2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

AI-Enabled Formal Verification Flow : From Spec to Sign-off

Rinu Mathew
Design Verification
SmartSoC Solutions
rinu.mathew@smartsocs.com

***Abstract*-Formal verification's adoption is hindered by complexity and scalability issues, limiting its use to small design blocks and creating a gap for complex systems. This paper proposes an AI-enabled formal verification approach, integrating Artificial Intelligence (AI) into every step, from property extraction to proof convergence, leveraging Large Language Models (LLMs) for automated property generation, abstract model creation, and building a comprehensive database of verification information. AI intelligently classifies, sorts, and reuses properties to reduce complexity, enhancing formal property verification (FPV) and datapath validation (DPV) through design decomposition and formal engine orchestration. Convergence techniques such as abstraction and property decomposition are employed to address state-space explosion, ensuring faster and more accurate verification. Comprehensive verification is ensured through analysis of coverage metrics, including cone-of-influence, formal core, and testbench analysis, guaranteeing thorough verification before sign-off. AI-driven analytics and detailed reporting simplify debugging and design optimization, while the systematic flow improves accuracy, scalability, and productivity, ultimately achieving faster, confident signoffs.**

Keywords— Formal Verification, Signoff, AI, Convergence, Datapath Validation

I. Introduction

The increasing complexity of digital systems has made it crucial to ensure that designs are correct and reliable. However, traditional simulation-based verification methods are no longer sufficient for testing complex designs. These methods require a large number of test cases, are time-consuming, and provide limited coverage, which can lead to errors and costly re-spins [2].

Formal verification, on the other hand, provides a mathematical proof that a design is correct, eliminating bugs and errors. This approach offers several benefits, including exhaustive coverage, mathematical certainty, reduced debugging time, and improved design quality [1]. However, formal verification has its own set of challenges.

One of the main limitations of formal verification is state-space explosion, where the number of possible states in a design becomes too large to analyse efficiently. Additionally, formal verification often requires complex properties to be specified, which can be difficult to define and verify [4]. Convergence issues can also arise, where the verification process fails to terminate or produces inconclusive results.

To overcome the limitations of formal verification, such as state-space explosion, complexity of properties, and convergence issues, Artificial Intelligence (AI) can be leveraged to enhance the formal verification process.

Artificial Intelligence (AI) refers to the development of computer systems that can perform tasks that typically require human intelligence, such as learning, problem-solving, and decision-making. In the context of formal verification, AI can be used to analyse complex designs, identify patterns, and generate properties that would be difficult for humans to create.

By integrating AI into every step of formal verification, a more efficient, faster, and automated process can be achieved. This AI-driven approach enables the verification team to achieve sign-off level confidence, ultimately ensuring that complex digital systems are correct, reliable, and error-free.

II. Property Extraction

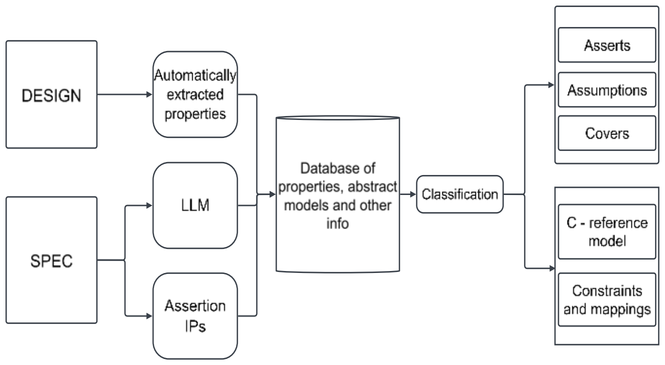

Fig. 1. Extraction and classification of properties

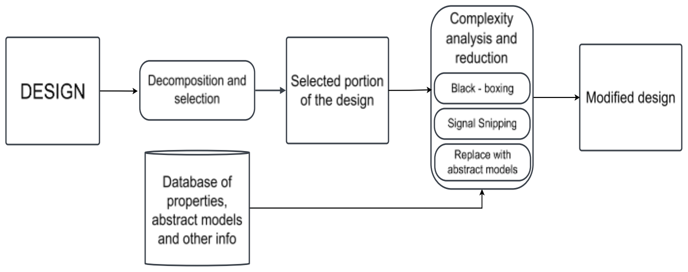
DESIGN
Decomposition and selection
Selected portion of the design
Complexity analysis and reduction
Black - boxing
Signal Snipping
Replace with abstract models
Modified design
Database of properties, abstract models and other info

