Grenze International Journal of Engineering and Technology, Jan Issue

Formal Vs Simulation based Verification on Receiver Training Sequence Block of USB3.2 Controller

Sai Asrith Tabdil[1], Dr Ramesh K Sunkaria[2] and Sakthivel Ramaiah[3]
[1-3]Department of Electronics and Communication Engineering Dr. BR Ambedkar National Institute of Technology Jalandhar, Punjab, India
Email: saiat.ec.21@nitj.ac.in, sunkariark@nitj.ac.in, sramaiah@cadence.com[3]

***Abstract*— The constant evolution of high-speed digital inter- faces has created a demand for robust and reliable designs. As a key component in ensuring reliable data transfer, receiver training sequence modules play a critical role in high-speed serial communication protocols like USB (Universal Serial Bus). Formal verification and Simulation based verification are two different methods for verifying the correctness of any digital design. The Simulation based verification checks the correctness of the design through the simulation of the design. It includes building of test bench architecture using System Verilog (SV) and Universal Verification Methodology (UVM), whereas Formal verification is a mathematical approach of verifying the design exhaustively. This paper gives a comparative analysis of a design using the two different verification methods. The block subjected for the verification has been taken from the Universal Serial Bus 3.2 (USB) controller. Using formal verification two corner case bugs were found and fixed at the early stage of the project and 100% code coverage is achieved.**

***Index Terms*— USB 3.2, Formal Verification, Functional Verification, Code coverage, RTL Sign off.**

I. INTRODUCTION

USB 3.2 controller is an advanced standard for USB technology, which is widely used in connecting various devices and peripherals to computers and other host systems. It is an evolution of the previous USB 3.1 and USB 3.0 specifications, offering increased data transfer rates and improved capabilities to meet the growing demands of modern devices. It has applications in mobile applications like keyboards and mice, routers, set-up boxes and in industrial automation. It is also used as source of power for mobiles and monitors. USB 3.2 introduces several key features and enhancements that enhance data transfer speeds, power delivery, and protocol efficiency. USB 3.2 controller supports higher data transfer rates compared to its predecessors. It offers two speed levels: GEN1 and GEN2. This protocol also supports dual lane which allows combining two 10 Gbps channels to achieve a max speed of 20 Gbps. This protocol is backward compatible which supports previous USB specifications' 3.2 devices can be connected to USB 3.1, USB 3.0, and USB 2.0 ports, ensuring interoperability between various generations of USB technology.

One crucial aspect of USB 3.2 controller is receiver training sequences (RXTS). RXTS is a mechanism used to compensate for channel impairments and ensure reliable data communication between the transmitter and the

Grenze ID: 01.GIJET.10.1.88

receiver. In USB 3.2 controller, RXTS involves a series of test patterns (TS1, TS2, SYNC OS) transmitted by the transmitter to the receiver during the link training process. These test patterns help characterize the channel and control the features like loop back, disable scrambling and Hot Reset. In addition to this, design also detects the polarity inversion.

TABLE I: USB 3.2 MODES

	Speed	Lane Count	Max Data rate (Gbps)
USB 3.2 1x1	GEN1	1	5
USB 3.2 1x2	GEN1	2	10
USB 3.2 2x1	GEN2	1	10
USB 3.2 2x2	GEN2	2	20

A. Verification Techniques

1) *Simulation based Verification*: Simulation-based verification for RTL (Register Transfer Level) sign-off is a critical process in digital design and verification. RTL refers to the level of abstraction in electronic design where the behavior and functionality of the circuit are described in terms of registers and data transfers between them. The purpose of RTL sign- off is to ensure that the RTL design meets all the specified requirements, adheres to the desired functionality, and is free from errors or bugs before proceeding to the next stage of the design flow. This method of verification for RTL sign- off involves using simulation techniques to thoroughly test and validate the RTL design. It aims to verify that the design behaves correctly, produces the expected outputs, and operates within the specified constraints and timing requirements.

2) *Formal Verification*: Formal verification uses mathematical techniques to prove or disprove specific properties of the RTL block. It involves creating formal properties and using tools like model checking or theorem proving to analyze the design and verify properties like safety, liveness, and functional correctness. Formal verification provides exhaustive and conclusive results, but it can be computationally expensive for complex designs. Formal verification for RTL sign-off offers several advantages over simulation-based techniques. It provides exhaustive analysis and guarantees complete coverage of the design space, eliminating the risk of undetected bugs or corner cases. It can detect deep corner-case errors that may be missed by simulation, and it offers a higher level of confidence in the correctness and reliability of the design.

