PropGen: An Automated Flow to Generate SVA Properties for Formal and Simulation methods

Amith Shambhu, Specialist Engineer, Infineon Technologies, Bangalore, India
(Amith.Shambhu@infineon.com)
Vishal Dalal, Principal Engineer, Infineon Technologies, Bangalore, India
(Vishal.Dalal@infineon.com)
Basavaraj Naik, Senior Manager, Infineon Technologies AG, Munich, Germany
(Basavaraj.Naik@infineon.com)

***Abstract* - Contemporary System-on-Chip (SoC) designs commonly incorporate intricate design elements that encompass complex Data and Control Path logic. As Register Transfer Level (RTL) designs continue to advance in complexity and reusability, there is a growing demand for versatile and robust strategies in functional verification. To address this, a combination of simulation and Formal Verification (FV) techniques are employed to ensure exhaustive verification of the Design Under Test (DUT). This approach is called as Hybrid Verification in this paper. While simulation offers the advantage of detecting bugs deep inside the design, FV plays a crucial role in improving the overall quality of Design Verification (DV) by exploring all possible combinations within the design state space. System Verilog Assertions (SVA) are key to check design behavior in both Formal and Simulation methods. Manual SVA writing take lots of efforts and is often repetitive to adapt to design changes.**

This paper introduces a novel strategy and implementation flow named PropGen (Property Generator) which automates SVA generation. PropGen effectively captures the design specifications provided in Visio diagrams/XLS/XML and transforms them into SVA or property files that can be seamlessly integrated with existing FV or simulation Testbench (TB).

By utilizing the PropGen approach, the DUT undergoes exhaustive verification using FV, achieving 100% formal code coverage. Furthermore, the SVA generated by PropGen are rigorously validated through simulations, affirming that the properties derived from PropGen can be effectively utilized in both FV and simulation methodologies. PropGen has many use cases, and this paper focuses on Finite State Machine (FSM) as one use-case that reduces FSM verification efforts by ~90%.

Keywords— FSM; PropGen; Visio; Design Verification; DUT; Formal Verification; Simulation

I. INTRODUCTION

The verification of present-day DUT involves a combination of simulation and FV techniques. Simulation-based testing entails executing test cases on the DUT using specialized verification environments and TB to verify its functionality and performance. However, this approach has limitations as it relies heavily on manual test cases (TC), which can be time-consuming, error-prone, and may not cover corner cases. FV on the other hand, utilizes mathematical algorithms and proofs to exhaustively verify the correctness of a DUT in a short time. It explores all states and transitions of the design, ensuring that it adheres to the desired specifications. However, FV often require right expertise, computational resources and state space becomes complex with increase in sequential depth.

These verification approaches encounter various challenges. Firstly, the increasing complexity of modern designs results in a vast state space that is difficult to comprehensively cover using manual techniques. Additionally, design

modifications further compound the verification efforts, make them repetitive, and take additional time. Moreover, manual TC generation are labour-intensive and prone to human errors, leading to potentially undetected design bugs.

Automation is an indispensable factor in the DV process, and it is a powerful solution to address these challenges. As reinforced by compelling evidence from the literature, these studies consistently demonstrate the paramount importance of automation in overcoming the challenges associated with DV. Notably, in research by Smith et al. [1] and Johnson and Lee [2], the implementation of automated verification techniques led to a staggering 50% reduction in overall verification time compared to traditional manual approaches.

Typically, the verification process involves the Concept Engineer or architect to understand and write the IP design specifications using formats like Visio/XLS/XML. The Design Engineer implements the RTL code that is compliant to specification. The Verification Engineer then verifies that the RTL fulfills the specification for all possible design combinations exhaustively. The verification stage becomes crucial as it solely depends on interpretation of the specification that could lead to unverified timing sequences with the risk of undetected design errors.

The verification of complex DUT poses challenges in terms of coverage, tight execution schedule, and potential bugs that get missed. SVA assertions are inevitable part of TB and consume lot of time in DV cycle if written manually. The need for a systematic and automated approach to generate SVA properties from design specifications arises to reduce DV cycle.

This paper aims to address this problem by developing a novel methodology and implementation flow called PropGen. PropGen captures the intent of the DUT specifications in Visio/XLS/XML and automatically generates SVA properties for various use cases like registers and FSM DV. The effectiveness and efficiency of the PropGen approach is evaluated here through comprehensive DV, with a focus on verifying a complex FSM as one of the use cases. 100% code and formal coverage for the FSM design is achieved which demonstrate the completeness of the PropGen methodology. The goal is to reduce turnaround time, improve coverage, and enhance the overall quality and reliability of the DUT verification process.

