International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-3, January 2019

Verification of JESD204BTX Soft IP using Universal Verification Methodology

Vinay P, Shylashree N

Abstract— JESD204B transmitter is a part of the serialized data interface between logic devices and data converters bases on the JESD204B standards. The organization, where this project is currently executed, is currently developing the JESD204B Tx and Rx IP for the avionics spacecraft applications where the fail proof and function safe and reliable data communication is essential. The verification of this IP is an important phase in the development wherein it is extremely important to perform rigorous tests on the design to confirm its acclaimed functionality and performance. The verification of an IP of this complexity is done in a systematic and efficient way using the Universal Verification Methodology which is basically constructed using the SystemVerilog. A verification environment is built using the UVM to verify the functionality of the IP. The test cases are written to verify each functionality of the design and the randomized stimuli are applied to cover all the possible input scenarios. The code coverage and the functional coverage is determined and further stimuli are applied to achieve the target coverage. The verification of the JESD204B Transmitter IP is completed with a functional coverage of around 39.17% for each test instance and an overall functional coverage of 100% and code coverage of 94.25%. The verification environment can be reused with minor changes to verify the JESD204B Receiver IP.

Key words: Coverage, IP verification, JESD204B, Universal Verification Methodology.

1. INTRODUCTION

All the electronics systems need to interact with the external world at some point which is analog in nature. As the speed of modern electronic systems has increased rapidly the sampling rate and thus the demand for high conversion rates has led to the advent of high speed ADC and DACs. Along with the speed, the need for higher resolution increases the bits per sample, and hence more bits of data is generated. This poses challenges in the interfacing of the data converters with the other devices as the interface must support data exchange at a very high speed.

JESD204 serialized data interface offers the solution. The JEDEC Solid State Technology Association came up with the JEDEC standards on serial interface for data converters which was revised twice to arrive at the current version, JESD204B. This standard aims to reduce the number of connections between the data converters and logic devices with increased speed and hence presently most of the data converters are coming up with support for JESD204B interfacing. JESD204B offers reliable increase in the data transfer bandwidth between a converter and a logic device such as FPGAs or ASICs [1].

A major portion of the development time is spent on the verification of the designs. It becomes important that the products are free from faults when it is delivered to the customer. Detection of faults at an early stage of development is very important to avoid the risks of project delay and cost overrun.

Verification of IP needs a detailed checking of the DUT functionality with using the coverage and constrained random stimuli generation techniques. This requires a robust testbench that has dedicated components such as scoreboard and monitors to perform the verification tasks. Hence the use of UVM's standards help to adopt the best practices in the industry to make the verification more efficient, effective and reusable [2] . In this work the JESD204B transmitter soft IP design is tested for its functionality using the Universal Verification Methodology. After the specifications of the design is ready, a test plan is created based on the complexity in functionality of the design and the inputs. The test plan would have the overview of the planned architecture of the verification environment and the test scenarios or the test cases to be considered.

The components of the verification environment is developed by extending the uvm base classes present in the library viz. uvm_test, uvm_env, uvm_scoreboard, uvm_agent, uvm_monitor, uvm_driver, uvm_sequencer, uvm_sequence, uvm_sequence_item, uvm_transaction, uvm_components, uvm_report_objects, uvm_phase, uvm_configuration, uvm_objects [3]. The signals to the design under test (DUT) is declared as items in the sequence items class extended from the uvm_sequence_item. The signals may be declared as random signals wherever randomization is necessary for the stimuli generation. The sequence class which extends from the uvm_sequence defines the sequence for the stimuli generation. The sequencer extends the uvm_sequencer and performs the task of supplying the items to the driver from the sequence. The driver which extends the uvm_driver class applies the stimuli to the DUT. The monitor extends from the uvm_monitor and extracts the signals around the DUT to check the functional coverage and/or protocol checking. It forwards the signal data to the scoreboard for the evaluation of the DUT response. The driver and the monitor together forms the agent and it extends from uvm_agent. The scoreboard extends the uvm_scoreboard class and performs the evaluation of the DUT responses to determine if there are any failures in the functionality. The environment extends

Manuscript published on 30 January 2019.
***Correspondence Author(s)**
Vinay P, 4thSem M Tech Student, VLSI Design & Embedded System, Rashtreeya Vidyalaya College of Engineering, Bengaluru, Karnataka, India
Shylashree N, Associate Professor, Dept of ECE, Rashtreeya Vidyalaya College of Engineering, Bengaluru, Karnataka, India

from the uvm_env and it encloses all the discussed components.

The test extends from the uvm_test and contains the test cases where each test case refers to the sequence of the stimuli to be generated [4-5].

The code coverage and functional coverages are determined and more constrained stimuli is supplied to improve the coverage. The verification is completed when the target coverage is achieved.

In Section II the JESD204B standard and the JESD204B transmiter is discussed in brief. Section III presents the detailed methodology adopted for the verification. Section IV presents the Results and Section V concludes the paper.

II. JESD204B

Traditional I/O technologies which use CMOS and LVDS exhibit considerable problems in the design of high pin count interface chips as well as PCBs. During the PCB design, numerous difficulties arise while dealing with the electrical noise and improving the interconnection densities. The conventional techniques for interfacing logic devices with the data converters involve the use of Low voltage differential signalling (LVDS) methods where they use lower voltage swing for transmission of data at higher speed up to 1 Gbps. Due to the increasing demand for larger number of data converter channels with higher speed transceivers for use in applications such as wireless infrastructures, JEDEC Solid State Technology Association came up with the JESD204 standard for serial interface with data converters[1].

