Acceleration of verification closure for “High Speed Complex Serial IO Interconnect” using UVM verification methodology

Kumuda T1, Deepthi S2
Associate Professor1, M.TECH2.
Dept. of Electronics & Communication EngineeringDept. of Electronics & Communication Engineering
Adichunchanagiri Institute of TechnologyAdichunchanagiri Institute of Technology
Chikmagalur, IndiaChikmagalur, India

Abstract-Technology developments permits for the construction of bigger and more complex designs. This positions new challenges, together with hard work to balance verification completeness with reducing overall verification effort and CPU cycle time. It is basically impossible to count all of the states and conditions to do a thorough test. Hence, it is imperative to use well-defined principles to measure and test when the verification is satisfactorily complete and come across a sensible quality threshold. Code coverage is one of the quality measure of design. One more is functional coverage, which is used to check that all essential features of the design are tested.

In this paper, we have designed a coverage for “High Speed Complex Serial IO Interconnect”. High speed complex serial IO interconnect IP is interconnect between CPU which uses serial protocol and operates at high speed in terms of GHZ. To guarantee the completeness of the verification of the correct functional behavior of “High Speed Complex Serial IO Interconnect”, we developed the functional coverage models by analyzing the specification of it and enabled the code coverage. Integrated the function coverage model into UVM based verification environment and through regression generated the coverage reports and analyzed the holes. Coverage metric and analyzed holes directs the verification engineer to achieve verification completeness by providing useful information about missing situations. Generally, our metric allows to find the verification progress and suggestively help the verification engineers in completing the verification. This project presents the design and implementation of coverage for “High Speed Complex Serial IO Interconnect”.

Keywords- Functional coverage, code coverage, High Speed Complex Serial IO Interconnect, verification, UVM, coverage models, functional behavior, test bench, design.

I. INTRODUCTION

High speed complex serial IO interconnect is a next generation cache-coherent, link based interconnect. This architecture can be used in a wide variety of server platform configurations. This interconnect also provides support for high-performance I/O devices which may be connected via standard I/O buses such as PCI express, through an appropriate bridge.

Due to the rapid increasing complexity of modern circuits, functional verification has been the main bottleneck in the design cycle. It has been reported that more than 70% of development time is occupied by verification [1], and this portion is still growing. Therefore, effective verification methodologies and techniques are essential. Constrained random tests are applied with different seeds to validate most design behaviors. If some scenarios are hard to be reached for them, direct tests are applied, or new constraints may be added.

Every design verification technique requires coverage metrics to gauge progress, assess effectiveness, and help determine when the design is robust enough for tape out. At every step of the way and with every bug finding technology and tool, verification engineers assess coverage results and make critical decisions on what to do next.

In fact, for the verification of large, complex system on chip (SoC) designs, coverage metrics and the responses to them guide the entire flow. The term "coverage driven verification" describes a methodology built around coverage metrics as the primary way to manage verification.

In CRV, the each point in test plan is generated automatically. As there points are generated automatically, we need a mechanism which tells us that all the points in test plan are not exercised. When all the points in test plans are verified and the code coverage is 100% we can stop verification.

In Directed verification, there will be a separate test case file for each feature to be verified. So to know how many features are verified, count the test cases. Verification is done when all tests are coded and passing alone with 100% code coverage. In constraint random verification all the features are generated randomly. Verification engineer need a mechanism to know the information about the verified features of DUT.

System Verilog provides a mechanism to know the untested feature using functional coverage. Functional Coverage is "instrumentation" that is manually added to the Test bench. This is a better approach then counting test cases. Functional coverage is better than code coverage where the code coverage reports what was exercised rather that what was tested. Through the functional coverage and a combination of code coverage analysis verified IP can improve the efficiency and reliability

II. COVERAGE OVERVIEW

Coverage is used as a guide for directing verification resources by identifying tested and untested portions of the design. Coverage is defined as the percentage of verification objectives that have been met. It is used as a metric for evaluating the progress of a verification project in order to reduce the number of simulation cycles spent in verifying a design.Two types of coverage metrics:

- code coverage: automatically extracted from the design code
- Functional coverage: measures how much of the design specification, have been observed, validated, and tested

A. code coverage

To check whether the Test bench has satisfactory exercised the design or not? Coverage is used. It will measure the efficiency of our verification implementation.Code coverage answers the questions like

- Have all the 'lines' of the DUT has been exercised?
- Have all the 'states in the FSM' has been entered?
- Have all the paths within a 'block' have been exercised?
- Have all the branches in 'Case' have been entered?
- Have all the conditions in an 'if' statement is simulated?