II. PROPGEN FLOW

A. Background

The methodology proposed in this paper, takes a different approach as compared to the existing FV solutions which mostly rely on the RTL code. Instead of focusing solely on the RTL, PropGen captures the intent of the DUT specifications, providing a more intuitive representation of the DV Scenario. PropGen allows the user to generate properties in a simple user-friendly manner. The generated properties are easy to analyse enabling better debugging and refinement. This approach not only enhances the efficiency of the DV cycle but also optimizes DV resources. Figure 1 gives an overview of the PropGen flow.

The flow takes the verification specifications in the form of a Visio diagram, an Excel sheet, or metadata in the form of an XML. The design I/O Signal information is captured in a CSV format and given as input. The PropGen flow then converts the specifications into an intermediate format and make use of the in-built templates to generate the SVA properties. These properties can then be used to verify design using both formal and simulation methods.

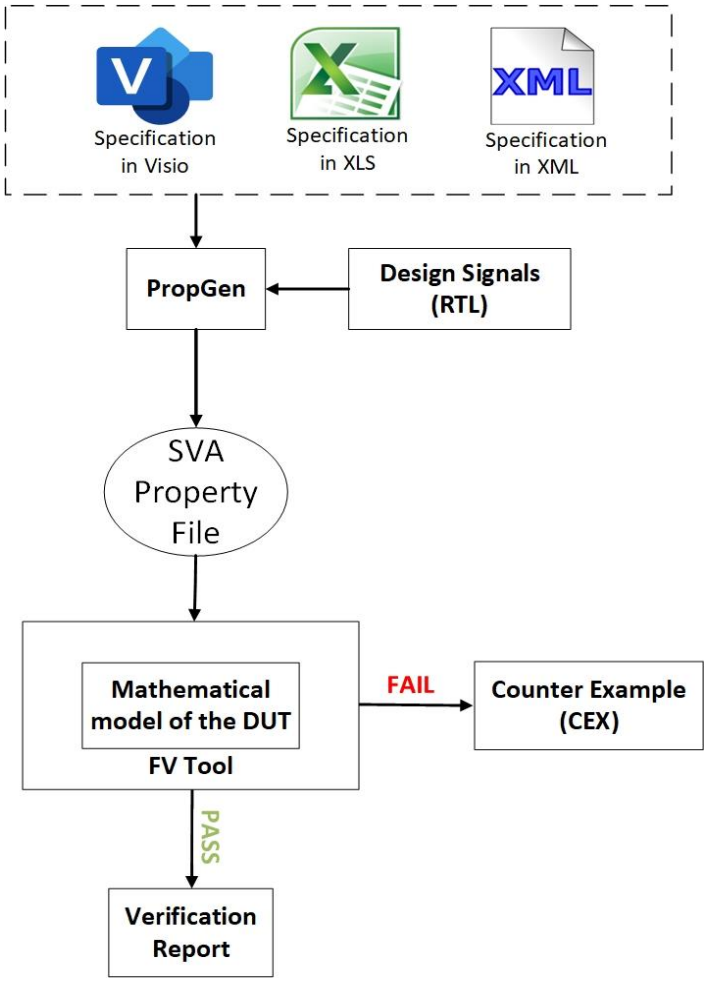


Figure 1. PropGen – detailed tool flow diagram

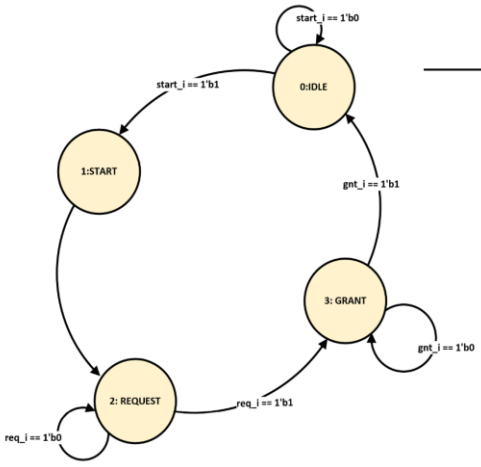
B. PropGen Architecture

The PropGen automation flow can be split into two parts:

i. Visio representation of FSM is converted to XML to restructure FSM as a data frame.
   - This step is skipped if the specification is already given as XML metadata or as an Excel sheet.

ii. Parsing the XML generated from above step to finally generate SVA property file.