The JEDEC standards organization has revised the JESD204 standards two more times after its first release. The specifications for the first version of JESD204 were released in 2006, which supported maximum speed up to 3.125Gbps. The specifications for the second version were released in 2008, with the name JESD204A. This version supported multi lane with sync. The subsequent release is the JESD204B in 2011, where it can achieve lane speeds of 12.5Gbps with determinable latency and frame clocking [6].

The JESD204 standard avoids the transmission of clocks for frame and data with the use of 8b to 10b encoding. Hence communication can attain greater speeds using single pair of lines. This protocol is synchronous by itself. This helps to have shorter PCB wires without worry about the clock skew. It also allows data lanes to be aligned and synced using time reference thus enabling high bandwidth channel for data converters. The enhancements explained before will greatly help numerous applications. Specifically some of the important applications that include wireless infrastructures made up of transceivers. Additional to it, the enhancement will help the applications that have synchronization issues thus enabling the use of high number of ADCs which is common in medicinal applications. JESD204B is best suited for upcoming applications which require higher data rate speed, deterministic latency and avoids the use of frame clock by utilizing the device clock [1].

Converter samples are initially associated with frames and multi-frames and then will be received or sent to a SERDES. Fig.1 shows the data framing. It is possible to transfer multiple samples in a single frame cycle. The number of samples(S) for every converter in one frame cycle must always be an integer. A sample consists of N bits which collectively hold N data bits that are followed by control and tail bits which are optional. It [6] requires that the number of octets in a single lane in one frame cycle must be a whole number. Hence extra tail bits might be attached to the frame.

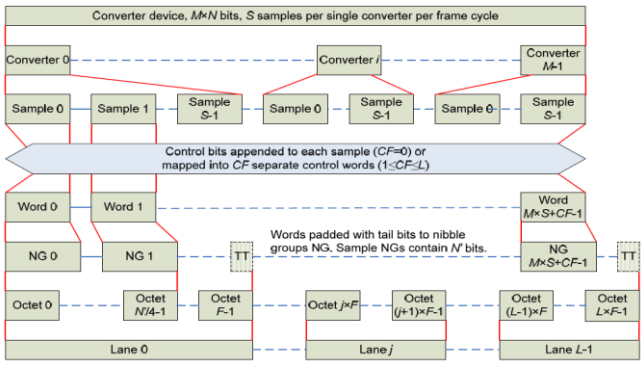

Fig. 1: JESD204B Data Framing [6]

The lacuna in earlier versions of JESD204B to have deterministic latency is filled by the current version of JESD204B. There are three subclasses in the specifications of JESD204B standard. Subclass 0 allows the device to be backward compatible with the JESD204A standard. It offers no support for deterministic latency. Subclass 1 offers support for deterministic latency using SYSREF signaling. Sub-class 2 is similar to sub-class 1 except it offers support for deterministic latency using SYNC sampling [6].

Synchronization with more than one device can be achieved with the use of the SYNCN signal. Every JESD204B receiver indicates the synchronization by sending the SYNCN signal. All such SYNCN signals are collectively considered to determine the readiness the receivers to receive the data. The earlier versions of the JESD204B needed a frame clock to be applied to all the devices. In JESD204B every device can have different clock frequencies derived from the same clock generator. The JESD204B gets its flexibility to setup the serial link due to the use of configuration parameters such as the number of converter devices, resolution of the sample and number of lanes [6].

The I/O signals of the JESD204B transmitter along with the width of each signal is depicted in the Fig. 2.

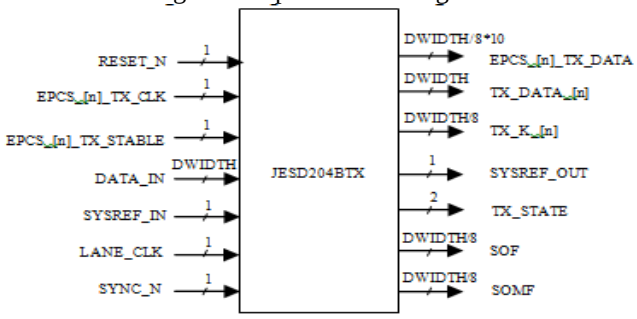

Fig. 2: JESD204B Transmitter I/Os and their width

The JESD204B transmitter consists of the following blocks. Alignment Character Generator block generates the

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

alignment characters and frame position signals. The ILA Sequence Generator block generates the initial lane alignment (ILA) sequence. The SYNC Signal Decoding block controls the sync request by decoding the SYNC_N signal. The Clock Generator block generates the internal LMFC and frame clock. 8B10B Encoder block encodes data using 8B10B encoding techniques. TX Controller block controls the triggering of the ILA sequence, generation of alignment characters, and the data presented to the 8B10B encoder. The Scrambler block scrambles data based on the polynomial 1 + x14 + x15 [6]. In [7] basic Block truncation Coding, the given data sequence is encoded in to a two level quantized approximation is described.

III. METHODOLOGY

The verification phase starts with the identification of the verification requirements. A test plan is framed and this plan becomes the guideline for the verification tasks. The DUT is tested for its functionality and performance using the verification environment thus created. Finally the test report will be created and stored in the record.

A. Test plan

The complexity in the functionality of the DUT is analysed and the necessary verification requirements are planned. The verification requirements represent the type of the test bench required, the kind of DUT modelling necessary, the simulation tools, the necessary components in the verification environments such as the number of agents or monitors, the scoreboard, the environment, the test scenarios or the test cases that must be considered, the range of the stimuli to be generated, and any other aspect that will concern the verification task. Thus the complete architecture of the verification environment is determined and agreed upon in the test plan.

B. The verification setup

The verification environment is developed using the SystemVerilog and the UVM. The architecture of the verification environment is represented as shown in Fig. 3. The components in the verification environment and the algorithm based on which they are developed is discussed in the following sections. Each class is preferably saved as a separate SystemVerilog file.

