2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Analogous Alignments: Digital "Formally" meets Analog

Hansa Mohanty, Infineon Technologies, Dresden, Germany (*hansa.mohanty@infineon.com*)

Deepak Narayan Gadde, Infineon Technologies, Dresden, Germany (*deepak.gadde@infineon.com*)

***Abstract*—The complexity of modern-day System-on-Chips (SoCs) is continually increasing, and it becomes increasingly challenging to deliver dependable and credible chips in a short time-to-market. Especially, in the case of test chips, where the aim is to study the feasibility of the design, time is a crucial factor. Pre-silicon functional verification is one of the main contributors that makes up a large portion of the product development cycle [1]. Verification engineers often loosely verify test chips that turn out to be non-functional on the silicon, ultimately resulting in expensive re-spins. To left-shift the verification efforts, formal verification is a powerful methodology that aims to exhaustively verify designs, giving better confidence in the overall quality. This paper focuses on the pragmatic formal verification of a mixed signal Intellectual Property (IP) that has a combination of digital and analog blocks. This paper discusses a novel approach of including the analog behavioral model into the formal verification setup. Digital and Analog Mixed-Signal (AMS) designs, which are fundamentally different in nature, are integrated seamlessly in a formal verification setup, a concept that can be referred to as "Analogous Alignments". Our formal setup leverages powerful formal techniques such as Formal Property Verification (FPV), Control/Status Register (CSR) verification, and connectivity checks. The properties used for FPV are auto-generated using a metamodeling framework [2]. The paper also discusses the challenges faced especially related to state-space explosion, non-compatibility of formal with AMS models, and techniques to mitigate them such as k-induction. With this verification approach, we were able to exhaustively verify the design within a reasonable time and with sufficient coverage. We also reported several bugs at an early stage, making the complete design verification process iterative and effective.**

***Keywords* —SOCs, IP, FPV, CSR, AMS**

I. Introduction

Verification has become the bottleneck in product development cycles, as it takes more than 60 % of the overall project time [1]. In contemporary electronic systems, analog circuits are integral components. The engineering community faces increasing challenges as the time-to-market shrinks amid the increasing complexity of mixed-signal designs. A critical strategy to optimize the verification process is simulation, employing a hierarchy of circuit models that vary in abstraction. The introduction of a behavioral model for AMS simulation represents a notable abstraction method. While this model is advantageous for simulation-based verification, particularly with respect to assessing the functionality of digital components, its integration into formal verification workflow can be difficult. Formal verification employs a mathematical approach to ascertain the correctness of a design. Traditionally, this verification requires the specifier to possess an understanding of property formulation in system verilog assertions. The manual implementation of these properties leads to certain challenges. Not only is it prone to human error, but it can also be monotonous and laborious, especially when dealing with extensive state spaces. To mitigate these issues, properties have been auto-generated utilizing a Metamodeling framework. Metamodeling serves as an automation framework designed to reduce Non-Recurring Engineering (NRE) costs. This framework leverages Unified Modelling Language (UML) diagrams, Python, and Mako templates to create diverse outputs across various programming languages. This paper delves into the details of automating property generation. Formal verification plays a crucial role in validating hardware circuits and in revealing numerous minor yet significant features, such as x-propagation, register, and connectivity issues. However, its effectiveness is diminished when applied to verifying AMS circuits. In the context of digital verification, the analog components primarily function to deliver control and status signals, which collectively ensure the correct operation of the IC. During digital simulations, the performance of the analog circuitry is not the focal point. In this paper, we provide insights into two methods firstly assuming the ideal behavior of the AMS behavioral model and the secondly a novel method that incorporates a formal-friendly analog model, which is conducive to the smooth integration into formal verification frameworks.