From the XML generated from the Visio diagram, FSM can be restructured into a format that can give comprehensive information about the start state of FSM, state transitions and respective trigger conditions for the transition to occur as shown in figure 2 states table. The second component of PropGen automation generates SVA in a (.sv) file. It uses design I/O signal information that is additionally provided as input XLS. This I/O information is used in declaring the signals in the assertion binding and generate checks related to the output ports. The flow combines all these scenarios and develops a final (.sv) file. More details about the flow can be found in [3].



fromState	toState	condition
0:IDLE	0:IDLE	start_i == 1'b0
0:IDLE	1:START	start_i ==1'b1
1:START	2:REQUEST	
2:REQUEST	2:REQUEST	req_i == 1'b0
2:REQUEST	3:GRANT	req_i == 1'b1
3:GRANT	3:GRANT	gnt_i == 1'b0
3:GRANT	0:IDLE	gnt_i == 1'b1

```
property state_cond_trans(STATE1, cond, STATE2);
@(posedge Clk) disable iff(!AsyncRst)
(state == STATE1) && cond |=> (state == STATE2);
 endproperty

check_IDLE_to_START_condition : assert
property(state_cond_trans(IDLE,(start_i== 1'b1),START));
```

Figure 2. Different stages in the PropGen Flow

Concept Engineers initially capture the design details in Visio at a higher-level of abstraction. PropGen can use this to generate an intermediate XLS format that can later be added with more implementation specific details of conditions that trigger FSM transitions along with values of outputs during various states. RTL designer provides design specification that captures implementation details. In case designers provide full FSM implementation in Visio, PropGen can directly read and convert this to final SVA. PropGen therefore support seamless usage of both high-level abstraction as well as finer implementation details to generate SVA.

III. Use Case Examples

At Infineon Technologies generation of these SVA assertions are automated using various property generators as part of PropGen methodology. There are various applications or use cases of PropGen and some of them are summarized below.

One of the use cases is to generate SVA properties from register XML specifications to verify all register bits in a given memory map. There are numerous register access policies to verify full register space. Some of these policies are specific to organizations like Infineon and can be easily incorporated at in-house flow for current as well as future register access policy updates. Highly automated EDA tools to verify register accesses exist but their generated properties are encrypted and therefore not user accessible. Also, these tools will not be able to cover some of organization specific access policies.

Second use case is to generate SVA properties from XLS/XML specifications to verify integration of various safety critical building blocks used inside safety IP. Configurations of these individual building blocks and their integration use case is defined in XLS/XML format. SVA generators use these specifications to automate the property generation with-in the framework of PropGen. The generated properties can verify correct integration of these blocks using FV.

Third use case is to use PropGen to verify FSM. Figure 3 clearly describes the problem statement to choose FSM verification as PropGen's use case and need to have hybrid verification using FV and simulation methods.