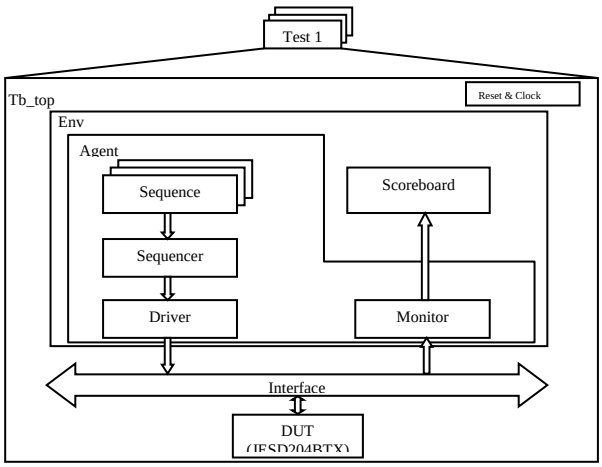

Fig. 3: The verification setup for JESD204B transmitter

1) Sequence items

The class _sequence_items extends from the uvm_sequence_items. This class contains the data fields required for generating the stimulus. The contents of the _sequence_items class is as shown in the flow chart of Fig. 4. The datafields are first declared and then registered into the factory using the utility macros. Then a constructor is defined in order to allocate memory using the function new.

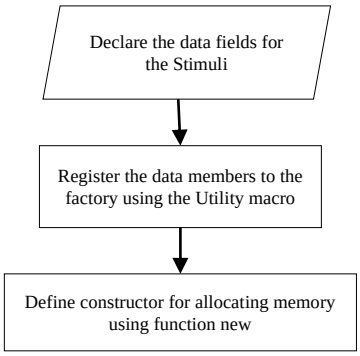

Fig. 4: The properties of the sequence item class

2) Sequence

The class _sequence extends the uvm_sequence with _sequence_items as the parameter. The contents of the _sequence class is as shown shown in the flow chart of Fig. 5. The stimuli pattern is defined in the sequence class. The objects in the sequence class is registered to the factory using the utility macros. Memory is allocated to the objects in the sequence using the function new constructor. The objections are raised in the pre_body method after which the stimuli are generated in the body method by randomizing the sequence item objects with necessary constraints. In the post_body method the objections are dropped.

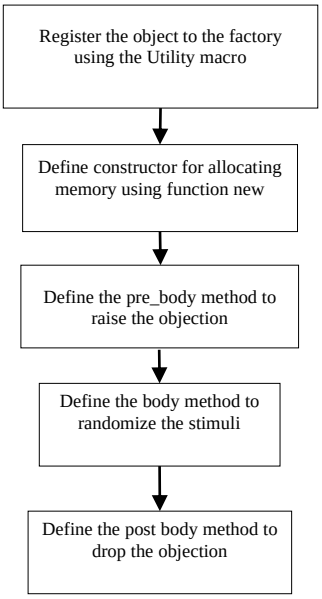

Fig. 5: The properties of the sequence class

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

3) Sequencer

The class _sequencer extends the uvm_sequencer with _sequence_items as the parameters. The contents of the _sequencer class is as shown in the flow chart of Fig. 6. The sequencer fetches the stimuli from the sequence and supply it the driver. The objects of the sequencer is registered to the factory using the utility macros after which memory is allocated to the objects using the function new constructor.

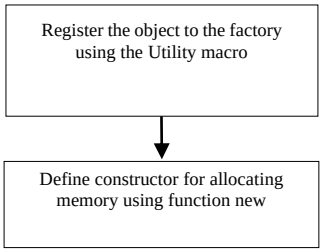

Fig. 6: The properties of the sequencer class

4) Driver

The class _driver extends the uvm_driver with sequence_items as the parameter. The contents of the driver class is as shown in the flow chart of Fig. 7. The driver collects the stimuli supplied through the sequencer and applies it to the DUT. The objects of the driver class is registered to the factory using the utility macros. The instance of the virtual interface is declared and mapped with the interface registered in configuration database using the get method in the build phase. A handle to the sequence items is declared and memory is allocated using the function new construct. In the run phase task, the constrained random values from the sequence is fetched using the sequencer through get_next_item method and drives the DUT through the interface.

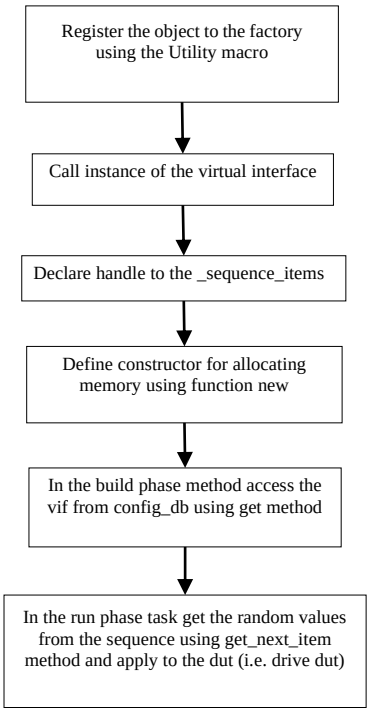

Fig. 7: The properties of the driver class

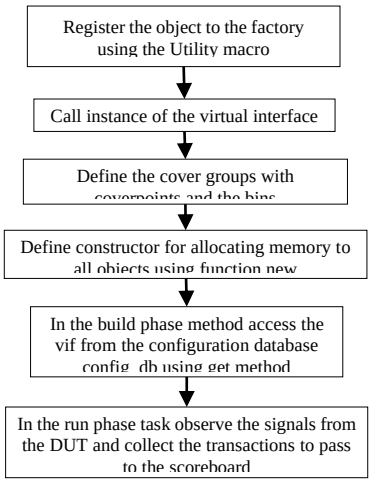

Fig. 8: The properties of the monitor class

5) Monitor

The class _monitor extends the uvm_monitor. The contents of the monitor class is as shown in the flowchart of Fig. 8. The monitor taps the inputs and outputs of the DUT from the interface. The signals are checked for the coverage and passed on to the scoreboard for evaluation. The objects in the monitor class is registered to the factory using the utility macro. A virtual instance of the interface is called mapped to the interface registered in the configuration database in the build phase. The cover groups are defined with coverpoints and bins to compute the functional coverage. In the run phase the signals are tapped from the DUT interface and the collected transactions are passed on to the scoreboard.