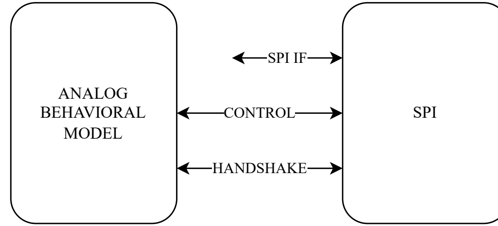

Fig. 1: Abstract level overview of the design

II. Design Overview

The IP consists of a Serial Peripheral Interface (SPI) that communicates with an AMS model as shown in Fig. 1. In practical applications, the AMS circuit is composed of a memory unit engineered to permit a single write operation followed by the capability for multiple read accesses. The constructed analog behavioral model, designated for digital simulation purposes, plays a crucial role in the transmission of specific control and handshake signals. These signals include a 'ready' indicator, which signifies that the analog circuitry is prepared to initiate the subsequent transaction and a 'busy' signal, which is asserted during the course of an active transaction. Furthermore, the Serial Peripheral Interface (SPI), which incorporates a register bank, serves as a conduit for serial inputs. These inputs are then channeled to the analog behavioral model. Concurrently, the values corresponding to various status signals, which originate from the AMS behavioral model, are accessible through a designated register within the register bank. This systematic approach ensures a seamless interface between the digital and analog components of the AMS system, facilitating efficient communication and operational synchronization. The principal objective of the Intellectual Property (IP) under examination is to archive values received from various components of the System-on-Chip (SoC). The iteration of the design presented here was realized as a test chip, with the primary intent of evaluating potential reductions in the area occupied by the analog circuitry. A fully equipped functional IP is implemented for the tasks of data rectification and autonomous data integrity checks. This comprehensive functional IP is equipped with both Single Error Correction Double Error Detection (SECDED) and Double Error Correction Triple Error Detection (DECTED). It should be noted that the complete formal verification of this robust IP infrastructure is anticipated as a subject of future investigation and is beyond the scope of the current paper.

III. Metamodel Framework

A Metamodeling framework is an automation framework where a model is used to represent a system in a certain abstraction level. A metamodel is utilized to define the structure of a model and the relationships between its constituents. Metamodels extend beyond models and are used to represent models of models. Metamodeling serves as an automation framework designed to decrease Non-Recurring Engineering (NRE) costs. This framework utilizes Unified Modelling Language (UML) diagrams, Python, and Mako templates to generate outputs in diverse programming languages. Metamodeling is rooted in the principles of Model Driven Architecture (MDA), which advocates for the use of modeling techniques to enhance productivity and abstraction levels during the development process. The different abstraction levels used in MDA are: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). In the first step the specification is added to the framework. The specification is generally added in the form of a UML diagram. The framework parser goes through the UML diagram and generates a Python-based Application Programming Interface (APIs). The APIs have been generated automatically, including several methods, such as setters(), getters(), and other frequently used methods, making it simpler to access the metamodel. The design definition is then entered using the Metamodel Graphical User Interface (GUI), which is also generated as part of the Python and Mako templates to framework after the APIs have been created. Data generated by the Metamodeling GUI is usually in an Extensible Markup Language (XML) format. By using the API methods, the reader reads the specifications from the XML file, and the writer can produce viewcode in a language of their choice [2]. This paper presents a application of the Metamodeling framework to facilitate the generation of SVA, a critical component in the verification of hardware designs. Our approach begins with the construction of a UML class diagram, meticulously detailing the interconnections and relationships amongst system components. This abstraction level encapsulates a high-level view of the system, integrating essential elements such as registers, bitfields, access policies, and reset values. Subsequently, we transmute this CIM representation into an XML file, affording us a versatile and universally compatible format that seamlessly feeds into the successive layer of abstraction. Advancing to the PIM, the register and other specifications are converted into an XML file. Consequently advancing to PSM level of abstraction, the XML file is converted to a PIM.xml, file and we define a

robust 'property class' which is otherwise known as a MAKO template, laying the groundwork for the automated generation of three crucial types of SystemVerilog assertions:

- Register read-write operations, functionality check between the SPI and register bank,
- Daisy chain mechanisms within the circuitry.
- Communication protocols between the AMS circuits and the SPI

Each category of assertion plays a pivotal role in ensuring the integrity and correctness of digital and analog interactions. Our methodology not only expedites the verification process but also enhances its accuracy, thereby contributing to the reliability and efficiency of hardware design verification.