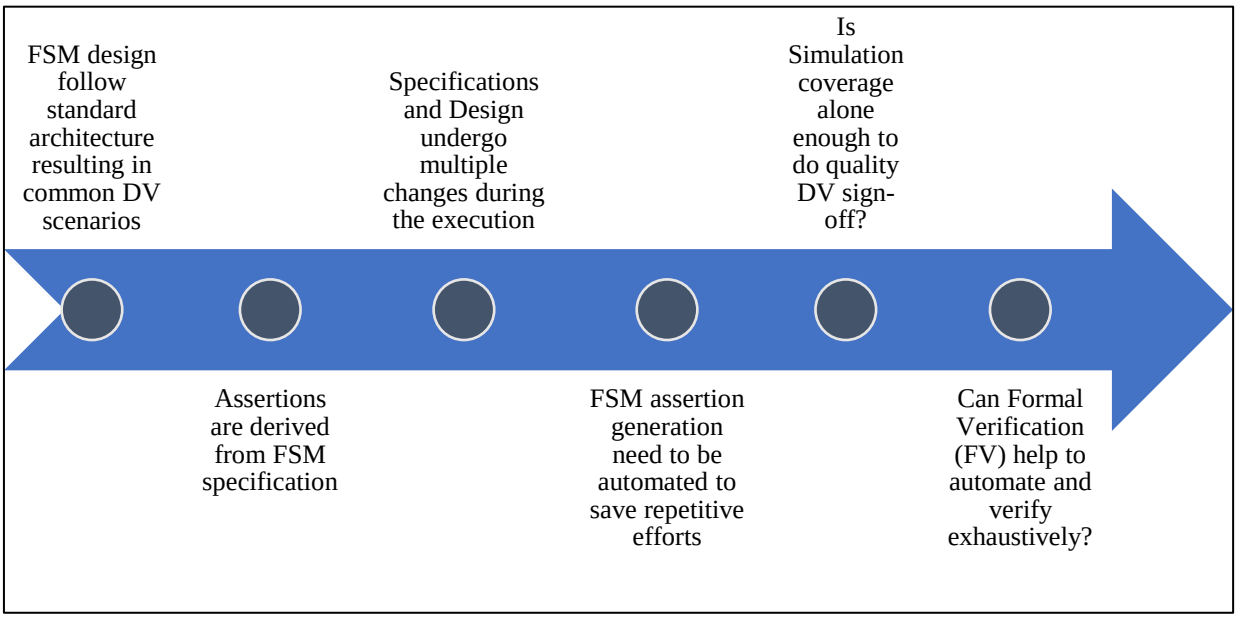


Figure 3. Challenges in FSM DV.

The SVA properties generated by PropGen for FSM verification are generic, and they cover various DV scenarios, required to verify any FSM. Some of these are listed below –

1. State transition checks – Properties that check for a state transition under certain input conditions.
2.Output reset checks – Check that input reset causes all output to get inactive values.
3. Livelock checks - Check that there is always a path back to the initial state. If the FSM states constantly change, but do not progress back to initial state, there will be a Livelock violation.
4. Deadlock checks – Check that there is an outgoing path from the current state whose condition can be met. If there is no outgoing path that particular state is stuck and there would be a deadlock violation.
5. Output checks – Design output checks when FSM enters different states.

A. What if Visio is not the tool of choice?

In this paper, the use of Visio as a tool for capturing design specifications is highlighted, and the PropGen infrastructure converts these specifications into SVA properties. Visio is widely used in industry to capture design specifications. However, it is acknowledged that Visio is NOT a universal tool of choice and therefore not all FSM specifications may be available in Visio. In such cases, there are workarounds available in PropGen to address this limitation. For example, users could use other commonly available tools such as Microsoft Excel (XLS) or XML formats to define and document their design requirements like FSM state transitions and input trigger conditions. The PropGen infrastructure can then be used to generate corresponding SVA.

By supporting multiple input formats like Visio/XLS/XML, PropGen ensures that users can leverage their preferred tool to define the design specifications. This flexibility allows a wider range of users, including those who may not have access to Visio, to utilize the benefits of PropGen. It is important to note that the key focus of PropGen is to capture the intent of the FSM design specifications, regardless of the specific tool used for input.

The Visio-to-SVA conversion is one example presented in the paper. The XLS option mentioned above enable users to achieve the same goal even if Visio is not available. Visio is the preferred tool of choice in FSM use case of PropGen to generate SVA directly from specifications but when XLS format is used, the FSM specification is fed manually. Hence, design and verification teams need to carefully review to ensure the accuracy of the updated information. Once the XLS details are entered and reviewed, PropGen seamlessly integrates with the updated specifications.

IV. Issues Faced In Porting Assertions From FV To IP Simulations

PropGen was applied to verify FSM sub-blocks of a complex IP used in AI/ML applications. These FSM sub-blocks were fully verified and sign-off was done using standalone FV. There were no constraints applied leveraging full randomness inherent in FV. DV of many such IP sub-blocks was signed-off using FV with 100% formal code coverage. Later FV assertions were ported to IP level simulations as end-to-end data path integrity checks were signed-off using simulations. This hybrid DV approach could catch corner cases bugs using constrained random FV and IP level checks using simulation TB leveraging advantages of both the methods. While porting assertions from FV to simulations, there were few lessons learned and they are summarised below.

A. Clock gating in simulations at IP level

One such issue encountered were the assertions to check outputs after asynchronous reset, which passed in sub-block-level FV but could not be triggered in IP level simulations which integrated many different sub-blocks. This discrepancy was attributed to clock's gating done at the IP level, when async_rst is asserted at the primary input of IP as shown in Figure 4. This clock gating mechanism caused the assertions to remain in the inactive assertion state in simulations as clocks were shut-off.