6) Agent

The class _agent extends from the uvm_agent. The contents of the monitor class is as shown in the flowchart Fig. 9. The agent class encloses the sequence, sequencer, driver and the monitor. The objects of the agent class is registered to the factory using the utility macro. The handles to the components are declared and memory is allocated using the function new construct. The child components and the ports are constructed and configured in the build phase. In the connect phase the export of sequencer is connected with the port of the driver.

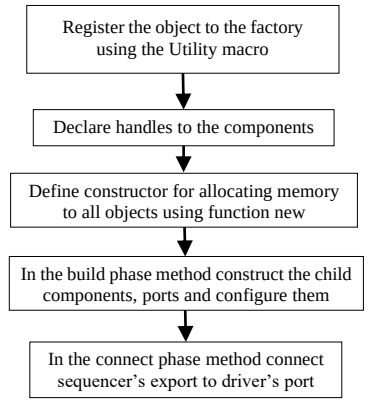

Fig. 9: The properties of the agent class

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

7) Scoreboard

The class _scoreboard extends from the uvm_scoreboard. The contents of the scoreboard class is as shown in the flowchart of Fig. 10. The scoreboard takes the stimuli and corresponding response of the DUT observed using the monitor. These DUT responses are compared with the computed or reference expected response to determine the validity of the DUT functionality. The objects of the scoreboard class is registered to the factory using the utility macro. The handles to the export, fifo and transactions are declared and memory is allocated using the function new construct. The build phase method constructs the component ports and configures them. In the connect phase the is connected to the tlmfifo. The run phase observes the transactions and compares them with the expected values.

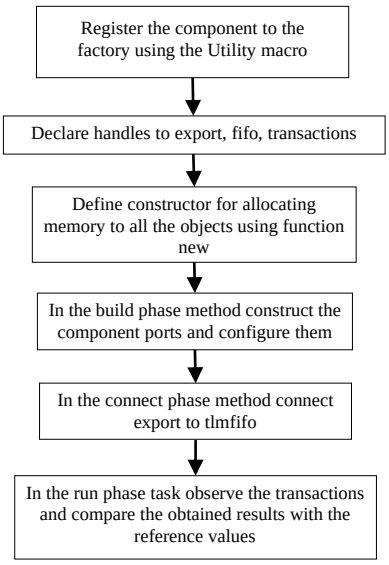

Fig. 10: The properties of the scoreboard class

8) Environment

The class _env extends the uvm_env. The contents of the _env class is as shown in the flowchart of Fig. 11. The environment contains the agent and the scoreboard. The objects of the environment class is registered to the factory using the utility macros. The handles to the components are declared and memory is allocated using the function new construct. The build phase constructs the component ports and configures them. The connect phase connects the agent's port to the scoreboard's export.

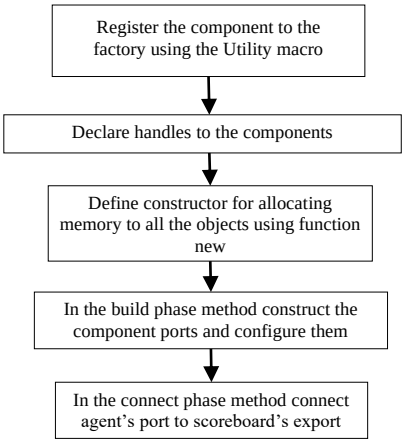

Fig. 11: The properties of the env class

9) Test

The class _test extends the uvm_test. The contents of the _test class is as shown in the flowchart of Fig. 12. The test encloses the environment. It runs the corresponding sequence associated with the test case. The components in the test class are registered into the factory using the utility macros. The handle to the environment is declared and memory is allocated to the objects using the function new construct. The child components are constructed and configured in the build phase. In the run phase the drain time is set for the environment. Errors reported by the scoreboard is checked in the extract phase and test result is reported in the report phase.

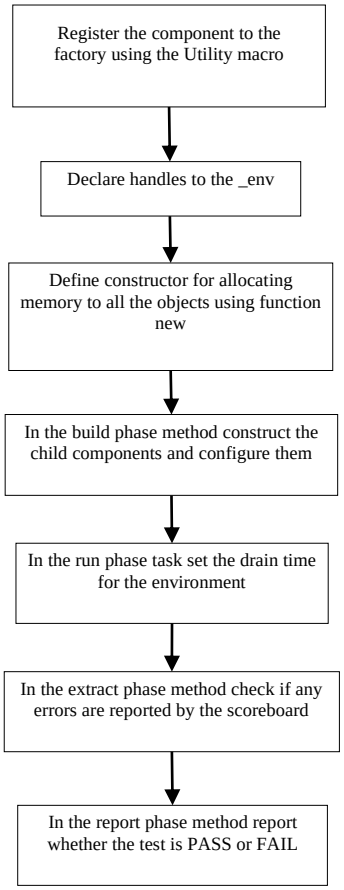

Fig. 12: The properties of the test class

10) Top module

The top most module of the testbench is the tb_top. The contents of the tb_top is as shown in the flowchart of Fig. 13. The tb_top connects the DUT and the verification environment through the virtual interface. In the tb_top libraries are imported the instance of the virtual interface is called. The virtual interface is registered in to the configuration database using the set method. The instance of the DUT is called and mapped with the virtual interface. The clock signals are generated and the reset signals are initialized before the run_test method is called.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