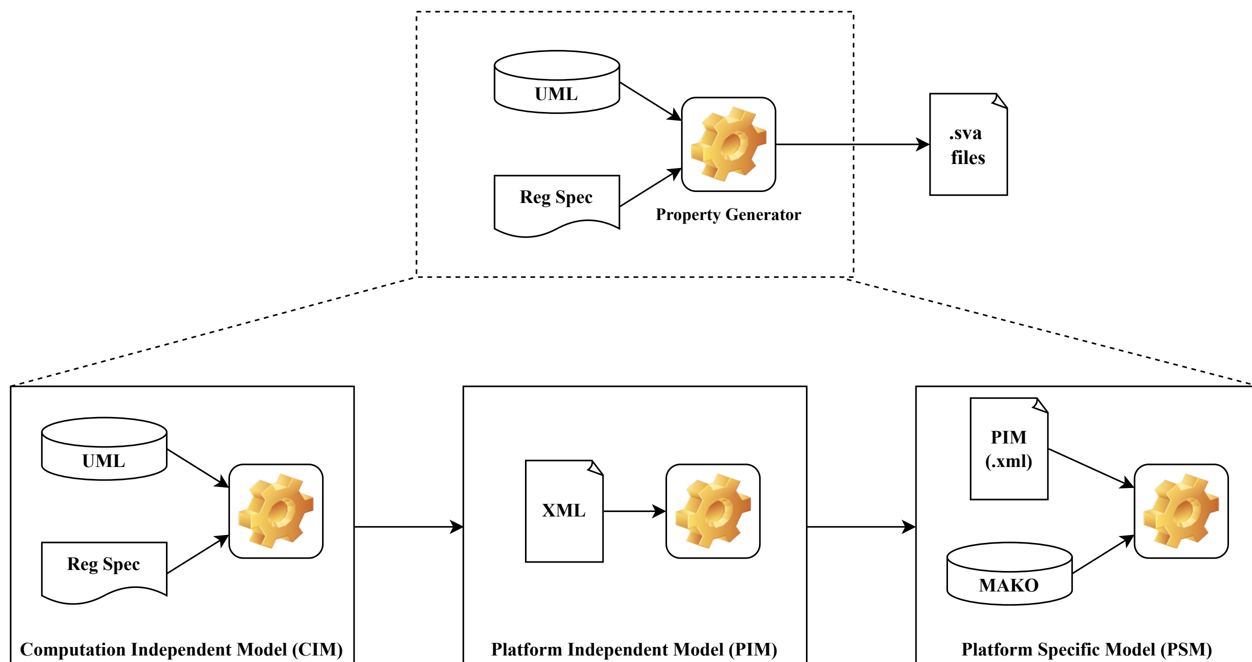

Fig. 2: Metamodel framework for SVA property generation

IV. Design Verification

The IP described in this paper is utilized within the realm of automotive applications. The digital components of these IPs, as delineated in Section II, present themselves as prime candidates for leveraging Formal Verification methodologies. Nonetheless, the incorporation of AMS components presents a formidable challenge to the seamless adoption of these verification techniques into our established workflow. The core of the challenge lies in the inherent complexity and non-synthesizability of AMS circuitry, which resists straightforward formal analysis. In light of this, our experiment bifurcates into two distinct investigative pathways. The initial pathway operates under the assumption that the analog components behave in an idealized manner—an assumption that simplifies the verification process but may not hold up against the details and anomalies of real-world applications. In this pathway, we also incorporate the use of additional Jasper applications during the bring-up phase, broadening the horizon of formal verification beyond its traditional routes. These applications, which extend to functions like register checks and connectivity validations, are not only user-friendly but also effective in early bug detection and ease of debugging. The second investigative pathway, which is more forward-looking, endeavors to assimilate the analog behavioral model, complete with its non-ideal characteristics, into the formal verification framework. This approach acknowledges the limitations of the ideal behavior assumption, particularly as the complexity of AMS circuits escalates. Subsequent sub-sections will expound upon the methodologies and findings associated with each investigative pathway, providing a comprehensive analysis and comparison of their respective efficacies and applicabilities in the context of automotive IP verification. Through this exploration, we aim to illuminate the path towards more robust and comprehensive formal verification strategies that can accommodate the nuanced behaviors of AMS circuitry. Both approaches are described in the following sections.

A. *Formal Verification*

Formal verification uses technologies that mathematically analyze the space of possible behaviors of a design rather than computing results for particular values [3]. It is an exhaustive verification technique that uses mathematical proof methods to verify whether the design implementation matches design specifications [4].