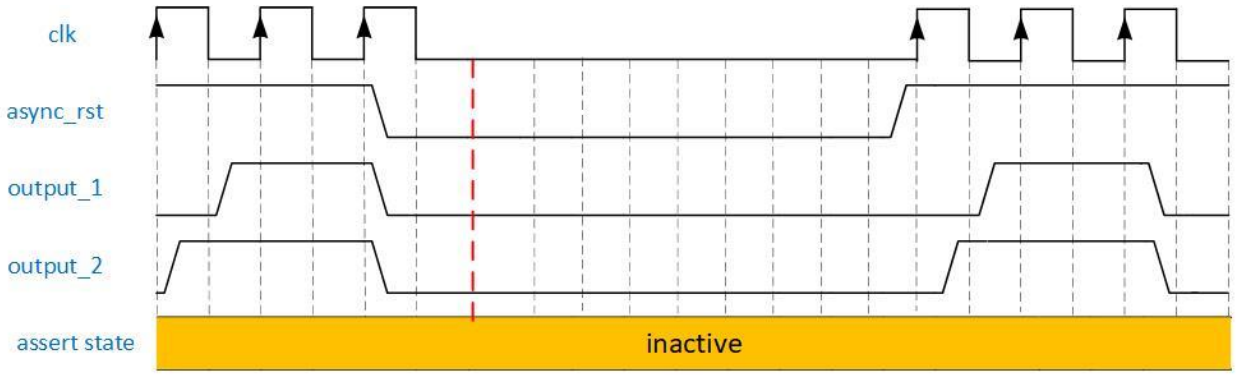


Figure 4. Async Reset check in Simulation.

One such simple property is shown below to check for outputs after asynchronous reset is applied and FV exhaustively checks all possible combinations.

```
property output_reset;
        @(posedge Clk)   (!async_rst) |-> (!output_1) && (!output_2);
endproperty
```

FV typically employs a free-running clock as shown in figure 5, while IP simulations had clock gating after the asynchronous reset as part of design implementation. Clock gating is implementation-specific and may introduce a delay of 2-3 clock cycles before shutting down the clock. This discrepancy poses a challenge as the intention of FV is to fully randomize and explore all scenarios at individual sub-block level and not worry about design implementation. Additionally, FV started early in the design cycle when some IP level implementation details were not yet available. Clock gate modelling for async reset could have been done at FV level later, but it would have taken considerable effort and would also mask randomness which is the main motivation to use FV.

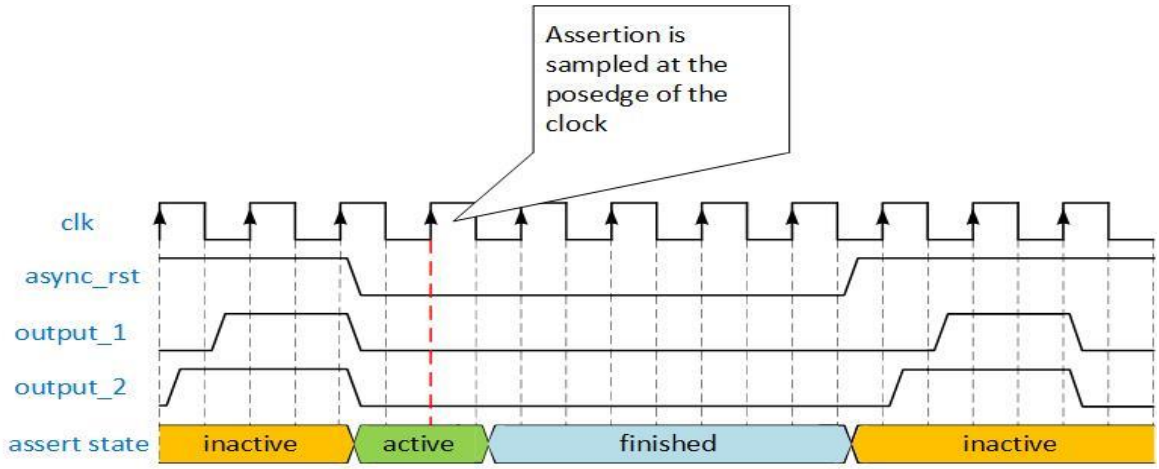


Figure 5. Aysnc Reset check in FV.

To address the above discrepancy, assertions were selectively removed from simulations using the built-in SV $assertoff() feature [4]. $assertoff () allowed selected assertions to be disabled. There were only handful of such assertions in single digit number that needed this waiver in simulations. However, this approach should be used judiciously to ensure that critical assertions are not omitted, and the necessary verification coverage is maintained. One such sample $assertoff() is mentioned below.