II. RECENT WORKS

In the paper" A Holistic Approach to CPU Verification using Formal Techniques" [1] Rajakumar et al. discussed about divide and conquer Formal based CPU verification approach to effectively verify the complex CPU. The verification time required for this type of verification is less when compared to the simulation-based debug. Formal verification allows module level verification and covers all the corner case scenarios. They have witnessed 50 percent deduction in verification effort.

In the paper" An Efficient Formal Verification Method in I/O Multiplexing Module Based on VC Formal CC" [2] L., Hu et al. in the SOC level verification the time taken for verification and Return of Investment (ROI) plays an important role. Using simulation, we need to generate more test cases to hit the features. They have discussed about static formal method to verify the input/output level multiplexing at the SOC level. The engineer work is mainly focused on extraction of the CSV from the design specification.

In the paper" Closing the Verification Gap with Static Sign-off" [3] Ashar et al, they have discussed about the complexity that lies in verifying the complex SOCs. We need to select effective SOCs suitable for the sign off using Formal verification. 100 percent correctness of the design must be delivered by the verification strategy.

In the paper". Application Research of Formal Verification in Aerospace FPGA" [4] Liu, et al, they have discussed about the FPGA feature verification. To ensure safety and high reliability for the functions like face recognition and signal processing, formal verification technology became more popular. They have discussed about the challenges they have faced during the formal verification. one of the challenges was the exhaustiveness of the

formal, where the tool drives the values which are not suitable by the design. Effective constrains and black boxing is must for the invalid bugs reported.

III. Methodology

A. Simulation based Verification

The Simulation based verification flow that is followed is mentioned in the below flow chart Figure 1.
When the design is to be verified using the simulation-based verification, first step is to understand the specification and design. The next step is to extract the features from the design. This verification method depends on creating the test bench and UVM architecture. Once the extraction of features is completed, test bench architecture is needed to be developed, test cases and generate random sequences are needed to be created to verify the features mentioned in the vPlan. This type of verification depends on the depth first approach. The test case which is written may or may not hit the bug present in the design as it is constraint random simulation. This kind of verification is best suitable for the IP level verification.
This kind of verification takes most of the time in creating the test bench architecture. The debug is easier because the wave forms and logs will be present to verify the flow, while as in Formal the debugging is hard as there is no waveform generated. Only the counter example waveform can be generated. After the test cases creation and debugging the next step is to check the functional coverage of the DUT. If there are any gaps in the Functional verification, we need to write more sequences or test cases to prove those features. The main target is to hit all the features mentioned in the verification plan. The next step is to focus on the Code Coverage. Once the FC and CC reaches to the decent percentage the DUT is ready to be signed off.

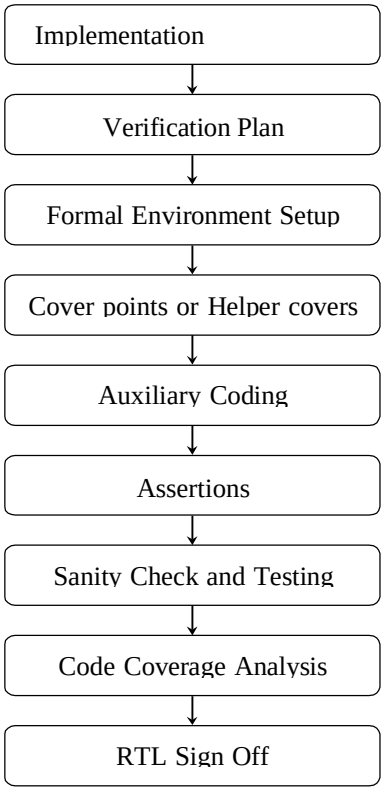

Fig. 2: Formal Verification Flow

B. Formal Verification

Formal verification techniques rely on formal methods, which are based on logic, automata theory, and mathematical reasoning. These techniques provide a formal framework for verifying the correctness of a design by exhaustively exploring its state space and checking whether the specified proper- ties hold under all possible scenarios.In formal verification, properties are formally specified to capture the behavior and requirements of the design. These properties can include safety properties (e.g., absence of certain errors), liveness properties (e.g.,