With the above information, verification engineer can plan for more test cases and excursive uncovered areas to find bugs. Code coverage specifies that how much deep level the design is checked.

A. Limitations of code coverage

Coverage does not know anything about what design supposed to do. There is no way to find what is missing in the code. It can only tell quality of the implementation. Sometime we get the bug because of the incorrectly written RTL code. If we found that all the lines of the code are used, it doesn't mean that we have tasted all the lines. Sometimes we want the 2nd input of the mux but due to mistake in stimulus if it has taken 1st during that cycle. So whether we got the correct data or not? This cannot tell by coverage. That's depend on us weather we are feeding correct stimulus or not? So "Verification Is Not Completed Even After 100% Code Coverage"

B. Functional coverage

It works on the functional part of the stimuli's implementation. Functional coverage will check the overall functionality of the implementation. Verilog does not support functional coverage. To do functional coverage, Hardware verification languages like System Verilog, Spec man E or Vera are needed.

System Verilog functional coverage features:

- Coverage of variables and expressions, as well as cross coverage between them
- Automatic as well as user-defined coverage bins
- Associate bins with sets of values, transitions, or cross products
- Filtering conditions at multiple levels
- Events and sequences to automatically trigger coverage sampling
- Procedural activation and query of coverage
- Optional directives to control and regulate coverage.

III. HIGH SPEED COMPLEX SERIAL IO INTERCONNECT

High speed complex serial IO interconnect IP is interconnect between CPU which uses serial protocol and operates at high speed in terms of GHZ. It is a next generation cache-coherent link based interconnect for server platforms. Interconnect also provides support for high performance I/O devices which may be connected via buses through an appropriate bridge. This links can be used to interconnect various numbers of processors in various ways such as rings, star or various mesh topologies like 2x, 4x, 8x, 16x processors.

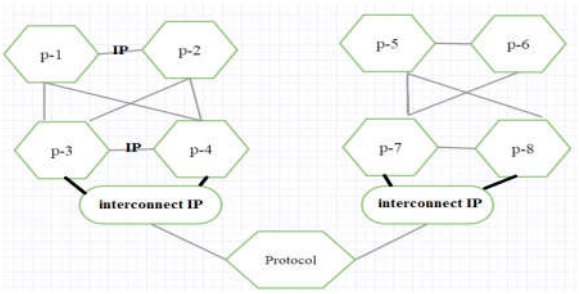

Fig. 1. Interconnect IP connecting 4x processors

The functionality of interconnect is partitioned into four layers.Each layer performs a well-defined set of non-overlapping functions. This layering allows for easier future upgrades to the interface by allowing fairly independent optimizations at each layer.

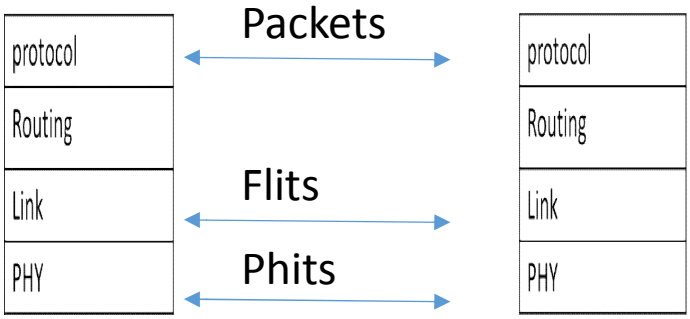

Fig. 2.Four layers of interconnect IP

These communicate with each other at the protocol layer. Each layer in the interface deals with a specific quantum of information. The data transfer unit at the physical layer is called a phit (physical unit). The link layer communicates at a higher granularity called a flit (flow control unit) which is the smallest granularity for flow control. The protocol and routing layers communicate at the granularity of a packet.

A. Physical layer

The physical layer is responsible for the fat transfer of information on the physical medium (electrical or optical or …). The physical layer is the point to point between two link layer entities.It supports:

- Multiple types of signaling like low voltage differential signaling.
- Multiple types of physical medium.
- Variable link widths.
- Link power management state transitions.
- Lane reversal and polarity inversion.