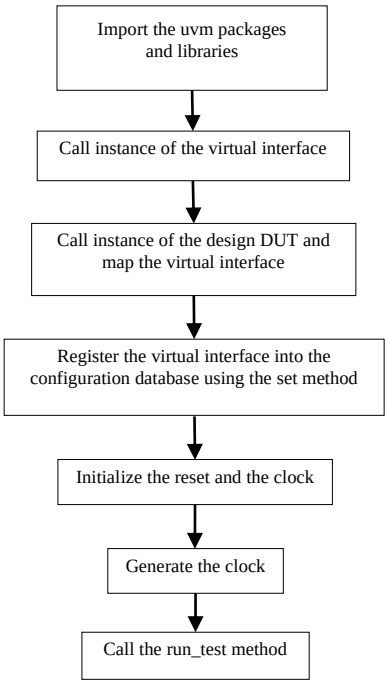

Fig. 13: The contents of the top module

C. Test Cases

The test cases are created by extending the _test class, already defined in the previous section, which is again extended from the uvm_test.

1) Reset Test

Test to check the reset functionality is represented by the flowchart in Fig. 14. This test case will generate reset sequences and random stimuli to verify that the DUT does not responds to any stimuli when reset is active. It also verifies if all the signals are taking up the default values at reset. The input user data is randomized and the reset sequence is generated. The response of all the outputs from the DUT are compared with the reference values or the default values that they must take. The process is repeated until the reset sequence is completed.

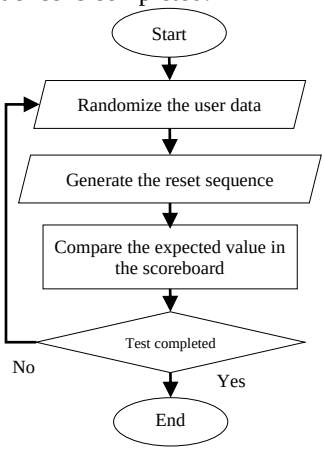

Fig. 14: Reset test case

2) Scrambler Test

Test to verify the scrambler functionality is represented by the flowchart in Fig. 15. This test case verifies the functionality of the scrambler by varying the parameters through their full range when scrambler is enabled. The test is performed in the following sequence. The transactions are randomly generated and applied to the DUT with Scrambler enabled. The scrambling is realized based on the polynomial 1+ x14 + x15. The scrambled user data is compared with the reference calculations. The scrambling behavior is also checked for the CGS and ILA sequence. The flags for start of the frame and multiframe is checked.

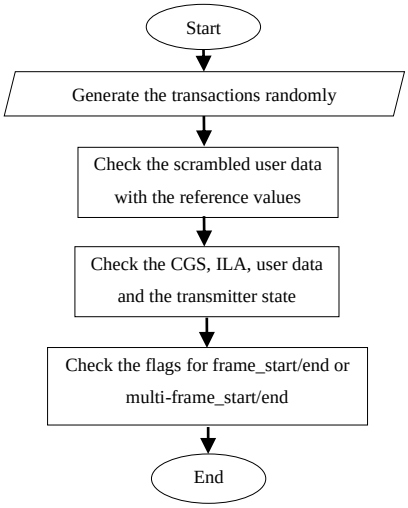

Fig. 15: Scrambler test case

3) Code Group Synchronization (CGS) Test

Test to verify the code group synchronization is represented by the flowchart in Fig. 16. This test case verifies the functionality of the code group synchronization task as per the JESD204B standard. CGS occurs at reset or when the receiver requests for synchronization through the SYNC_N signal. The CGS test performs the test is the following steps. The transactions are generated randomly with constraints. After reset is released, check if the transmitter emits stream of at least four K28.5 symbols and till the SYNC_N is disabled and the end of the frame boundary or the multi frame boundary is reached as per the configuration. Repeat the exercise by making SYNC_N signal low for more than four frame clocks and then disable the SYNC_N.

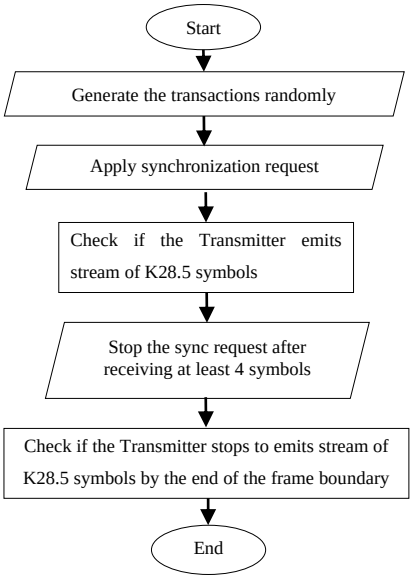

Fig. 16: CGS test case

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

4) Initial Lane Alignment (ILA) Test

Test to verify the ILA sequence is represented by the flowchart in Fig. 17. This test case verifies if the ILA sequence generated by the transmitter is in order as per the JESD204B standards.