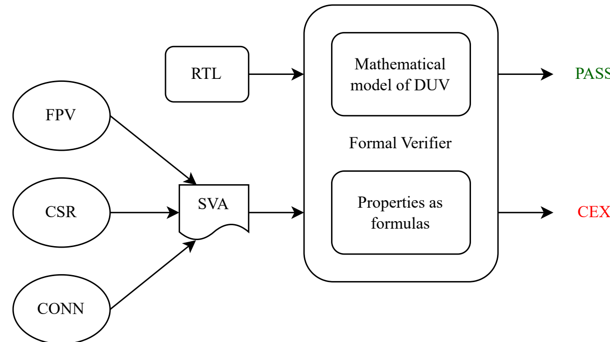

Fig. 3: Formal verification

Fig. 3 shows the working of a formal verifier. There are two inputs to the formal verifier tool. On the one hand, the Design Under Verification (DUV) is fed into the tool which is converted into a mathematical model. On the other hand, properties, written in SystemVerilog Assertions (SVA) that capture the intent of the design are fed into the tool. The tool then converts these properties into mathematical formulas. In the next step, the tool tries to prove these mathematical formulas on the mathematical model of the DUV. If the properties do not hold, the tool announces, and a Counter Example (CEX) is generated by the tool to further debug. In general, the absence of a CEX is nothing but a pass or proven result [5]. To meet different verification requirements, we used different formal verification techniques and apps from Cadence Jasper as shown in Fig 3. The following sub-sections provide an overview of the techniques used.

1) *Control/Status Register Verification*

The CSR verification app from Cadence Jasper offers a pragmatic solution for verifying the registers in the DUV. It is especially beneficial at the early stage of the project when the UVM testbench is not fully functional, yet we need to verify the correct behavior of the registers such as the register access policies, reset values of register fields and basic register write reads. The register description is prepared by the concept engineer and is used to generate the Comma Separated Values (CSV) file, which is an input to the formal tool. Since this approach is fully automated as a push-button solution, it also helps design engineers verify their RTL for every register change before releasing the next version of the RTL. The DUV mentioned in this paper has several registers that have varied policies therefore a preliminary check gives confidence over the design. A major benefit of using a formal based approach for register verification is the ease of debugging. The CEXs in case of a failure are usually within 10 clock cycles which helps to identify the root cause easily. However, there are also limitations associated with this approach. Usually, the properties generated by the tool are encrypted and cannot be accessed. Additionally, custom register schemes that are specific to the project cannot be verified with this approach as the properties are only generated for standard register types.

2) *Connectivity Verification*

The CONN application from Cadence Jasper is a useful tool for checking connections in circuits, specifically designed for both simple and conditional links. This tool is part of the formal verification process, known for its thoroughness, ensuring that no connection is overlooked. In our project, we paid special attention to the links between the analog parts (like sensors) and digital parts (like processors) of our design. We made sure that these connections were properly established from start to finish. The connectivity checker within the Jasper suite is a program that helps us do this automatically. It creates a list of checks that need to be made based on information we provided in a table. This table is straightforward — it lists where the signal starts (the source block), the name of the signal, where the signal is going (the destination block), the name of the signal there, and any special conditions that must be met for the connection to work. Once we fill out this table, we implement a tcl script, which is a small computer program that uses the table to check everything automatically with the Jasper tool. We can also do the checks backward using a feature where the app figures out the connections and fills in the table for us. This

is especially helpful to double-check our work and make sure it matches the original plans given by the concept engineer.