Fig. 2. Reduction of design complexity

A. *LLM based*

LLMs like ChatGPT have become immensely popular for generating texts. Recent advancements have brought LLMs to the field of EDA such as for RTL code generation and syntax correction. Many researches are already advancing towards the use of LLMs for property generation [3]. It is only a matter of time before these tools would be able to generate quite accurate properties from a proper specification written in human language. LLMs could also generate high level abstract models such as C-models and behavioural models of the design from the specification provided. This process, however heavily depends upon the expectation that the specification would be accurate.

B. *Automatic extraction of properties*

Many EDA vendors already provide the facility to extract properties from designs. Synopsys' VC Formal Automatic Extracted Properties (AEP) is one such tool which has showcased immense capabilities in this realm [4].

C. Assertion IPs

Assertion-based verification IPs (Assertion IPs) for standard protocols are commonly included in formal EDA tools and also offered by specialized IP vendors. These pre-verified IPs enable design teams to efficiently verify protocol compliance, ensuring their digital systems meet specific standards and regulations. By leveraging these Assertion IPs, teams can streamline their verification process, reduce errors, and improve overall design quality.

D. *Database of Properties, Abstract models and other info*

Abstract models and Properties generated or extracted by the afore-mentioned methods are stored in a database from where they could be easily accessed and modified as per need, as shown in Fig. 1. It would also contain other information such as signal mappings, clocks, resets etc.

III. Classification of properties

Once the database of properties is ready, the next step would be to classify these properties based upon whether they would be used as Asserts, Assumptions or Cover properties.

These properties would also need to be ranked based upon their complexity and span for incremental verification. The classification would not be a strict one. Initially, the AI may classify a few properties as assertions, a few as assumptions or constraints and a few as covers, based upon how it was trained. The AI, based upon its learning may change the classification to accelerate the verification process and to make it more accurate.

For example, a property used as an assertion for the formal verification of a particular block of the design may be used as an assumption for the verification of another block, directly dependent upon the previous one so as to limit the state space and reduce complexity.

The classification needs to be extended to the abstract models as well depending upon whether they need to be used to abstract portions of the design to reduce complexity in the FPV exercise or to be used as reference models in the exercise of DPV.

IV. Design Verification

After organizing the properties and models, the process can advance to the formal verification of the design. Nevertheless, it is often impractical to verify the entire design simultaneously, especially during the initial stages. The design needs to be decomposed to smaller units efficiently. AI could analyse the design and decompose it in the most efficient manner.

Although there are many subsets of formal verification, This paper will focus on two specific subsets of formal verification: Formal Property Verification (FPV) and Data-Path Validation (DPV), which are within the scope of this discussion. The design could be too complex for these exercises even after decomposition. It could lead to the properties not converging or the being concluded as bounded proofs. Therefore, the design needs to be intelligently abstracted as intended in Fig. 2. Formal engineers have found many methods to abstract the design over the years[5]. These include :

a) Blackboxing : A portion of the design is, in effect, excluded from verification. Modern tools are able to automatically blackbox complex yet non- critical entities. However, the AI should be able to accurately detect these entities for efficient blackboxing. It should also be able to properly constraint the outputs of these entities going into other entities, so as to prevent state-space explosion downstream.

b) Snipping signals : Snipping the signals from their drivers is another way to ignore complex logic irrelevant to the verification targets. The selected signals are snipped from their respective drivers and driven independently by the tool. This process necessitates careful signal identification and the application of suitable constraints by the AI algorithm to ensure accurate verification results.

c) Replace with abstract models: Parts of the design under test, having complex logic, yet non-critical to the current objectives could be replaced with less computation-intensive models from the database or models previously verified in the DPV exercise or with previously proven assertions converted into assumptions.

Abstracting entities may not be necessary in the first iteration, but the AI algorithm should be able to perform this efficiently as the verification flow progresses through subsequent iterations.