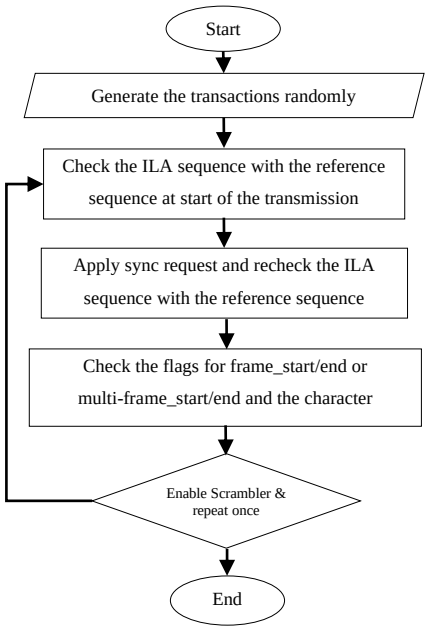

Fig. 17: ILA test case

All the test cases is included in the test sequence library. During the simulation of test cases the execution will go through in a number of phases, which is defined by the UVM library constructs. The initial block in the test bench starts all the phases through a task named run_test (). This run_test () task will lead to the construction of the top level components test which includes the construction of all the sub modules in the environment, sequences and execution of the sequences, end of the simulation and generates an overall report of the activities.

The test library supplies the test sequences, which is controlled by the sequencer. The transactions are then passed to the driver, which drives at the pin level. The DUT will receive all these transactions for the specific functionalities and accordingly produce the output. The monitor on the other side will also collect these transactions and pass it to the scoreboard, which has a reference model and get the expected output. Both the outputs are compared in the scoreboard and the result is displayed. The functional coverage is calculated based on the hits to the bins of each cover points defined in the cover group of the monitor.

The simulation is initially run for each test case using the GUI for one of the configurations. The simulation for the complete range of parameter is automated using the pearl script.

The overview of the actual methodology for verification of JESD204B transmitter is discussed in this chapter. A test plan is created initially and the verification environment is built as per the test plan. The DUT is then tested for different test cases using the verification environment. The simulation is ran with the aid of automation using the perl scripts.

IV. RESULTS

The complete simulation is performed in the Linux environment using the QuestaSim tool from Mentor Graphics. The verification code is written using the gVim editor.

A. Simulation outputs

1) Reset Test

The screenshot of the simulation waveforms for the reset test is presented in Fig. 18. The simulation is run with the following configuration: L=0, F=60, K=15, ILA-MFS=50, SUBCLASSV = 2, SCR = 0, SERDES_MODE=1, D_WIDTH=16, Encoder_EN=0, JESDV=1

From Fig. 18 it can be observed that when the reset is applied the DUT outputs are also getting reset to the default values.

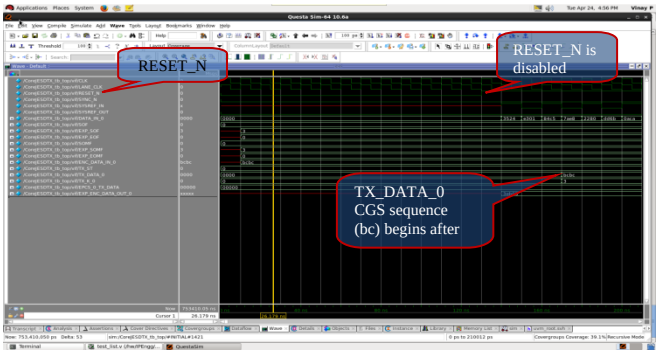

Fig. 18: screenshot of the simulation waveform for the reset test

In Fig. 19 the simulation transcript can be seen where the verification environment has completed the test and the reset test is passed.

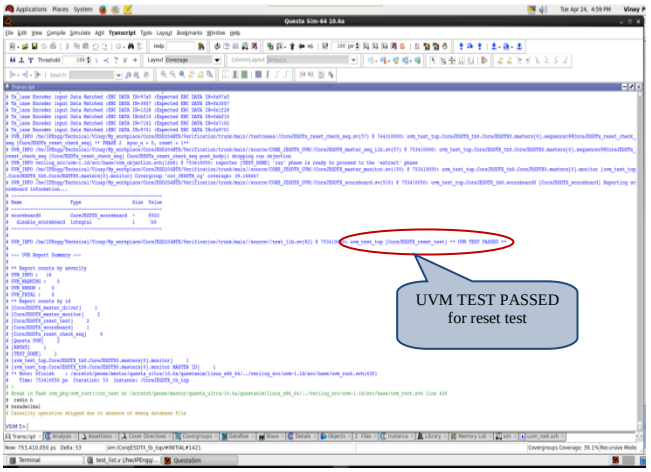

Fig. 19: screenshot of the simulation transcript for reset test

2) Scrambler Test

The screenshot of the simulation waveforms for the scrambler test is presented in Fig. 20. The simulation is run with the following configuration: L=0, F=60, K=15, ILA-MFS=50, SUBCLASSV = 2, SCR = 1, SERDES_MODE=1, D_WIDTH=16, Encoder_EN=0, JESDV=1.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

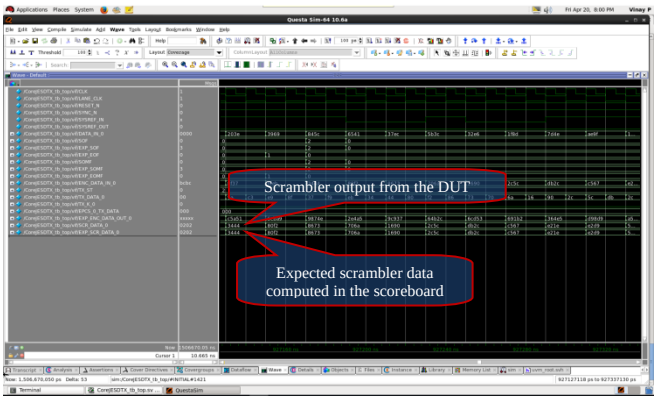

Fig. 20: screenshot of the simulation waveform for scrambler test

It can be observed that the user data DATA_IN_0 is given as input to the scrambler and the signal coming out of the scrambler is taped(SCR_DATA_0) for the verification purpose. The scrambler output matches with the expected signal(EXP_SCR_DATA_0) which is computed in the scoreboard.