eventual occurrence of specific events), and temporal properties (e.g., timing constraints). Formal Verification checks for all the combinations of inputs.
The Formal verification flow which we have followed is mentioned in the flow chart Figure 2.
1)Implementation Specification: The implementation specification of the module is must for the formal verification. The designer need to provide the whole information about the Design Under Testing (DUT). This document gives a clear idea about the inputs and output behaviour of the DUT and should also give some basic functional diagrams of the DUT.
2)Verification Plan: The Verification Plan is the most important step of formal verification as all the features of the DUT need to be captured in this Verification Plan(vPlan). This mainly contains features and helper covers.
3)Formal Environment Setup: In this stage after the vPlan creation, formal environment is to be created. The top file and checker file for the formal should be created and bind it with the DUT. The checker module contains the model, constraints and assertions. In the initial stage constrains or the assume properties are important as these help in constraining the behaviour of the DUT. Since the tool checks for 2n combinations, constraints help in sending valid inputs to the DUT. The reset assertion is added to ensure that there wont be any false positives due to reset condition.
4)Cover points or Helper Covers: At the initial stages of the verification we can take the help of the cover properties to verify the straight forward scenarios. These covers help in determining over constraints, false positives and vacuous pass. These help in tuning the model at the early stages and at the final stages. using some helper covers we can ensure that all corner case scenarios are being hit. These also help in determining over constrains in the environment.
5)Auxiliary Coding: When we are dealing with some com- plex designs the complex logic cannot be directly checked in the form of assertions. We need to model the DUT behavior such that the complexity that lies in the assertions can be reduced. As debugging is harder in Formal Environment we can use as many flags as possible which help in debugging the properties.
6)Assertions: These are the key properties that exhaustively checks the design. If any property fails, then a counter example is generated, and it is needed to be debugged. The assertion properties are written based on the features extracted in the vPaln.
7)Sanity Checking and Debugging: Once all the assertions are written we need to run the tool and check for correctness of the assertions. Once all the assertions are passed, we check for the coverage analysis.
8)Code Coverage Analysis: In this stage we check the toggle expression and block coverage, and we need to make sure that we reach 100 percent coverage. If we have any gaps in the coverage more assertions need to be written to cover those scenarios.
9)RTL Sign off: The major goal of formal verification is to verify the DUT with no corner cases being uncovered. Since Formal uses breadth first approach there is no chance of the bug to escape. Once the code coverage is 100 percent and all the properties are proven, the DUT is ready to be signed off.

IV. Case Study: Receiver Training Sequence Of USB 3.2 Controller

In USB 3.2 we have selected receiver training sequence module in the RX Path. This module is responsible for the link configuration and training. The training sequences received by this DUT will be tracked and based on the required count the LTSSM transition takes place. Additionally, the RX TS module also tracks the polarity inversion.

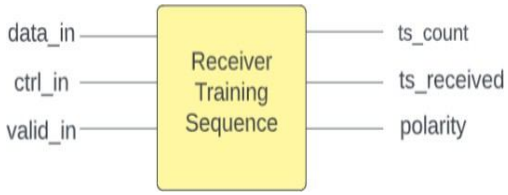

Fig. 3: Design under Verification

Calculation for the total number of combinations:

- Number of operational modes = 2 (GEN1, GEN2)
- Number of ordered sets = 3 (TS1, TS2, SYNC OS)
- Number of symbols for an OS = 16 and number of combinations = 216
- Number of consecutive and identical OS required = 8 and number of combinations = 28
- Number of alignments = 4

- Total number of combinations = 2 * 3 * 216 * 28 * 4

This makes it improbable to cover all the possible scenarios in the simulation-based verification. Using formal the toolconsiders for all these combinations and hence exhaustive in nature.

