test Case name
✓ No. of Inputs to Testcase
3
Script to Generate Test Suite and Input Files
✓ Generates input files that contains the unhit bin shown in figure 9.
✓ Generates test suites
Input Files (This have all the Extracted Unhit Bins)
Directed Test Suites

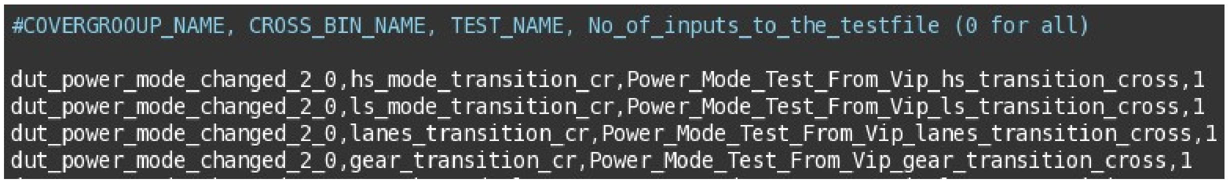
Figure 6. Working Flow of Coverage Hole Script.

Finally, the third script takes the extracted bins for the user-specified cover groups and leverages them to create input files as shown in Fig 9. Also, this script generates the test suites associated with these input files. The purpose of these generated test suites is to specifically target the un-hit bins in script generated inputs files.



```
cov_db_path ="/user/cov_work/scope"
refine_path ="/user/unipro_fc.vRefine"
proj_name="unipro"
create_single_vsif="0"
cover_model_path="uvm_pkg.uvm_test_top.i_env.monitor.coverModel"
```

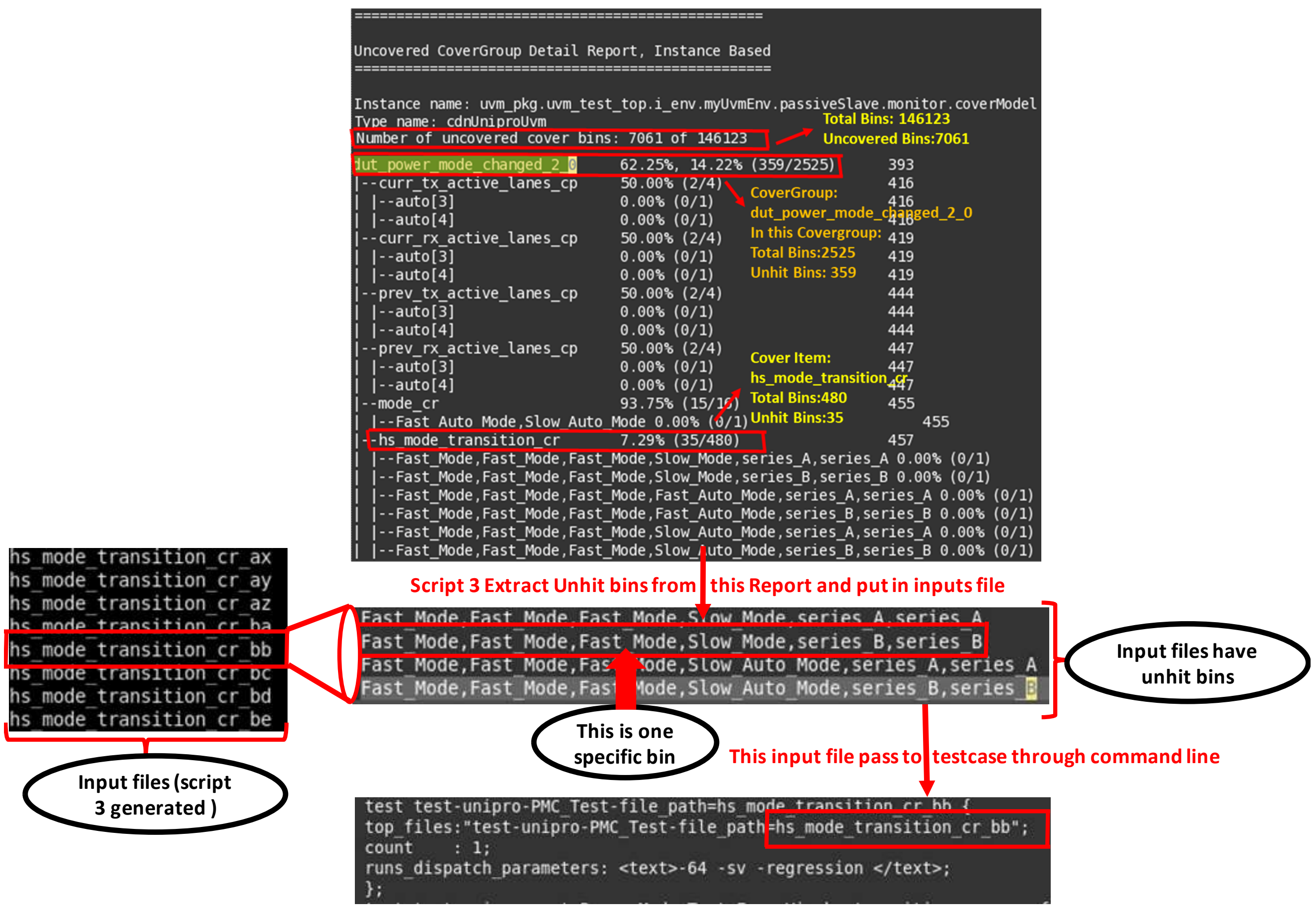
Figure 7. 1st Input File to Script for Main Script.



```
#COVERGROOUP_NAME, CROSS_BIN_NAME, TEST_NAME, No_of_inputs_to_the_testfile (0 for all)

dut_power_mode_changed_2_0,hs_mode_transition_cr,Power_Mode_Test_From_Vip_hs_transition_cross,1
dut_power_mode_changed_2_0,ls_mode_transition_cr,Power_Mode_Test_From_Vip_ls_transition_cross,1
dut_power_mode_changed_2_0,lanes_transition_cr,Power_Mode_Test_From_Vip_lanes_transition_cross,1
dut_power_mode_changed_2_0,gear_transition_cr,Power_Mode_Test_From_Vip_gear_transition_cross,1
```