$assertoff (0, dut_top.<assertion binding_hierarchy path>).

B. Black boxing of Multipliers

In general, Multipliers are often black boxed in FV tools due to their complex, resource-intensive nature and can significantly increase the runtime of FV. This can lead to state space explosion. By black boxing multipliers, the FV tool can focus on verifying the surrounding logic rather than internal details of the multiplier.

One of the FSM assertions sets was impacted due to multiplier auto black boxing. There were many multipliers that impacted the design logic and FV properties were failing. Detailed analysis from FV traces could not find the root cause of failure to multipliers black boxing as random values were getting driven at multiplier output. When same assertion was tried in simulations it passed as multipliers were NOT getting black boxed in simulations. Although there was a WARNING message issued by FV tool about multipliers black boxing it got ignored due to numerous such WARNING messages. Warning messages from FV tool need to be carefully reviewed before final waive off as they can point to such potential issues.

In many cases, the exact implementation of a multiplier may be relevant to the verification task at hand. Hence, in situations where critical verification is blocked due to the black boxing of multipliers in FV tools, auto black boxing of multipliers needs to be removed using tool-specific commands or settings.

Overall, porting assertions from FV to simulations requires careful consideration of the differences in clocking behavior, asynchronous reset handling, and the need for selective assertion removal. It highlights the importance of addressing implementation-specific details and ensuring that the assertions effectively capture the desired verification objectives in both FV and simulation environments.

V. RESULTS

The effectiveness of PropGen to verify FSM was evaluated on two different IPs, the first IP being an AI/ML-based accelerator, and the second IP was used in voltage regulation for multiple CPU. PropGen was able to generate a considerable number of properties for each FSM, as mentioned in table 1. The required time for generating properties and preparing the first FV report was only 0.5 days for each FSM. Manual property writing would have taken ~10 days of efforts directly translating to ~90% of efforts saving per FSM. This demonstrates the quick turnaround time facilitated by PropGen, enabling quick verification iterations for FSM changes, and reducing the overall design verification cycle.

Table 1. FSM FV Results with PropGen

	Design For which PropGen Tested			
	IP 1 (3FSMs)			**IP 2 (1 FSM)**
	FSM1	**FSM2**	**FSM3**	
Total Properties generated by PropGen	103	48	29	97
Total time required for first FV report	0.5 Days	0.5 Days	0.5 Days	0.5 Days
Formal Code coverage achieved	100%	100%	100%	100%
Properties Portable to Simulations	YES	YES	YES	YES
Property re-generation time for design change	0.25 days	0.25 days	0.25 days	0.25 days

VI. CONCLUSION

PropGen emerged as a powerful and versatile methodology for the verification of complex designs, as demonstrated by its successful application and results obtained. DV efforts can be reduced by ~90% for every FSM in the design. PropGen significantly enhances the efficiency and effectiveness of the design verification process, with its ability to generate SVA properties towards achieving 100% code coverage, ensuring the portability of SVA properties to simulations and swiftly adapting to design changes. Its automation capabilities make it a novel methodology for future design verification endeavors.

ACKNOWLEDGMENT

The authors would like to extend sincere thanks to Vidya Sagar Kantamneni and other team members for their valuable support.

REFERENCES

[1] Smith, A., et al. "Accelerating Design Verification through Automated Techniques." IEEE Transactions on VLSI Systems, vol. 42, no. 2, pp. 245-256.
[2] Johnson, B., and Lee, C. "Efficiency Improvement in Design Verification through Automation." IEEE Design & Test of Computers, vol. 39, no. 4, pp. 78-84.
[3] A. Shambu, V. Dalal, B. Naik and S. M, "Automation of FSM Verification Using Formal Tools," 2023 International Conference on Communication, Circuits, and Systems (IC3S), BHUBANESWAR, India, 2023, pp. 1-5. DOI: 10.1109/IC3S57698.2023.10169571.
[4] 1800-2017 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language. IEEE, 2017.
[5] Chen, Q., Wang, L., & Li, Z.,"Formal Verification of ASICs: Challenges and Opportunities". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(5), 987-1001, 2023.
[6] S. Johnson, B. Anderson, "Efficient Verification Strategies for Complex ASIC Designs." IEEE Transactions on VLSI Systems, vol. 30, no. 5, pp. 987-1001 2022.