B. Link layer

The link layer abstracts the physical layer from the upper layer and provides the capability to reliably transfer data and manage flow control between two directly connected interconnect entities. To provide reliable transmission CRC error checking and recovery procedures are provided by the link layer. It checks the CRC by generating CRC at the TX and checking at RX. Flow control checks whether the information is credited at TX and depicted at RX properly.

C. Protocol layer

Coherent protocol (reads, writes, invalidations), ordering of writes and maintaining coherency across n controller protocols.

- Non-coherent protocol: peer-to-peer I/O, interrupt delivery, and other higher-level common protocol.
- Memory protocol: uses a link and physical layer combination.

D. Routing layer

This layer provides a flexible and distributed way to route packets from a source to a destination.

IV. COVERAGE DRIVEN VERIFICATION METHODOLOGY

Verification planning and super vision includes recognizing the features of the design under test that must to be tested, prioritizing those features, determining progress, and correcting the allocation of verification assets so that on the necessary timescale verification closure can be grasped.

Code coverage and functional coverage measure different things. Execution of the actual RTL code is measured by the Code coverage. (Therefore before code coverage can be run at all RTL code must be present). The collection of code coverage figures is generally automatic, which containing state coverage, statement and branch coverage, and state transition coverage. Code coverage can be deliberated as a quantitative measure of design under test code performance. On the other hand, functional coverage, efforts to measure whether the features defined in the verification plan have really been implemented by the design under test. To create the verification plan, I have to come up with the features to be measured from the design's specification and implementation, and therefore functional coverage can be deliberated as a qualitative measure of design under test code performance. The number of features to be hit by functional coverage depends upon the carefulness and thoroughness of the users who draw up the verification plan.

For example when the design is partial complete it is somewhat possible to achieve 100% code coverage but only 50% functional coverage. Equally, it is possible to have 50% code coverage but 100% functional coverage, which might point out that some important features of the design is missed by functional coverage model. The two coverage methodologies are complementary, and both of them will benefit high quality verification.

As opposite to a list of directed test explanations it is good practice to create a verification plan that contains a list of features to be tested. All participants in the verification procedure should donate to the documentation and ordering of features in the verification plan, since for the successive verification development this feature set will form the groundwork. From the verification plan if some of the features are missing this would strictly damage verification quality. Original design requirement or the RTL operation should be cross referenced to the features planned in the verification plan. Features must be recognized as being task - dangerous, most important, less important, elective, stretch goals, and so forth, and must be place in order. This benefits with requirements traceability and makes it relaxed to measure verification progress in contradiction of comprehensive product goals, such as the talent to differentiate between the verification of task - dangerous features against less important features. To collect coverage statistics, this verification plan is then realized by converting each feature into code or else known as the coverage model. This can be a mixture of sample - based coverage (System Verilog cover groups), code coverage, and property based coverage (System Verilog cover property statements).

Coverage driven verification have need an important alteration in outlook and practice when related to direct testing. To exercise precise features instead of writing tests, the features to be verified are completely enumerated in the coverage model, and tests help only to direct the constrained random stimulus generation on the way to filling any coverage holes. To find the close relationship between the original design specification and the output from the functional verification process coverage is used as the measure of fruitful execution of the verification plan. To makes it easier to respond to specification changes this relationship improves requirements traceability.

Actual verification planning and supervision depends on a combination of code coverage, functional coverage, and automated checking. Agent specific functional coverage, carried out within subscribers linked to separate agents, can gather coverage data for distinct interfaces. From numerous agents coverage collector mechanisms that accept transactions can cross data from several DUT interfaces. In both case, coverage data gathering can be activated by the appearance of incoming transactions at scoreboard and subscriber mechanisms. Before coverage collection starts the coverage model must be strictly reviewed in contrast to the verification plan and the verification plan strictly reviewed in contrast to the design specification. When the design works properly then only a feature can be considered and covered: failed tests must not contribute in the direction of increasing coverage record. For the coverage driven verification methodology Self-checking verification environments are dangerous. Checks can be performed at all stages in the verification environment containing end-to-end checking in scoreboards, assertions in System Verilog interfaces, and checks integrated with coverage collection.

Before starting to collect functional coverage information code for the stimulus generation, drivers, monitors, the checkers and the coverage model must be complete and debugged: coverage information measured in the nonappearance of complete checking is meaningless. Until the coverage goals are met the same verification environment is executed repetitively using different tests during the verification process.