Figure 8. Input File to the 2nd Script.



```
==================================================

Uncovered CoverGroup Detail Report, Instance Based
==================================================

Instance name: uvm_pkg.uvm_test_top.i_env.myUvmEnv.passiveSlave.monitor.coverModel
Type name: cdnUniproUvm
Number of uncovered cover bins: 7061 of 146123

dut_power_mode_changed_2_0     62.25%, 14.22% (359/2525)       393
|--curr_tx_active_lanes_cp     50.00% (2/4)                    416
|  |--auto[3]                  0.00% (0/1)                     416
|  |--auto[4]                  0.00% (0/1)                     416
|--curr_rx_active_lanes_cp     50.00% (2/4)                    419
|  |--auto[3]                  0.00% (0/1)                     419
|  |--auto[4]                  0.00% (0/1)                     419
|--prev_tx_active_lanes_cp     50.00% (2/4)                    444
|  |--auto[3]                  0.00% (0/1)                     444
|  |--auto[4]                  0.00% (0/1)                     444
|--prev_rx_active_lanes_cp     50.00% (2/4)                    447
|  |--auto[3]                  0.00% (0/1)                     447
|  |--auto[4]                  0.00% (0/1)                     447
|--mode_cr                     93.75% (15/16)                  455
|  |--Fast_Auto_Mode,Slow_Auto_Mode 0.00% (0/1)                    455
|--hs_mode_transition_cr       7.29% (35/480)                  457
|  |--Fast_Mode,Fast_Mode,Fast_Mode,Slow_Mode,series_A,series_A 0.00% (0/1)
|  |--Fast_Mode,Fast_Mode,Fast_Mode,Slow_Mode,series_B,series_B 0.00% (0/1)
|  |--Fast_Mode,Fast_Mode,Fast_Mode,Fast_Auto_Mode,series_A,series_A 0.00% (0/1)
|  |--Fast_Mode,Fast_Mode,Fast_Mode,Fast_Auto_Mode,series_B,series_B 0.00% (0/1)
|  |--Fast_Mode,Fast_Mode,Fast_Mode,Slow_Auto_Mode,series_A,series_A 0.00% (0/1)
|  |--Fast_Mode,Fast_Mode,Fast_Mode,Slow_Auto_Mode,series_B,series_B 0.00% (0/1)
```

```
hs_mode_transition_cr_ax
hs_mode_transition_cr_ay
hs_mode_transition_cr_az
hs_mode_transition_cr_ba
hs_mode_transition_cr_bb
hs_mode_transition_cr_bc
hs_mode_transition_cr_bd
hs_mode_transition_cr_be
```

```
Fast_Mode,Fast_Mode,Fast_Mode,Slow_Mode,series_A,series_A
Fast_Mode,Fast_Mode,Fast_Mode,Slow_Mode,series_B,series_B
Fast_Mode,Fast_Mode,Fast_Mode,Slow_Auto_Mode,series_A,series_A
Fast_Mode,Fast_Mode,Fast_Mode,Slow_Auto_Mode,series_B,series_B
```