A. vPlan

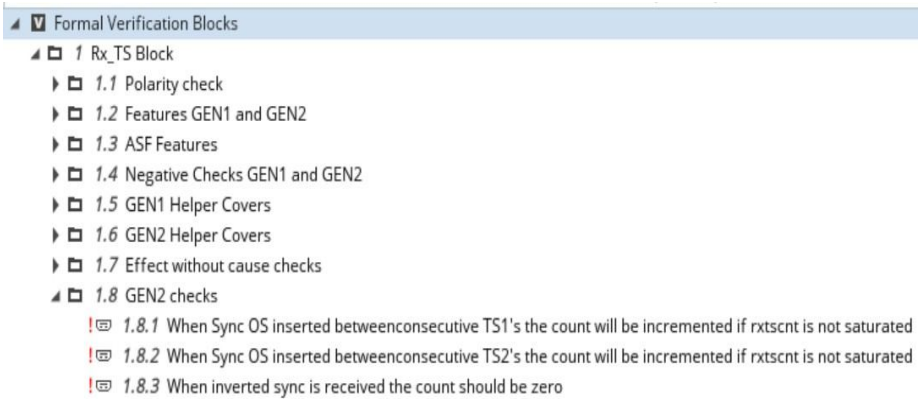

Fig. 4: vPlan Hierarchy

In the vPlan all the features are extracted. The features are extracted based on the functionalities of the DUT mentioned in the USB3.2 specification as well as the Implementation document. Each functionality can be described in three different features. The strength of Formal verification lies in the extraction of all the features of the DUT.

1) *Positive Check*: Positive check will check for" When there is a Cause effect should occur".[6]
Example: When DUT has received consecutive and identical TS1/TS2 OS, the count should be incremented.
2) *Negative Check:* These checks will check for" When there is no cause, effect should not be present"
Example: When DUT has not received a normal TS1/TS2 OS, DUT should not detect it.
3) *Effect Without Cause Check*: These checks will check for" When the effect is present, the cause should have hap- pened."
Example: When the count is incremented, the DUT should have received consecutive and identical TS1/TS2 OS.

B. Auxiliary Logic

Tracking the Training Sequences is the main task of this module. We have two different modes and three types of training sequences TS1, TS2 and SYNC OS. When we try to track these sequences directly in the form of assertion properties, the complexity needed to keep in the assertions will be increased. To reduce this we have modelled the behaviour of the RTL in the checker module. Using the flags we can test the behaviour of the DUT. This reduces the load on the formal verification tool. In this analysis we have used Cadence Jasper FPV app.

V. RESULTS

The DUT was tested using both the verification strategies. Figure 6 shows the summary of the test run using formal verification. Total assertions written were 45 and all are passed and all the cover properties are covered.

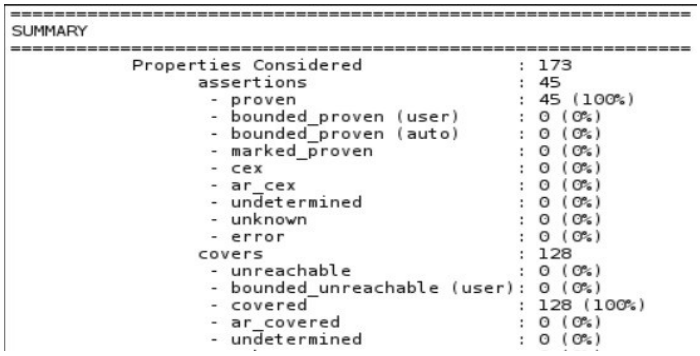

Fig. 5: Formal Run Summary

A. Corner Case bugs

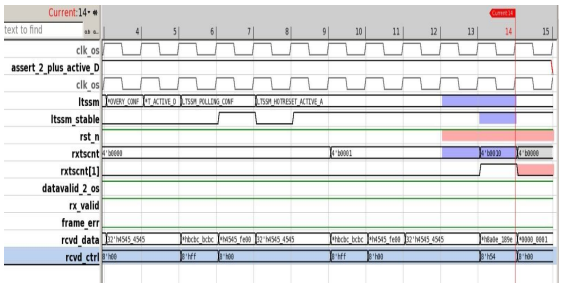

Fig. 6: Bug Scenario 1

1) *Bug Scenario 1*: When the DUT is in HOT reset states and received consecutive and identical TS1/TS2 the count should be incremented. When the count is sampled to 2, then it should hold the value until LTSSM changes. In the figure 6 we can see that in clock 14, the count is saturated (count = 2) and the LTSSM is in HOT RESET state, but the count is being reset in clock 15. These kind of corner cases will be easily hit in the formal as all the combinations of input signals will be tested. This scenario was hit for the assertion property mentioned below.
AST tsx 2 plus ts2 hold : assert property (disable iff (!datavalid 2 os) hot reset state ##1 rxtscnt == 4'd2 && ltssm stable =⇒ rxtscnt ==4'd2);

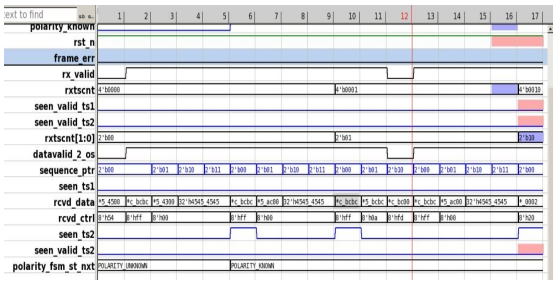

Fig. 7: Bug Scenario 2

2) *Bug Scenario 2*: When there is any gap in between the TS1/TS2 the consecutiveness should be broken, and the count should be reset to zero. But as shown in the counter example figure 7, in the 12th cycle even if the rx valid signal is de asserted, the count is being incremented in the next cycle.