One test can be different from another test just by starting simulation with a dissimilar random seed or by constraining or arranging the verification environment in a different manner. Tests can be classified according to a various different principles together with simulation run time, functional coverage, and bugs detected. The best performing tests, rendering to some arrangement of these principles, can then be used as the foundation for the regression test suite:

I wanted to select a set of tests that helps us to attain the maximum coverage in the least possible run time, and any tests that do not enhance to the functional coverage attained by the top set of tests are discarded. It is potential to reach at a highly improved regression test suite in the direction of the end of the verification process. It is common to run many tests in parallel on compute server farms. The coverage results from isolated tests essential to be merged together and marked up back on top of the verification plan to form an aggregate record of progress. In order to make available up-to-date statistics for verification management this needs to be done efficiently. From the merge failing tests should be excluded for coverage results. From each test run satisfactory information must be captured to permit the simulation run to be replicated. This information will contain all significant software version numbers, the random seed, and the revision identifiers of the verification code and the RTL code.

For a number of reasons features in the verification plan can stay uncovered, even after widespread random testing, and the reasons essential to be analyzed. Features may possibly be unused or disabled in the product, unimplemented, unintentionally forgotten about, or impartially very tough to reach with random tests. After analysis, further specific tests are added to reach features that still remain uncovered. Where in practical, rather than resorting to very precise directed tests it is desirable to regulate the handles used in the constraints to make tighter up random distributions.

Functional coverage helps to recognize which features in the verification plan have been verified successfully? Which features in the verification plan until now have not been verified and thus need additional work? What percentage of the features have been verified and therefore to completion how close the verification process is?

Assurance that until that time fixed design bugs have not introduce again into the design. This can be distinguished with a traditional directed testing methodology in which the nonexistence of more bugs being discovered is taken as confirmation that verification is close to completion. This type of methodology can result in an over-confident assessment of the correct state of the verification process.

The essential steps in the coverage driven verification process are as follows:

- Create the verification plan with the involvement of stakeholders
- Create the coverage model from the verification plan
- Debug the verification environment, checkers, and coverage model
- Run tests with multiple random seeds until cumulative coverage flattens off
- Annotate coverage results back onto the verification plan
- Run further tests with modified stimulus constraints to close coverage holes
- Analyze and prioritize any unverified features and allocate resources accordingly
- Run directed tests for particularly hard to reach coverage holes.

V. IMPLEMENTATION

A. Interconnect IP

Interconnect IP has PHY layer, higher layers of the processor for connection, two register interfaces, debug interfaces and power management and clock interface.

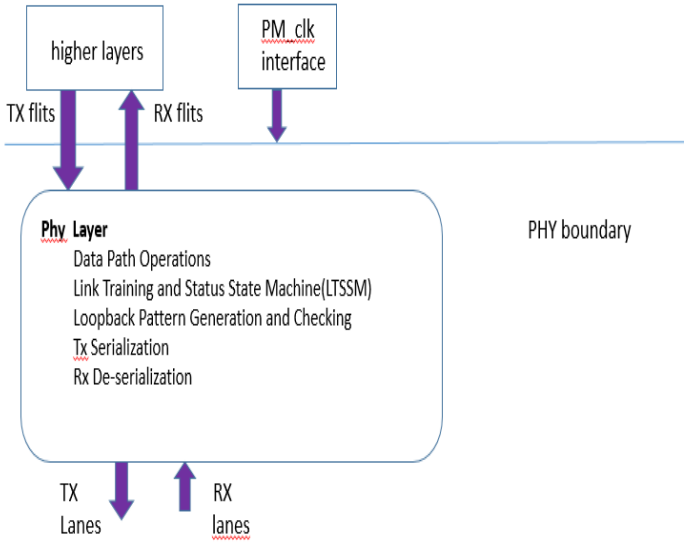

Fig. 3. Interconnect IP

B. PHY layer

PHY layer has data path operations, link training and status state machine, Loopback pattern generation and checking, TX serialization, RX De-serialization, control and status registers.

Configuration interface 1:The configuration interface-1 is the primary interface for configuration and control. This interface has separate address, data, byte enables, type, and Rd/Wr control signals to control read and write access to the associated CSR's.