A. Formal Property Verification

Formal Property verification is an exhaustive check of the design against the specification defined in terms of properties which are asserted to be always true [1]. Once the design is modified as per the objectives of the current run, AI selects a number of assertions, assumptions and covers relevant to the design under and runs them in the FPV tool . The properties used as assumptions constraint the values that the inputs can be assigned with. The tool uses mathematical methods to try and falsify the asserted properties. If they can be falsified, a counter-example is issued demonstrating one of the ways the property can be falsified. The tool issues the status of full proofs for properties that have been proven to hold true always, bounded proofs for properties that have held true up to a certain depth and inconclusive for properties that cannot reach a conclusive result [4], as shown in Fig. 3.

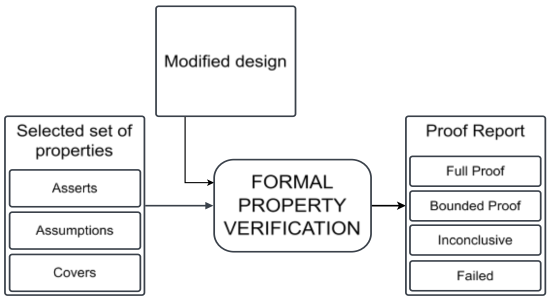

Fig. 3. Formal Property Verification

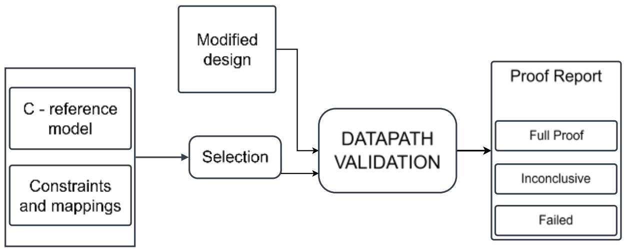

Fig. 4. Datapath Validation

B. Datapath Validation

Those parts of design that are computationally intensive, such as AI, graphics or math cores cannot easily be verified using traditional FPV efforts. These can much more efficiently be verified by the datapath validation methodology that checks a transactional equivalence between a reference model, usually written in C++ against the design [6]. The AI algorithm should be able to identify those parts of the design which can be better evaluated using DPV and select the relevant reference model from the database, map the inputs and outputs between the spec and the design and provide necessary constraints. The tool will then apply input stimulus to both the C++ spec and the RTL design and compare their outputs and report the proof results. This is illustrated in Fig. 4.

V. Engine Orchestration

Since formal verification is a heavily tool-dependent methodology, tuning the formal verification tool to right parameters plays a vital role in the quality of the proof results and the time and effort required for proof convergence. Selecting the right formal engines, setting the right time to run, effective utilization of resources and tuning the engine in the most appropriate way for each run are some of the tasks that are critical for tool performance and requiring good skills and experience on the part of the engineer at the same time. It is also one of those areas where AI can completely remove the need for human intervention and do a much better job. In this case, the AI/ML algorithm would need to tune the formal engines and tools to the precise configurations for maximum throughput and minimum time depending upon the tasks at hand in every run [7].

VI. Convergence Efforts

Formal verification, being an exhaustive check of the design is a computationally intensive process. Due to the complexity of design and properties, the state space of the proofs grow exponentially with increasing depth, causing the proofs to not be able to converge. AI can be used to utilise various convergence techniques at the appropriate places efficiently.

a) Abstraction: As discussed earlier, abstraction techniques can be used to reduce the design complexity dramatically, thus limiting state-space in the cone-of-influence of the properties [8].

b) Property Decomposition and exclusion: Just like with designs, complex properties can be broken down into multiple simpler properties, reducing the computational effort required to prove each of the assertions [8] . Properties whose COI has already been covered by other properties may not be necessary to be evaluated, saving precious time and effort.

c) Helper Assertions and Constraints: AI algorithms can be used to constrain the COI of the proofs and strategically explore the state space to focus on specific areas using helper assertions [5].

d) Assume Guarantee Reasoning: Certain assumptions can be proven about parts of the design, guaranteeing the proofs of other parts [5].

e) Simulation: AI can use simulation as a last resort to efficiently steer the verification flow from a point at high risk of state-space explosion to one where the state space will be limited.

VII. Formal Signoff

Signing off a formal verification effort needs to meet certain metrics. These are similar to the coverage criteria in traditional simulation-based verification. Formal tools are available for signoff analysis and it dictates the quantity and quality of the results of verification process [9].

A. Over-Constraint Analysis

If the stimulus has been over-constrained, it means that the entire state-space has not been explored. Properties could have been over-constrained previously in an attempt to converge certain properties. The verification process must intelligently adapt its assumptions and re-run the analysis to encompass all possible states.