B. Back Annotated vPlan

Name	Mapped Elements	Assertion Formal Proved	Formal Covered
(no filter)	(no filter)	(no filter)	(no filter)
Formal Verification Blocks	60 / 60 (100%)	47	60 / 60 (100%)
1 Rx_TS Block	60 / 60 (100%)	47	60 / 60 (100%)
1.1 Polarity check	4 / 4 (100%)	4	4 / 4 (100%)
1.2 Features GEN1 and GEN2	15 / 15 (100%)	15	15 / 15 (100%)
1.3 ASF Features	3 / 3 (100%)	3	3 / 3 (100%)
1.4 Negative Checks GEN1 and GEN2	17 / 17 (100%)	17	17 / 17 (100%)
1.5 GEN1 Helper Covers	7 / 7 (100%)	0	7 / 7 (100%)
1.6 GEN2 Helper Covers	6 / 6 (100%)	0	6 / 6 (100%)
1.7 Effect without cause checks	5 / 5 (100%)	5	5 / 5 (100%)
1.8 GEN2 checks	3 / 3 (100%)	3	3 / 3 (100%)

Fig. 8: Back Annotated vPlan and Formal Covered

In the back annotated vPlan all the assertions and cover properties are mapped to the verification plan. When the regression is run, we get the result as shown in the Figure 9. This helps in analyzing the vPlan and if any of the feature is not covered we can add more properties for the particular feature. When the mapping and functional coverage reaches 100%, code coverage can be analysed.

C. Code Coverage Analysis

Both of the verification methods gave decent code coverage, but formal allowed us to attain 100 % code coverage. Few of the negative scenarios were not hit in the Simulation based verification. For some scenarios the number of random test cases required will be more and may consume lot of time to hit the remaining cover items.

TABLE II: CODE COVERAGE ANALYSIS

Verification Strategy	Overall (%)	Block	Expression (%)	Toggle (%)
Simulation based	98.44	93.1	89.24	99.39
Formal	100	100	100	100

VI. CONCLUSION

The Formal verification and Simulation-based verification techniques have their own strengths and limitations. The Formal verification strategy can exhaustively verify the DUT covering all the corner scenarios. Once signed off using Formal, it is not required to sign off the DUT using any other verification methods. When the complexity of the DUT increases or verifying DUT at the IP level, the formal takes more time to converge. So in IP level simulation based verification with test bench architecture will be the best possible strategy. It is recommended to employ a combination of formal verification and simulation-based verification to achieve a comprehensive result.

REFERENCES

[1] Sri, D.B., Rajakumar, K., Madhusoodhanan, P. and Edarapalli, V.N., 2022, June. A Holistic Approach to CPU Verification using Formal Techniques. In 2022 IEEE Women in Technology Conference (WINTECHCON) (pp. 1-5). IEEE.

[2] Duan, L., Hu, Y., Liu, H., Feng, W. and Gan, J., 2020, December. An Efficient Formal Verification Method in I/O Multiplexing Module Based on VC Formal CC. In 2020 IEEE 3rd International Conference on Electronics and Communication Engineering (ICECE) (pp. 112-116). IEEE.

[3] P. Ashar and V. Viswanath, ”Closing the Verification Gap with Static Sign-off,” 20th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 2019, pp. 343-347, doi: 10.1109/ISQED.2019.8697608.

[4] Liu, S., Li, D., Shen, W., Wang, Z., Yang, G. and Song, X., 2021, December. Application Research of Formal Verification in Aerospace FPGA. In 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 797-805). IEEE.

[5] Hamish Hendry, Sakthivel Ramaiah, Derek McAulay, Gary Dick, Ca- dence, Accelerate Adoption of High-Speed, Low-Latency, Cache-Coherent Standards Using Formal Verification.