Configuration interface 2:This is a write-only interface. It is to configure the common lane logic. It is a bus that writes directly to registers that are critical to the operation of the PHY. In this interface, address and data are multiplexed on the same bus. It is used to check the add-valid and the multiplexed bus.

Higher layers:Link layer interface sends the data to the physical layer. LTSSM controls the process of link bring-up. All handshake and training patterns required for link bring-up are handled by the LTSSM itself without any dependency from the link layer.

Debug interface:This interface will provide the access to internal digital signals and corresponding clock during debug on silicon.

PM_CLK interface:Power management and clock control interface provide powering on the PHY and resetting the PHY and also generate the clock required for PHY to operate.

C. Functional coverage for interconnect IP

To create the coverage model I created the coverage plan by looking interconnect IP spec and verification plan. Coverage plan includes all stimulus, scenarios, cover group definitions, and cover group name, cover point name, and estimated no. of bins that should be required to check the quality of the stimulus and verification status.

Coverage plan and coverage model:Coverage plan for interconnect IP is done based on functionality of each interface by looking at the spec.

Register coverage: I had put plan for 33 registers. Each register contains so many fields. I made each register as cover group and fields as cover points. Cover group definition contains cover points and cover points legal value and illegal value as its bins. By looking the coverage plan I coded the coverage model.

Link interface coverage: for this interface I had put plan for link operation with different modes, transition from one mode to other mode with link operation, Link initiated low power entry and exit scenarios, link operation in high and low power mode, transition from high power mode to low power mode with link operation, asymmetry of high power mode and low power mode, reset entry and exit in high power mode and low power mode, register initiated reset entry and exit in high power mode and low power mode, reset entry during initialization, all the FSM state transition during TX and Rx.

Debug interface: All the LTSSM tracker registers, status registers, condition registers, lane reversal, polarity inversion, TX lane disable, and RX lane disable, failure during entry and exit scenarios.

Configuration interface: all possible address, data, types, read and write, back to back read, write, read to write, write to read scenarios for the registers.

PM_CLK interface: power operation with all possible frequencies in different modes, all possible frequency transition, different types of reset entry during initialization and with reset high power mode and low power mode operation, and cross between resets.

Test logic coverage: to check IP on test chip we need to enable the test chip mode for link interface coverage along with normal mode. Check for no CRC errors both at TX and RX, request and acknowledge for CRC error check scenario at TX and Rx, supported flits and unsupported flits when CRC check enabled.

By looking the coverage plan I coded the coverage model for both normal mode and test chip mode. Then integrated the function coverage model into UVM based verification environment of high speed serial IO interconnect. Through regressions generated the functional coverage and the code coverage report for high speed serial IO interconnect.

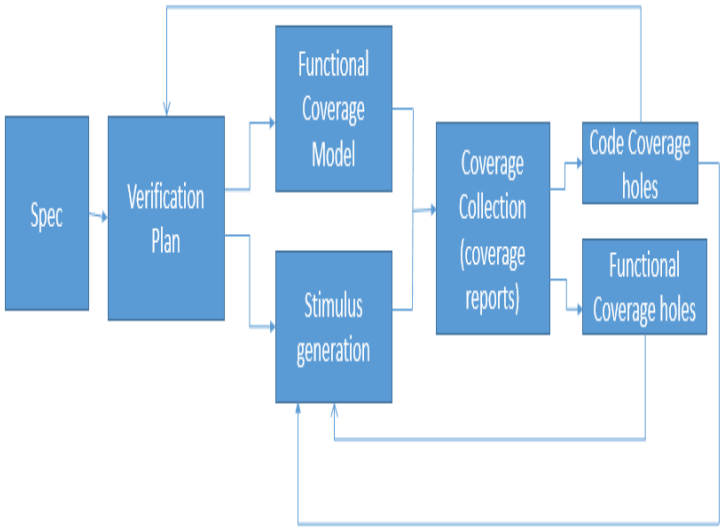

Fig. 4 Flow of coverage

VI. RESULT

For functional coverage I coded the coverage model but for code coverage I just enabled the coverage in regression script during simulation so that coverage will automatically extracted from the design code.

A. Analyze of Functional coverage report

Once we got the code and functional coverage report we need to analyze the report. Analyzing the report is very easy because report is so clear that by seeing the report we will come to know which stimulus, scenarios are not covered. In the fig 4.5the functional coverage report if we see the cover group reports it has different cover groups and their respective score.