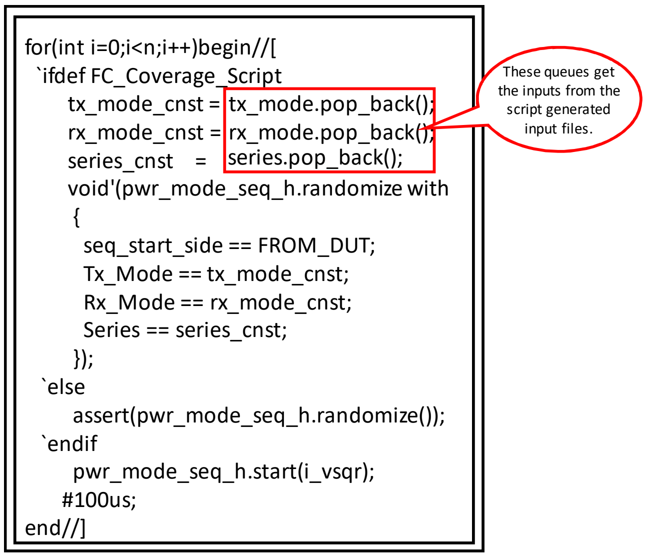
```
test test-unipro-PMC_Test-file_path=hs_mode_transition_cr_bb {
top_files:"test-unipro-PMC_Test-file_path=hs_mode_transition_cr_bb";
count     : 1;
runs_dispatch_parameters: <text>-64 -sv -regression </text>;
};
```

Total Bins: 146123
Uncovered Bins:7061
CoverGroup:
dut_power_mode_changed_2_0
In this Covergroup:
Total Bins:2525
Unhit Bins: 359
Cover Item:
hs_mode_transition_cr
Total Bins:480
Unhit Bins:35
Script 3 Extract Unhit bins from this Report and put in inputs file
Input files have unhit bins
This is one specific bin
This input file pass to testcase through command line
Input files (script 3 generated)

Figure 9. Snaps of Uncovered Covergroup Detail Report, Script generated input file and Test Suite.

Our case study on complex High-Speed Link UniPro IP have around total 146123 bins as shown in Fig 9. The coverage hole script will take approximately 40 to 60 seconds to extract and create directed test suites for these many bins and it can be scaled up easily for more complex IP's or sub-system. Perl is a high-level, general-purpose language which is easy to learn and good choice for short scripting and text processing. Based on the use-case and scripting requirement this looks perfect fit for our need.

Our approach is for test reusability. We are using "define" mechanism to determine variable values in our test cases. When user used "FC_Coverage_Script" define then test variables get the values from the script generated input files and this allow tests to target specific combination of un-hit bins in single run as shown Fig 9. Otherwise, variable values are decided randomly during random regression. This flexibility can save time and effort in testcase development as same testcase, created for random verification, is reused.