3) Formal Property Verification

The Jasper application is a tried-and-true method for ensuring that computer systems perform their functions correctly through a formal verification process. This particular application stands out because it requires users to have a foundational understanding of SVA, which are essentially rules that guide the verification process. To operate this tool, users must provide three essential components: the SVA specification written in the form of properties, a control file (TCL file), and the hardware design tested. The application boasts substantial adaptability, making it suitable for a range of industrial applications, regardless of their scale. An impressive feature of this Jasper tool is its ability to allow users discretion in selecting different solver engines for evaluating each specification/formula. These engines are algorithmic methods that the tool uses to ensure the system adheres to the specification provided. Moreover, formal property verification offers a strategy for simplifying the complexity of the design under verification, which can make the verification process more efficient. An advanced technique for simplification will be introduced in the following sections of the paper. Venturing further, the tool is designed to accommodate explorations into areas traditionally challenging for formal verification, such as AMS behavioral models, which simulate complex circuit behavior. Initially, this seemed impractical, yet a comprehensive review revealed that adopting an AMS model compatible with formal verification would be less cumbersome than creating a new simulation setup altogether. The ensuing sections of the paper will delve into the initial obstacles encountered when implementing this approach and the solutions devised to overcome them. A particular focus will be on maintaining the authenticity of the system's behavior throughout the verification process, ensuring that the simplifications do not alter the expected outcomes.

Unaligned:

FPV is the most commonly used app to verify the correct behavior of the design. This is a traditional method of formal verification where the features/specifications are written in the form of properties. An overview of the types of properties written is given in III. Although all of these properties are autogenerated the output file still had many lines of the same code, as the SPI is a serial communication, and the settings (polarity and phase) made the data transmission and reception in every second clock cycle. Therefore to avoid the same lines of code written repetitively for each register (read, write, and its effect on the digital/analog interface), system verilog directive sequence was used. The example sequence is given in 1.

Building on the mechanism of data transmission and reception, the data integrity checks required write and then read cues from the registers and then influence on the Digital-Analog interface. The operation of subsequent read after write requires an intermediate write/read operation which is a dummy. However, the register address can't be invalid as error handling is not a part of the specification. Therefore, this intermediate transaction is a valid one. The bitfield for defining the transmission to be read/write was exercised by the tool, which increased the state space to manifold. Therefore, the abstraction technique of k-induction was used. Induction is a method to check if the design is in a random, good state and whether it will be in a good state at the next cycle [6]. K-induction bounded model checking consists of two steps. These are the base case and the induction step. Firstly, rather than being just one state, the property is assumed to hold for a path of n successive states. This means that a more extensive base-case must be demonstrated. Secondly, the path's states are assumed to be distinct. Finiteness implies that the second strengthening completes the method in the sense that there is always a length for which the induction-step is provable, this can be formalized as equations (1) and (2) [7]:

$$\text{Base}_n \quad := \mathbf{I}_0 \wedge ((\mathbf{P}_0 \wedge \mathbf{T}_0) \wedge \ldots \wedge (\mathbf{P}_{n-1} \wedge \mathbf{T}_{n-1})) \wedge \overline{\mathbf{P}_n} \tag{1}$$

$$\text{Step}_n \quad := ((\mathbf{P}_0 \wedge \mathbf{T}_0) \wedge \ldots \wedge (\mathbf{P}_n \wedge \mathbf{T}_n)) \wedge \overline{\mathbf{P}_{n+1}} \tag{2}$$

where I_0 is the initial state, P_0 is the property in it's initial state T_0 is time zero, and n is the time step. The interpretation of these formulas/equations is mentioned in Fig. 4. If the n-th base-case is unsatisfiable, the statement should be interpreted as "There exists no n-step path to a state, violating the property, assuming the property the first n-1 steps". If the n-th induction-step is unsatisfiable, it should be read as "Following an n-step trace where the property holds true, there exists no next state where it fails". SAT solvers are used in modern EDA tools to solve such induction equations.

```
//===============================================================
// SPI read data sequence
//===============================================================
  sequence spi_read(logic [15:0] data, int cycles);//checking actual with expected
    ($past(`miso_out,30+cycles) == data[15]//adding the subsequent number of cycles
      && $past(`miso_out,28+cycles) == data[14] //because of a dummy transaction
      && $past(`miso_out,26+cycles) == data[13]
      && $past(`miso_out,24+cycles) == data[12]
      && $past(`miso_out,22+cycles) == data[11]
      && $past(`miso_out,20+cycles) == data[10]
      && $past(`miso_out,18+cycles) == data[9]
      && $past(`miso_out,16+cycles) == data[8]
      && $past(`miso_out,14+cycles) == data[7]//Comparing the MISO output with
      && $past(`miso_out,12+cycles) == data[6]// data written in write transaction
      && $past(`miso_out,10+cycles) == data[5]
      && $past(`miso_out,8+cycles) == data[4]
      && $past(`miso_out,6+cycles) == data[3]
      && $past(`miso_out,4+cycles) == data[2]
      && $past(`miso_out,2+cycles) == data[1]
      && $past(`miso_out,cycles) == data[0]);
  endsequence
```