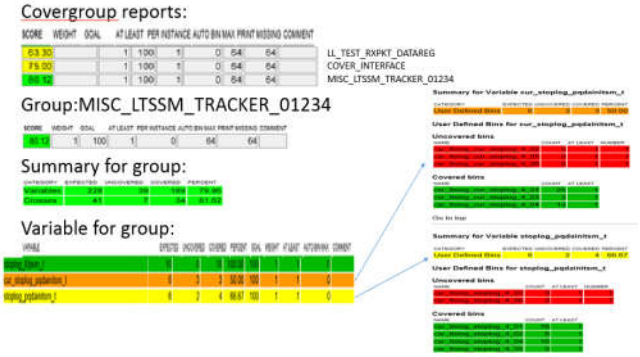

Fig. 5 analysis of functional coverage report

If take one cover group to analyze as shown in fig MISC_LTSSM_TRACKER_01234 it contains summary for group in that we can see how many variables and crosses are expected, covered and uncovered and their percentage. If we see the variable for group in fig 5 we can see the variable name i.e. cover point and their expected, covered, uncovered and percentage and color also differentiates their percentage. If we open one of the cover point we can see the covered bins and uncovered bins. Now we will get the clear picture of covered and uncovered one.

Now we can increase the coverage no. by

- Increasing the randomization seeds.
- Adding the missing test scenarios.
- Adding test cases.
-

A. Analyze of Code coverage report

In the fig. 6 we can see the line coverage report and branch coverage report. We can analyze the line coverage report by looking the red color lines which indicates not covered. Branch coverage report mentions branches, line no., total, covered, percentage. If we go to that line no we can see what are covered and what are not covered.

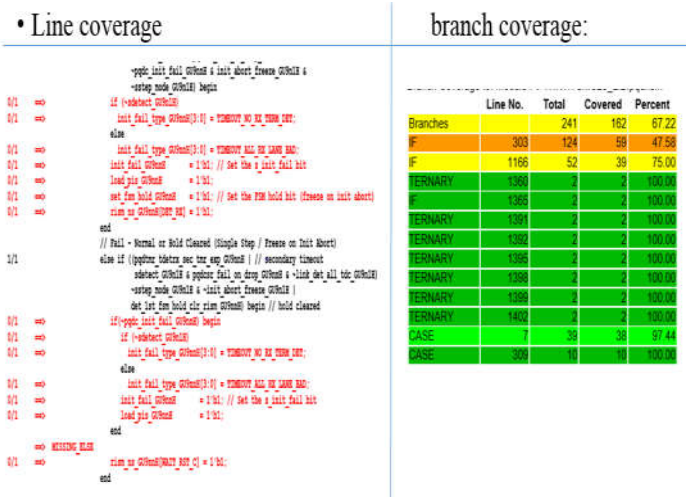

Fig. 6 analysis of code coverage report

I analyzed the holes and converted them into understandable format by verification engineers and designers for reviews. And based on review and feedback added exclusions to increase the coverage number.