In Fig. 21 the simulation transcript can be seen where the verification environment has completed the test and the scrambler test is passed.

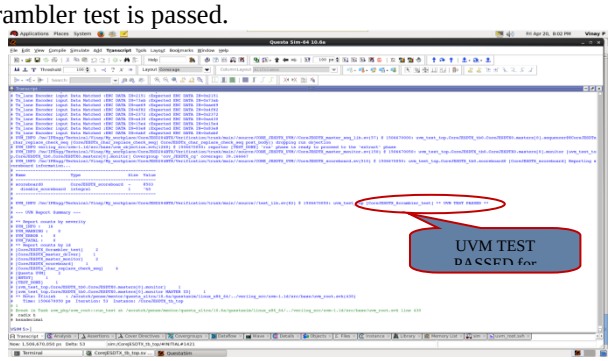

Fig. 21: screenshot of the simulation transcript for scrambler test

3) Code Group Synchronization Test

The screenshot of the simulation waveforms for the code group synchronization test is presented in Fig. 22. The simulation is run with the following configuration: L=0, F=60, K=15, ILA-MFS=50, SUBCLASSV = 2, SCR = 0, SERDES_MODE=1, D_WIDTH=16, Encoder_EN=0, JESDV=1. During code group synchronization a stream of K28.5(/K/) control characters is sent by the transmitter until the sync request by the receiver is stopped.

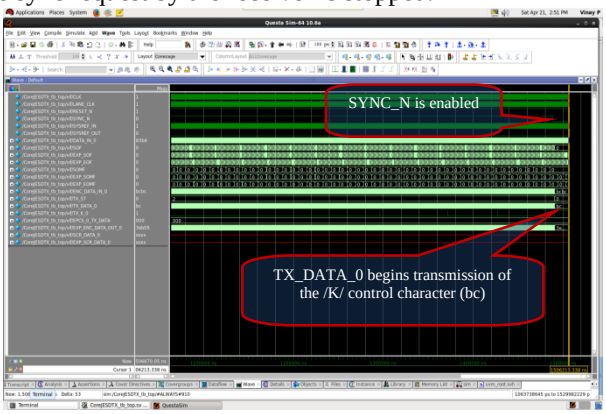

Fig. 22: screenshot of the simulation waveform for CGS test

In Fig. 23 the simulation transcript can be seen where the verification environment has completed the test and the scrambler test is passed.

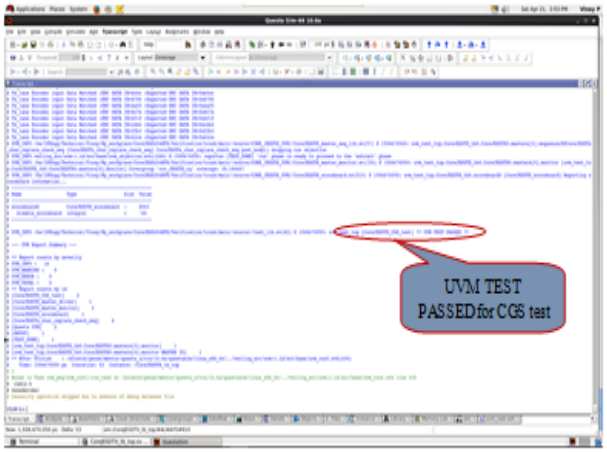

Fig. 23: screenshot of the simulation transcript for CGS test

4) Initial Lane Alignment Test

The screenshot of the simulation waveforms for the initial lane alignment test is presented in Fig. 24, 25. The simulation is run with the following configuration: L=0, F=60, K=15, ILA-MFS=50, SUBCLASSV = 2, SCR = 0, SERDES_MODE=2, D_WIDTH=16, Encoder_EN=0, JESDV=1. After the code group synchronization, initial lane alignment begins at the next multiframe. The first octet in every multi frame is represented by the K28.0(/R/) control character, which can be observed in the ENC_DATA_IN_0 and also the TX_DATA_0 signals in Fig. 24.

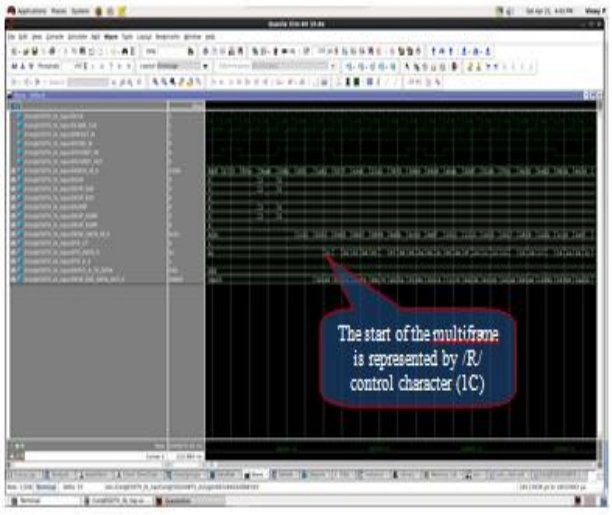

Fig. 24: screenshot of the simulation waveform for ILA test

At the end of the multiframe control character K28.3(/A/) is transmitted. The second multiframe in the ILA sequence contains fourteen octets of the link configuration data. The start of the link configuration data is preceded by the control character K28.4(/Q/). This can be observed in the Fig. 25.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