B. Cone-of-influence

The Cone-of-influence (COI) is the collection of the entire logic in a design that is affected by a property. For Formal Signoff, it is imperative that every element in a design fall under the COI of some property [4].

C. Formal Core

The formal core is a subset of the COI of a property that is actually tested by the assertion. Analysis of the formal core gives more confidence regarding the completeness of the verification [4].

D. Formal Testbench Analysis

Testbench analysis is a novel idea that gives even higher confidence in the verification efforts by injecting errors into the design to test their capturability by the formal testbench [9]. EDA vendors already provide tools curated just for testbench analysis, providing utmost confidence for signoff [4].

Ideally, the COI, Formal Core and testbench analysis should give 100% coverage for verification Signoff.
The AI algorithm should put efforts to improve coverage after each run. It could be done by analysing the spec and database of properties and models again, updating the database with new properties, adding and modifying new properties and models and altering the constraints in the current test run etc [10].

VIII. Verification report

Generative AI could be adopted for compiling the proof reports from the tools, the coverage reports and other information in a detailed and easy to understand way for engineers to easily debug and fix design errors for total signoff.

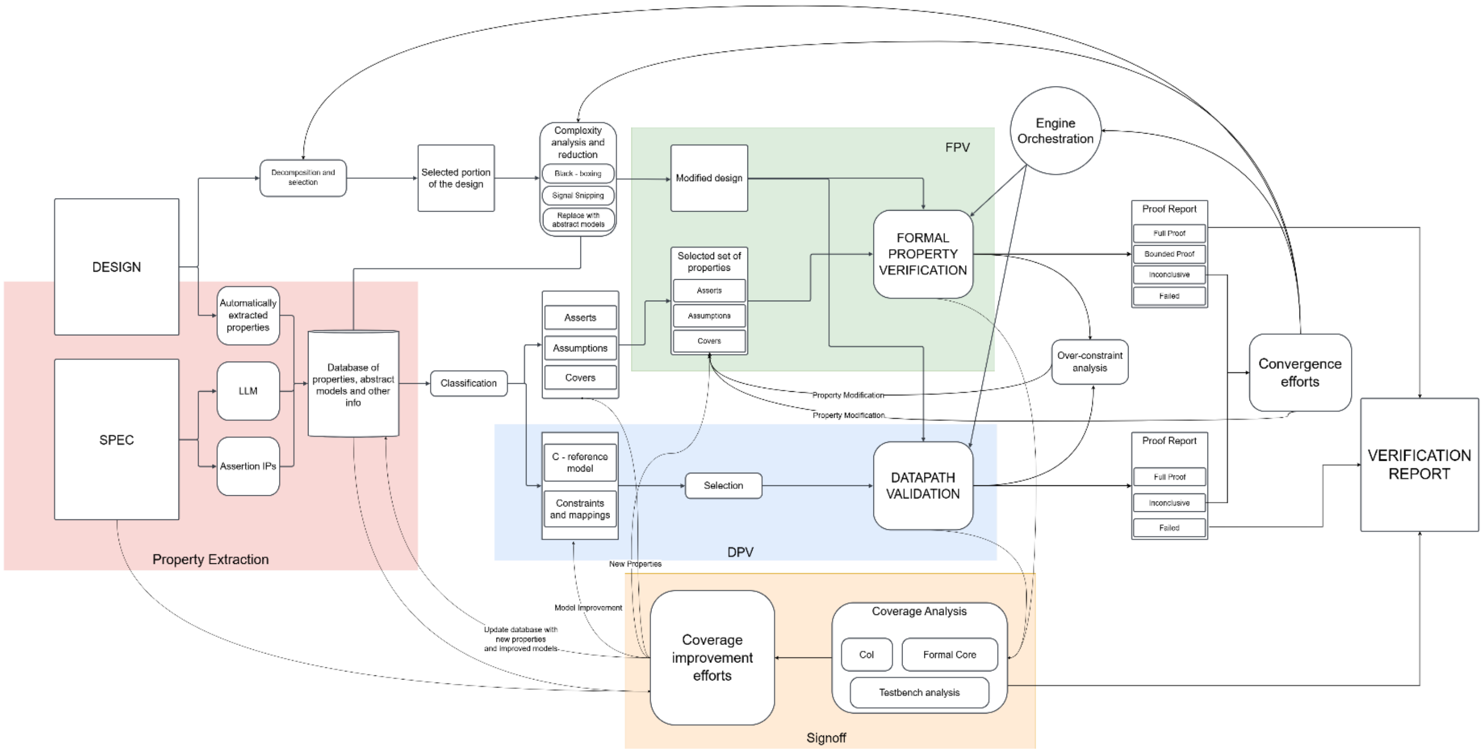

Fig. 5. Proposed AI-Enabled Formal Verification Flow.