B. Functional and code coverage report

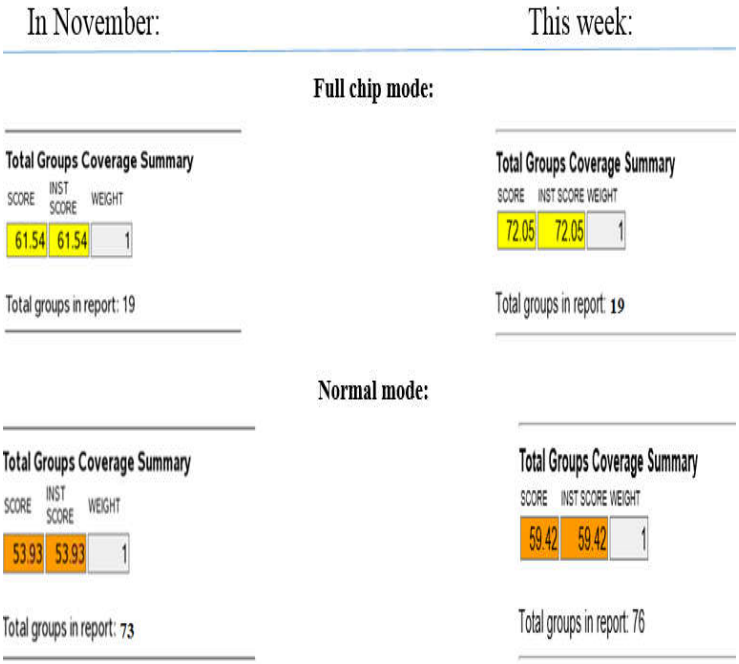

Fig. 7 functional coverage report

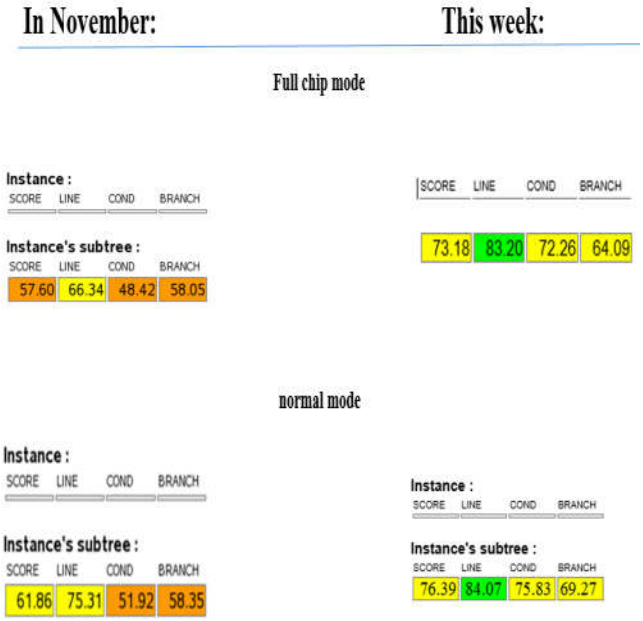

Fig 8 code coverage report

As shown in fig 7 and 8 there are two reports for each case as in November and this week. Because when we coded and enabled coverage we got report i.e. named as in November. After analyzing the report and updating the verification plan and stimulus we got another report named as this week. By seeing the report we can see the improvement in coverage this speeds up the verification closure for high speed complex serial IO interconnect.

C. Impact of coverage

After coding the coverage model: (coverage % was 61.54)

- We got to know how good our verification is.
- How much validation plan was covered?
- What are the holes in verification?

After analyzing the holes:

- Identified the exclusions and added exclusion file.
- Increased the randomization seeds.
- Added the missing test scenarios.
- Now the coverage % is 72.05

VII. CONCLUSION

Functional verification, in design automation, is the task of verifying that the software design conforms to specification. This is a complex task that consumes the majority of time and effort in most large system design projects. Several methodologies have been developed lately in order to tackle the functional verification problem such as simulation based verification, assertion based verification, and formal verification. In this paper, we proposed an approach of coverage driven verification for high speed complex serial IO interconnect IP's verification using UVM based methodology. Time for verification is reduced and this speeds up the verification closure by getting expected code and functional coverage reports.

REFERENCES

[1]. Jan Langer and Ulrich Heinkel, Vasco Jerinic and Dietmar Muller "Improved Coverage Driven Verification and Corner Case Analysis usingDecision Diagrams" 0-7803-9740 @2006 IEEE
[2]. Yang Guo, Wanxia Qu, Tun Li, Sikun Li "Coverage Driven Test Generation Framework for RTL Functional Verification" 978 -1-4244-1579-3/07/ © 2007 IEEE
[3]. Finn Haedicke, Daniel Große, Rolf Drechsler "A Guiding Coverage Metric for Formal Verification" 978-3-9810801-8-6/DATE12/ c 2012 IEEE
[4]. Zhang Ying, Wu Ning, Ke Xiazhi, Ge Fen "A Coverage-driven Verification Platform for Evaluating NoC Performance and Test Structure" 978-0-7695-4652-0/12 © 2012 IEEE
[5]. Wang Jiawen, Liu Zhigui, Wang Suliang, Liu Yang, Li Yufei, Yang Hao "Coverage-Directed Stimulus Generation Using a Genetic Algorithm" 978-1-4799-1142-4/13/ 2013 IEEE
[6]. High speed complex serial IO interconnect specification.
[7]. IEEE Standard for System Verilog— Unified Hardware Design, Specification, and Verification Language.
[8]. Universal Verification Methodology (UVM) 1.1 User's Guide.