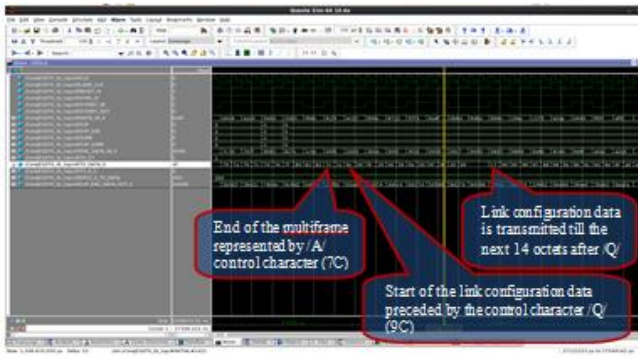

Fig. 25: screenshot of the simulation waveform for ILA test

In Fig. 26 the simulation transcript can be seen where the verification environment has completed the test and the ILA test is passed.

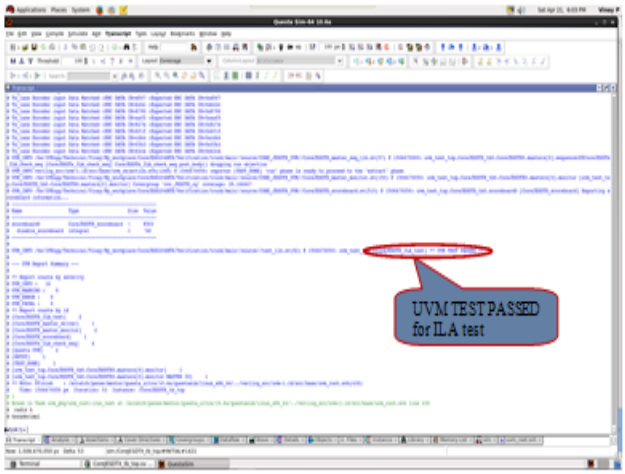

Fig. 26: screenshot of the simulation waveform for ILA test

B. Coverage Report

The Questa coverage report summary shows the overall functional coverage is 100%. The code coverage is 94.25% with statements coverage of 96.27% and branch coverage of 92.23%. This can be seen in Fig. 27.

Fig. 27: Coverage report summary generated using the questasim simulator

The logs of the summary of individual tests with their status can be seen in Fig. 28. For one such tests (one configuration of ILA test with L=0, F=60, K=15, ILAMFS=50, SUBCLASSV2, SCR=0, SEERRDES MODE=1, DWIDTH = 16) the verification failed and an error status is displayed.

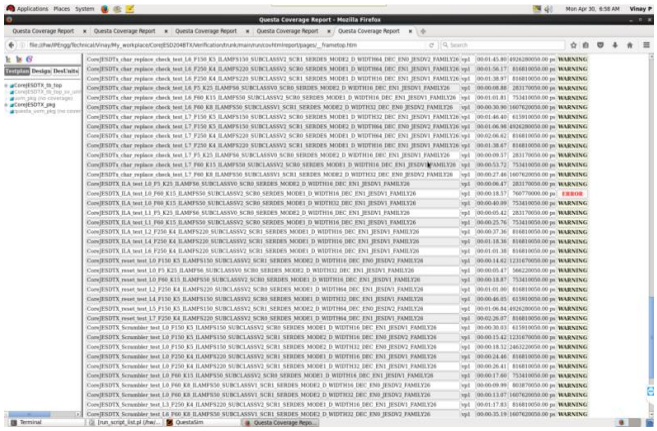

Fig. 28: The coverage report for tests and results from questasim

V.CONCLUSION

Verification of a design of any complexity is a very important phase in the development cycle. It is important that the design is tested for all the possible scenarios that may occur during it operation, so that it does not fail or malfunction after it is released for operation. If the defect in the design is detected at the last stage of production, such as system validation, it becomes an expensive task to reiterate the development cycle from the beginning. Thus it is desirable to detect the defects in the very early stage of development; hence design verification plays an important role.

The JESD204BTX Soft IP design was verified for its functionality using the SystemVerilog based UVM verification environment. The universal verification methodology provided a systematic framework for the construction of the testbench through its base class libraries, thus easing the task of writing complex testbench code. The IP design was tested for all the possible operational scenarios using different test cases at different configurations. The completeness of the verification is measured using the concept of functional coverage and code coverage. For each test case functional coverage of around 39.17% is achieved. A overall functional coverage of 100% was achieved with a code coverage of 94.25%.

The code coverage can still be improved by identifying the part of the code that is not getting hit and including more stimuli/configuration/test cases appropriately. The UVM components can be reused for the verification of the JESD204B receiver.

REFERENCES

1. H. Saheb and S. Haider, "Scalable high speed serial interface for data converters: Using the JESD204B industry standard," 9th International Design and Test Symposium (IDT), Algiers, pp. 6-11, 2014
2. Y. N. Yun, J. B. Kim, N. D. Kim and B. Min, "Beyond UVM for practical SoC verification," International SoC Design Conference, Jeju, pp. 158-162, 2011

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

3. Universal Verification Methodology(UVM) 1.2 Class Reference, Accellera, 2014
4. Universal Verification Methodology(UVM) 1.2 User's Guide, Accellera, 2015
5. IEEE Standard for Universal Verification Methodology Language Reference Manual," in IEEE Std 1800.2-2017, vol., no., pp.1-472, May 26 2017
6. JEDEC Standard JESD204B (July 2011), JEDEC Solid State Technology Association.
7. N.Shylashree, "Two Stage Block Truncation Coding for Lower Mean Square Error", IJRTE, 2018.