Listing 1: SVA sequence to check the actual with expected output

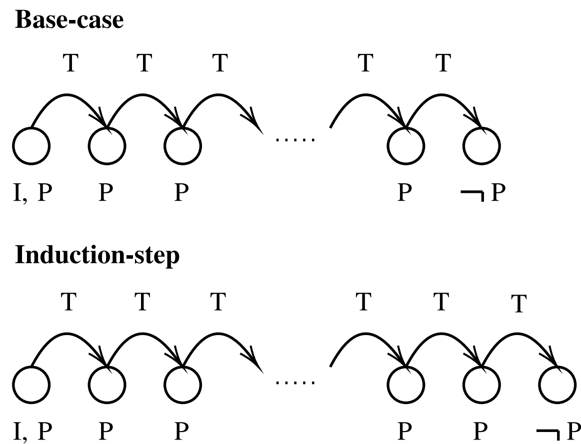

Fig. 4: Base-case and induction-step [7]

In a simple and abstract explanation to understand proof using induction, assume equations (3) and (4) about a design:

$$A_1 = 1 \tag{3}$$

$$(A_n \rightarrow A_{n+1}) = 1 \tag{4}$$

Step (3) proves whether the assertion holds true for the 1st clock cycle and step (4) checks to prove whether the

assertion A holds true for a random state A_n which would mean it will hold true for the next clock cycle (A_{n+1}) as well. If step (3) and (4) are true, it proves that A is true for all clock cycles [6].

Modern EDA tools such as Cadence JasperGold offer solutions like the State Space Tunneling (SST) based on induction-based proof, where a systematic process is used to reduce the state space of a target property. The reduction is essentially done by identifying such states and writing helper assertions called lemmas. Helper assertions are properties about internal signals (states) in the design that, if proven, will be used as "assumptions" in the proof process of the target property (or other properties), allowing these proven properties to aid in the elimination of states [6]. In our case, the tool pointed out a helper assertion based on the property that wasn't converging. The tool pointed out to keep the register data stable. Which helped the property to converge in a short time. Apart from all generated assertions for FPV, we also used Assertion Baed Verification IP (ABVIP) for SPI protocol verification. The property set also verified the features like a daisy-chain configuration.

Aligned:

While the preceding section outlines a comprehensive formal verification methodology, the totality of the verification's completeness remains uncertain. The behavioral model was implemented keeping just simulation set-up in mind. However, the prospect of creating a simulation setup solely to validate the communication between digital circuits and analog memory is deemed excessively complex for the task at hand. In terms of the AMS behavioral model, a detailed disclosure of signal names is not permissible, but a generic framework can be outlined. The model encompasses input signals including 'start', 'operation', 'time_delay_sel', and 'analog_start', with outputs such as 'ack' and 'ana_start_o'. The initiation of a transaction prompts the 'start' signal, accompanied by an operational indicator, subsequently setting a 'no_more_transaction' flag to prevent further transactions. The design incorporates a pipeline mechanism that induces a delay, dictated by 'time_delay_sel', before resetting for the next operation. Additional outputs provide external indicators like address, data, and a 'correct_data_flag'. The model also integrates a memory component for data storage during write operations and produces 'data_o' for reads. The primary issue at hand is that while most inputs and outputs are digital, the few analog inputs serve only as circuit enablers, with their functionality being beyond the scope of simulation-based verification. Thus, the primary objective was to verify that the digital circuits acknowledge the enable signal from the analog circuit, without the need for a full simulation setup. Consequently, the transformation of the non-synthesizable behavioral model into a version amenable to formal verification required minimal effort and was a logical progression. The integration of this model revealed both beneficial and adverse implications, with the latter to be described subsequently:

- **Incompatible Data Types:** It is a reminder that the Formal method does not accommodate analog data types. To resolve this limitation, a specialized package was integrated into the setup. This package effectively translates analog data types into digital logic, thereby circumventing the compilation errors initially encountered. As mentioned in a previous section, the evaluation of analog values has not been a priority within digital verification frameworks. This is primarily due to the inherent design of formal verification methods, which are optimized for discrete value checking rather than continuous analog values. Consequently, while the package enables the tool to process analog data types by converting them to a digital equivalent, it does not facilitate the verification of their correctness in the analog domain.
- **Timing:** A further challenge encountered in the system setup pertained to the treatment of delays. Delays represent non-synthesizable elements that impede achieving feature equivalence. Specifically, in our configuration, handshake signals like ack signals were subject to timing delays. These delays corresponded to an approximate number of clock cycles. To address this, single-bit handshake signals were delayed using flip-flops to match the number of cycles. However, it is critical to acknowledge that the timing of these signals' arrival did not precisely mirror real-time behavior. Although the implemented delay was quantitatively aligned with clock cycles, the lack of exactness means that the solution was an approximation rather than a precise replication of real-time signal timing.
- **Equivalence check:** The system design incorporated certain unsynthesizable constructs, notably the "initial begin" and "task" block. It is recognized that these constructs, while commonly used in simulation environments, do not contribute to the functionality of hardware modules post-synthesis and hence can be substituted with synthesizable constructs such as "always" blocks. Transitioning from unsynthesizable to synthesizable constructs could potentially result in a disparity between the original and revised versions of the modules, making it imperative to ascertain their equivalence. Establishing this equivalence is essential to ensure the integrity of the design's functionality after synthesis. This aspect of verification, while critical, is identified as an area for future investigation and is beyond the scope of the current paper. However, these changes didn't prove any

hinderance with core focus of the functionality check.

The favorable aspects of the model's integration, such as the limited gate count, facilitated a seamless conversion. As highlighted in earlier discussions, this forward-looking approach not only mitigates the risk of bugs but also paves the way for the evolution of new methodologies, thereby reinforcing the robustness of formal verification at an industrial scale.

V. Results

The development took one full working week and properties were proven in minimum time. The following bugs were reported through different applications of formal verification:

- **Width mismatch:** Width mismatch was detected with some connections and was reported. The fix was done long before the final release
- **Reset problem:** There was a register reset problem. As the register was not being reset on starting a new transaction, the expected data did not match the actual data.
- **Access policy mismatch:** Some register fields were writable even though being specified as read-only
- **Special case bugs:** There was no error response implemented for SPI. For example, if the sequential read/write was interrupted in between, it not only maligned the ongoing transaction but also the next one.

VI. Conclusion

The state-of-the-art formal verification methods were successfully deployed. The results were derived in a short amount of time. The primary reason to incorporate the analog behavioral model was to gain confidence over the behavioral model as that is also delivered to the end project owners along with RTL. The equivalence checking of the synthesizable model with non-synthesizable version is a future work. The inclusion of the analog behavioral model not only contributed to the completeness but also narrowed the possibility of a bug escape on scaling up the AMS model. Not only was the primary goal of gaining confidence in the design successfully attained, but the process of reconfiguring the entire setup in response to specification changes was also made significantly more efficient.

References

[1] H. Foster, "2022 Wilson Research Group Functional Verification Study," Mentor, A Siemens Business, Tech. Rep., Oct. 2022.

[2] K. Devarajegowda, "Model-based Generation of Assertions for Pre-silicon Verification," doctoralthesis, Technische Universität Kaiserslautern, 2021, pp. V, 167. DOI: 10.26204/KLUEDO/6640. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-66403.

[3] E. Seligman *et al.*, *Formal Verification, An Essential Toolkit for Modern VLSI Design*. Morgan Kaufmann Publishers, 2015.

[4] A. Kumar, "Pragmatic Formal Verification of Sequential Error Detection and Correction Codes (ECCs) used in Safety-Critical Design," DVCon US, 2023.

[5] A. Kumar *et al.*, "A Semi-Formal Verification Methodology for Efficient Configuration Coverage of Highly Configurable Digital Designs," DVCon US, 2021.

[6] "Jasper Gold - State Space Tunneing (SST) Rapid Adoption Kit," Cadence, Tech. Rep., Aug. 2017.

[7] N. Eén *et al.*, "Temporal Induction by Incremental SAT Solving," Mar. 2003, pp. 543–560.