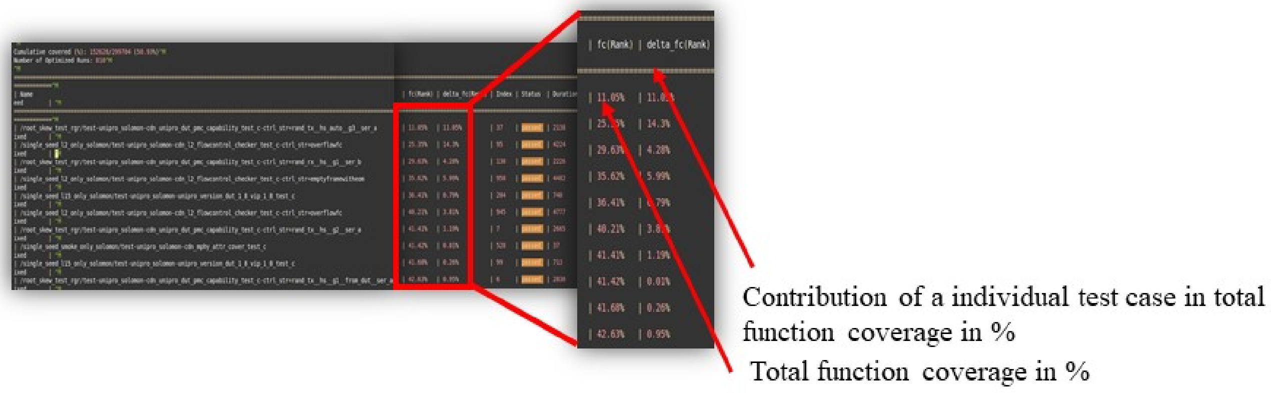


```
for(int i=0;i<n;i++)begin//[
 `ifdef FC_Coverage_Script
     tx_mode_cnst = tx_mode.pop_back();
     rx_mode_cnst = rx_mode.pop_back();
     series_cnst   =  series.pop_back();
     void'(pwr_mode_seq_h.randomize with
      {
        seq_start_side == FROM_DUT;
        Tx_Mode == tx_mode_cnst;
        Rx_Mode == rx_mode_cnst;
        Series == series_cnst;
      });
  `else
      assert(pwr_mode_seq_h.randomize());
  `endif
      pwr_mode_seq_h.start(i_vsqr);
    #100us;
end//]
```

These queues get the inputs from the script generated input files.

Figure 10. Example code of Power Mode Change (PMC) with Script generated Unhit Bins

2) Coverage Ranking Script: Another important script in efficient verification is the coverage ranking script.



```
| fc(Rank) | delta_fc(Rank)

| 11.05%   | 11.05%
| 25.35%   | 14.3%
| 29.63%   | 4.28%
| 35.62%   | 5.99%
| 36.41%   | 0.79%
| 40.21%   | 3.81%
| 41.41%   | 1.19%
| 41.42%   | 0.01%
| 41.68%   | 0.26%
| 42.63%   | 0.95%
```

Contribution of a individual test case in total function coverage in %
Total function coverage in %

Figure 11. Ranking Script output log file

This script:

a) Ranks regression runs based on their contribution to coverage as shown in Fig 11.
b) Creates a test suite that focuses on the specifically on all bins with single run.

c) Proves particularly valuable in scenarios where multiple RTL (Register-Transfer Level) releases occur, and the verification team needs to quickly close coverage gaps as part of iterative development.

In summary, our approach leverages automated scripts and ranking methodologies to expedite the verification process, especially in scenarios with time constraints and multiple RTL releases. These proposed methodologies aim to reduce the burden on verification engineers, streamline the closure of coverage holes, and improve the overall efficiency of the verification process.

II. RESULTS AND CONCLUSIONS

The proposed methodologies and setup help DV engineers in finding the bugs at early stage of the product development. Also, user will be able to close the DV in quick succession with the help of proposed coverage scripts as results shown in Table I and Table II. User will not able to achieve 100% coverage in single run in case of major changes in RTL and hence TB changes, because same seed will not create the same test vector. As coverage hole script stores the test vectors, it reduces the coverage slip as shown in Table II and still able to achieve 90-95% coverage in single run. In addition, the same DV setup can be and scaled up for sub-system verification and reused for other high speed link verification with minimal effort.

Table I. Comparison of Number of Test Runs with and without Script (Reference: UniPro2.0 DV)

Approximate Number of Test Runs to Achieve 100 % FC Coverage			
Method	Random Regression	After Adding PMC Script	After Running Ranking Script
Total Test Run	20000-25000	8000-10000	3000-3500

Table II. Comparison of Coverage Replay with Beta Test Suite (Reference: UniPro2.0 DV)

Approximate Coverage Replay with Beta Test Suite regression (New RTL release)		
Method of generation	Ranking Script (only)	Ranking + Coverage Hole Script
FC (%)	80-87%	90-95%

REFERENCES

[1] "MIPI Alliance DRAFT Specification for Unified Protocol, version 2.0 r10", mipi.org. https://members.mipi.org/wg/UniPro/document/folder/14599 (accessed Jan 17, 2022).

[2] "MIPI Alliance DRAFT Specification for M-PHY v5.0 r06," mipi.org. https://members.mipi.org/wg/M-PHY/document/folder/14115 (accessed Jan 24, 2021