Efficient Verification of a RADAR SoC Using Formal and Simulation-Based Methods

Aman Kumar, Infineon Technologies, Dresden, Germany (*aman.kumar@infineon.com*)

Mark Litterick, Verilab GmbH, Munich, Germany (*mark.litterick@verilab.com*)

Samuele Candido, Infineon Technologies, Dresden, Germany (*samuele.candido@infineon.com*)

Abstract—As the demand for Internet of Things (IoT) and Human-to-Machine Interaction (HMI) increases, modern System-on-Chips (SoCs) offering such solutions are becoming increasingly complex. This intricate design poses significant challenges for verification, particularly when time-to-market is a crucial factor for consumer electronics products. This paper presents a case study based on our work to verify a complex Radio Detection And Ranging (RADAR) based SoC that performs on-chip sensing of human motion with millimetre accuracy [1]. We leverage both formal and simulation-based methods to complement each other and achieve verification sign-off with high confidence [2]. While employing a requirements-driven flow approach [3], we demonstrate the use of different verification methods to cater to multiple requirements and highlight our know-how from the project. Additionally, we used Machine Learning (ML) based methods, specifically the Xcelium ML tool from Cadence, to improve verification throughput [4].

Keywords—*RADAR; Human-to-Machine Interaction (HMI); formal verification; Universal Verification Methodology (UVM), Unified Power Format (UPF), Machine Learning (ML)*

I. Introduction

Verification has become the bottleneck in product development cycles, as it takes more than 60% of the overall project time [5]. Complex designs such as RADAR-based SoC contribute even more to the challenges in verification on top of existing ones. Our RADAR SoC is a tiny device that has analog and digital domains that collect data from different sensors to act as a human-machine interface.
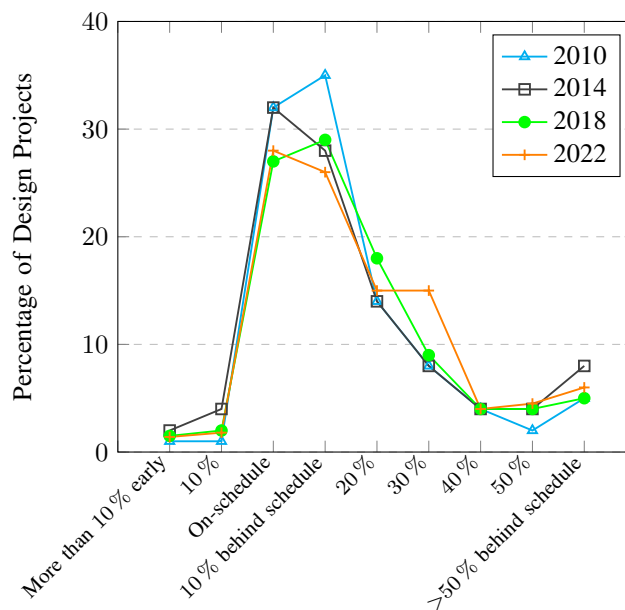

Fig. 1: Actual ASIC design completion compared to project's original schedule [5]

The study in Fig. 1 [5] shows that around 66% of Application Specific Integrated Circuit (ASIC) projects are behind schedule and 27% of projects are behind the schedule by 30% or more. One of the significant contributors to

2023
DVCON
DESIGN AND VERIFICATION
CONFERENCE AND EXHIBITION
EUROPE
10 YEAR ANNIVERSARY

the schedule not being met is functional verification, which could be expensive for a consumer electronics product. To overcome the verification challenges for a product that requires a faster Time to Market (TTM), we used different verification techniques to verify different functionalities. The project uses a requirements-based approach to collect all design requirements and create a corresponding verification plan (vPlan) to cover the requirements. We use a SystemVerilog-UVM based testbench to verify the SoC top-level with individual UVM Verification Components (UVCs) for different subsystems. Since the usage of formal property verification and automated formal checks is increasing which brings real benefit to the project [5], we use formal verification to verify complex features and blocks that are formal-friendly [6] [7]. We use Formal Property Verification (FPV) to verify design behaviours, Connectivity (CONN) verification to check end-to-end conditional connections, Control/Status Register (CSR) verification to verify the register blocks, Assertion-Based Verification IPs (ABVIPs) to verify standard protocols such as AHB and Unreachability (UNR) analysis to improve faster code coverage closure. We also perform power aware simulations to verify different power domains and Gate-Level Simulations (GLS) to confirm the correct design behaviour on the netlist level. On top of all the verification methods, we use the Xcelium ML tool from Cadence to achieve optimized regression with efficient functional and code coverage.
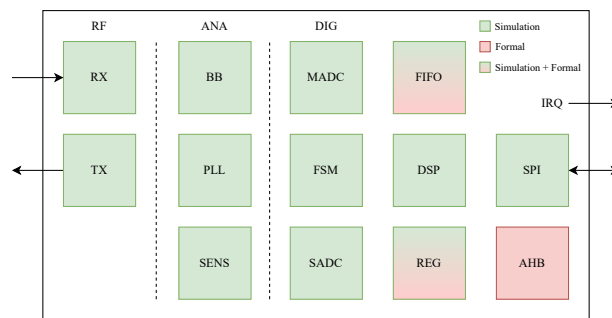
II. DESIGN OVERVIEW