[6] Saji, S.A., Agrawal, S. and Sood, S., 2022, June. Formal Verification of Deep Neural Networks in Hardware. In 2022 IEEE Women in Technology Conference (WINTECHCON) (pp. 1-6). IEEE.

[7] Singh, R.K., Kumar, B., Shaw, D.K. and Khan, D.A., 2021. Level by level image compression-encryption algorithm based on quantum chaos map. Journal of King Saud University-Computer and Information Sciences, 33(7), pp.844-851.

[8] Mu¨ller, J., Fadiheh, M.R., Anto´n, A.L.D., Eisenbarth, T., Stoffel, D. and Kunz, W., 2021, December. A formal approach to confidentiality verification in SoCs at the register transfer level. In 2021 58th ACM/IEEE Design Automation Conference (DAC) (pp. 991-996). IEEE.

[9] K. Devarajegowda, L. Servadei, Z. Han, M. Werner and W. Ecker, ”Formal Verification Methodology in an Industrial Setup,” 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 2019, pp. 610-614, doi: 10.1109/DSD.2019.00094.

[10] Hofer-Schmitz, K. and Stojanovic´, B., 2020. Towards formal verification of IoT protocols: A Review. Computer Networks, 174, p.107233.

[11] A. Yadav, P. Jindal and D. Basappa, ”Study and Analysis of RTL Verification Tool,” 2020 IEEE Students Conference on Engi- neering and Systems (SCES), Prayagraj, India, 2020, pp. 1-6, doi: 10.1109/SCES50439.2020.9236747.

[12] Yeung, P., Khurana, A., Gupta, D., Prasad, A. and Mittal, A., Raising the level of Formal Signoff with End-to-End Checking Methodology.

[13] Chen, H., Vallabhapurapu, K.S., Peverelle, S., Yee, R., Kim, H.C., Te, J. and Hotz, J., Maximizing Formal ROI through Accelerated IP Verification Sign-off.

[14] Li, A., Chen, H., Yu, J.K., Teoh, E.L. and Anand, I.P., 2019. A Coverage- Driven Formal Methodology for Verification Sign-off. In Design and Verification Conference and Exhibition (DVCon) United States.

[15] Anand, A., Chen, C., Subramanian, B.N. and Hupcey, J., 2020. Bounded Proof Sign-off with Formal Coverage.

[16] Bhattacharyya, Joydeep. ”Formal Bug Hunting: An Immensely Powerful Merger of Simulation and Formal Verification Methodologies.”

[17] Duran, C., Morales, H., Rojas, C., Ruospo, A., Sanchez, E. and Roa, E., 2020, October. Simulation and Formal: The Best of Both Domains for Instruction Set Verification of RISC-V Based Processors. In 2020 IEEE International Symposium on Circuits and Systems (ISCAS) (pp. 1-4). IEEE.

[18] Chen, H., Sun, Y., Li, A. and Cao, D., 2020. A Systematic Formal Reuse Methodology: From Blocks to SoC Systems.

[19] Hofer-Schmitz, K. and Stojanovic´, B., 2020. Towards formal verification of IoT protocols: A Review. Computer Networks, 174, p.107233.

[20] Corsi, D., Marchesini, E. and Farinelli, A., 2021, December. Formal ver- ification of neural networks for safety-critical tasks in deep reinforcement learning. In Uncertainty in Artificial Intelligence (pp. 333-343). PMLR.

[21] Riedmaier, S., Ponn, T., Ludwig, D., Schick, B. and Diermeyer, F., 2020. Survey on scenario-based safety assessment of automated vehicles. IEEE access, 8, pp.87456-87477.
[22] Awasthi, A., Sagar, K.V., Dacha, A. and Sai, D.R., Deployment of Formal Verification for RISC Processor RTL Signoff.
[23] Yadav, A., Jindal, P. and Basappa, D., 2020, July. Study and analysis of RTL verification tool. In 2020 IEEE Students Conference on Engineering & Systems (SCES) (pp. 1-6). IEEE.
[24] Parikh, B. and Narayana, S., Deadlock Free Design Assurance Using Architectural Formal Verification.
[25] Chen, H., Vallabhapurapu, K.S., Peverelle, S., Yee, R., Kim, H.C., Te, J. and Hotz, J., Maximizing Formal ROI through Accelerated IP Verification Sign-off.
[26] Bhattacharyya, J., Formal Bug Hunting: An Immensely Powerful Merger of Simulation and Formal Verification Methodologies.