IX. PROPOSED FLOW: BLOCK DIAGRAM

The proposed flow is illustrated in the block diagram shown in Fig. 5. It is implied that intelligent automation, leveraging machine learning and artificial intelligence techniques, is integrated throughout each step of the flow, as detailed in this paper.

X. CONCLUSION

This paper proposes a formal verification flow, integrating AI to minimise human efforts and need for skill in every step, focusing mainly on two subsets of formal verification, viz. FPV and DPV. It can be extended to other formal verification applications as well in the future. Since use of AI for generation of RTL code is already being extensively researched [11], it could be integrated with the formal verification flow for extreme left-shift in the front-end of the chip development cycle. AI can also be leveraged to extend the flow to include netlist creation and formal sequential equivalence comparison with RTL, lint checks, connectivity checks, low power verification etc. This could mean, in the future, that the design rolls out to the backend in just a matter of days.

Although the implementation of this flow might require extensive human intervention in the early days, especially in areas such as property generation, design decomposition and selection and proof checking, we expect it to mature relatively quickly. Anyway, we believe this step would be a milestone in removing the fear of formal verification and enable its widespread adoption by verification teams.

REFERENCES

[1] E. Seligman, T. Schubert, and A. Kiran, Formal Verification. Elsevier, 2023.

[2] “Formal Verification vs. Functional Verification,” VLSI Worlds, Dec. 26, 2024. https://vlsiworlds.com/2024/12/26/formal-verification-vs-functional-verification/ (accessed Jan. 18, 2025).

[3] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, “Using LLMs to Facilitate Formal Verification of RTL,” arXiv (Cornell University), Jan. 2023, doi: https://doi.org/10.48550/arxiv.2309.09437.

[4] S. Ganesh, “A Blueprint for Formal Verification - systemverilog.io,” Systemverilog.io, 2015. https://www.systemverilog.io/verification/blueprint-for-formal-verification/

[5] P. Yu, “Understanding and Managing Complexity in Formal Verification,” Forum for Electronics, Jul. 18, 2024. https://www.edaboard.com/blog/understanding-and-managing-complexity-in-formal-verification.2291/

[6] T. Drane, “Formal Verification and Validation of High-Level Optimizations of Arithmetic Datapath Blocks,” Academia.edu, 2024. https://www.academia.edu/26351213/Formal_Verification_and_Validation_of_High_Level_Optimizations_of_Arithmetic_Datapath_Blocks

[7] E. M. Elmandouh and A. G. Wassal, “Guiding Formal Verification Orchestration Using Machine Learning Methods,” ACM Transactions on Design Automation of Electronic Systems, vol. 23, no. 5, pp. 1–33, Oct. 2018, doi: https://doi.org/10.1145/3224206.

[8] D. Puri, “Revolutionizing Proof Convergence for Algorithmic Designs: Combining Multiple Formal Verification Tools.” Accessed: Jan. 18, 2025. [Online]. Available: https://dvcon-proceedings.org/wp-content/uploads/2A1_DVCon_India_2023_Final_Paper_628.pdf

[9] M. Parmar, I. Singleton, and G. Jacob, “The Importance of Complete Signoff Methodology for Formal Verification.” Accessed: Jan. 18, 2025. [Online]. Available: https://dvcon-proceedings.org/wp-content/uploads/the-importance-of-complete-signoff-methodology-for-formal-verification.pdf.

[10] X. Feng, X. Chen, and A. Muchandikar, “Coverage Models for Formal Verification.” Available: https://dvcon-proceedings.org/wp-content/uploads/coverage-models-for-formal-verification.pdf

[11] S. Subramanyam, “AI-Driven RTL Generation from Schematic Images: Revolutionizing IC Design.” Accessed: Jan. 18, 2025. [Online]. Available: https://ijfmr.com/papers/2024/6/30939.pdf