Fig. 2: RADAR SoC block diagram

A simplified block diagram of the RADAR SoC is represented in Fig. 2. The SoC provides all the functionality required for a single-chip RADAR sensor device. It consists of Radio Frequency (RF) Transmit (TX) and Receive (RX) blocks which provide an air-side interface via the in-package antennas. Several key Analog (ANA) blocks provide the Phase-Locked-Loop (PLL) for the TX frequency ramps and base-band processing of the RX signals. In addition, there are embedded sensors for power and temperature measurement (among others). The Digital (DIG) part of the device includes Multichannel Analog-to-Digital Converter (MADC) and Sensor ADC (SADC) blocks, microcoded control engine (Finite-State Machine (FSM)), Digital Signal Processing (DSP) capability, memory, First In First Out (FIFO), register, and power management blocks connected via an AHB bus. The main interface to the host is via a Serial Peripheral Interface (SPI) and discrete interrupt and trigger signals.

III. DESIGN VERIFICATION

These RADAR SoC devices are primarily targeting portable consumer applications, resulting in the inevitable pressure on project timelines. From an architectural perspective, the main challenges facing the verification process include the high degree of analog content in the device, demanding low-power performance criteria, complicated and flexible control operation, and complex mathematical signal processing. To meet our design verification goals, a variety of verification methods were used for different blocks (simulation, formal or a mixture), as shown in Fig. 2. The following sub-sections provide an overview of the techniques used.

A. Simulation Based Verification

The simulation testbench is written in SystemVerilog using UVM. The same testbench and test suite are used to support digital, power-aware, gate-level and (indirectly) analog/mixed-signal simulations. A simplified block diagram of the UVM testbench is shown in Fig. 3. The testbench consists of multiple UVCs, each with responsibility for active stimulus on external interfaces, internal analog or digital streaming capabilities, or internal passive monitoring. The environment includes an automatically generated UVM Register (REG) model, which is integrated to the SPI UVC using Frontdoor (F), Adapter (A) and Predictor (P) components. The environment uses Callbacks (CB) to maintain the configuration status in each UVC in response to register operations on the Design Under Verification (DUV) [8].
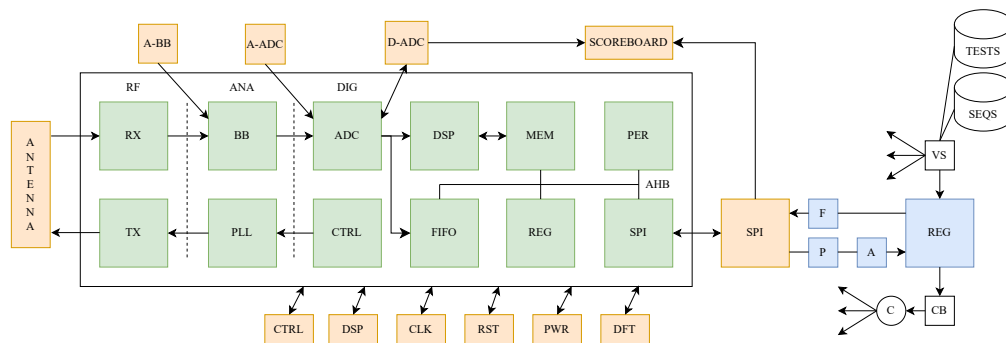
Fig. 3: UVM testbench block diagram

The DUV in the UVM environment uses real-number models for the corresponding analog blocks (TX, RX, BB, part PLL, and part ADC). The test stimulus is coded in sequences that run on the top-level Virtual Sequencer (VS) and make use of a constrained-random sequence library and low-level sequences belonging to each UVC. The main data path is validated by means of a scoreboard UVC which is connected to ADC and SPI components, but there are also signal protocol checks in each UVC interface as well as self-checking mechanisms in all tests. Functional coverage is distributed throughout the environment.

1) Digital Simulation

The digital UVC agents for external connections to Clock (CLK), Reset (RST), Power (PWR) and SPI all provide a set of sequences for stimulus generation. Analog model stimulus for RX, BB and ADC components is provided by configuration sequences and real-number streaming mechanisms (which force real number values and patterns onto internal analog nets) using the streaming techniques described in [9]. Most of the digital tests use the corresponding ADC inputs as the main path stimulus (MADC or SADC), but a few device features require BB or antenna stimulus as the source.

The DSP block requires some advanced verification techniques in order to validate the mathematical algorithms embedded in the hardware. These techniques include combining design-for-verification modes with a known-answer-test strategy and using pre-defined data scenarios that are generated externally by MatLab to predict results for specific configurations. These tests are supplemented by constrained random exploration of the configuration space at a later stage in the project once additional mathematical models are available. In these tests, the raw data source values from the MatLab scenarios are streamed directly onto the ADC digital outputs using a technique that is also described in [9].

In order to manage the linear development of the design while the verification environment evolved and make debugging more effective, the sequence hierarchy is carefully architected to enable isolation and combination of features using the techniques described in [10]. Specifically, we isolate individual features into independent exhaustive tests, and separately combine relevant features in additional compound tests, ultimately providing a few tests that combine all features with limited random exploration of the configurations space. With this goal in mind, the main UVM testcases are divided into the following groups:

- Basic tests: verify basic features and paths with limited randomization of the stimulus
- Feature tests: verify each feature with comprehensive constrained-random stimulus
- Stress tests: verify stress conditions that could reasonably be expected in the application
- Usecases tests: verify real application scenarios defined by concept, application or customer

Note that stress tests are not intended to address all Garbage-In/Garbage-Out (GIGO) scenarios such as incorrect configuration, but rather illegal conditions that could reasonably be expected to occur in real life (e.g. length errors on SPI, data path overflow and underrun, trigger mismanagement and interrupt handling errors, etc.). In addition, we also have dedicated tests for Design-for-Test (DFT) modes and provide support for re-simulation of test equipment scenarios to assist software development, post-silicon validation, and test-engineering teams.

2) Power Aware Simulation

Standard RTL simulations are used for digital verification. These simulations are not power-aware; however, power control signal protocol checks are embedded within relevant UVC agents (e.g. clock, power, control FSM) and are active during all simulations. Full power-aware simulations using the Unified Power Format (UPF) standard are supported using a run-time command (and dedicated regression). These simulations run the same tests on the same environment but enable UPF operation and therefore power-awareness during simulation.

Low power consumption is a key element that allows the RADAR to be adopted in consumer and IoT applications [1]. Power gating is one of the techniques adopted in this architecture, in order to reduce power consumption. Partitioning the design in multiple power domains allows to switch off parts of the chip that are not needed. The power management task is taken over by a power domain controller, which drives the power switches and enables and disables isolation for a certain power domain.

In a traditional RTL simulation approach, important aspects of the power strategy cannot be verified. Power ground nets, power switches, and isolation cells are among the power aspects not considered in traditional RTL simulations. In order to identify power management issues and potential bugs in the early stage of the project, power-aware verification becomes highly relevant. The most common approach is based on the IEEE Std 1801 UPF, which allows one to describe, model, and specify the power intent of the design. Multiple UPF files are used in the RADAR SoC. For each IP in the design, a UPF file is provided that describes its power intention. In addition to that, a main UPF defines the power domains, as well as the power switches; creates and connects the power supplies and loads the UPF files of IP modules.

First, a testbench environment was created for the UPF simulation of RTL with Cadence Xcelium. After that, a simple simulation was run to verify the basic RADAR functionality. Between this phase and the passing of the first test, we faced a few challenges. First of all: if the main UPF file was written for synthesis tools, there might be differences in the paths to design instances or signals. This is particularly true for module instances created with generate statements. This kind of issue usually causes errors during compilation. A more serious issue is the isolation of SystemVerilog interfaces, which are intensively used in RADAR design as ports on the power domain boundaries. The support for SystemVerilog interface isolation is limited to the Xcelium version adopted in the project. The isolation of interface elements of type `enum` is, for example, not supported. In order to overcome this issue, which led to failing simulations, workarounds had to be used. In this case, the UVM routine `uvm_hdl_deposit` was used to mimic the isolation behaviour. With such workarounds great care is required since they might hide functional issues. After solving the mentioned issues and other secondary problems, all related to the setup and the tools, the RTL UPF simulation showed its benefits. Missing isolation and, therefore, propagation of X's and wrong switch-off sequence are two prominent examples. It is worth mentioning that this kind of bug could have probably been found in GLS. However, GLS is normally performed in a phase close to tape-out and the time and effort for debugging would have been much higher.

3) Gate-Level Simulation

We have a requirement to perform GLS not just to validate the post-synthesis netlist and cross-check the static-timing-analysis constraints, but also to enable us to do more accurate power analysis on the cell-based netlist after clock-tree synthesis. This power analysis is required for low-power and high-power evaluation and must be repeated with various timing corners (e.g. nominal, slow, and fast). It is not the goal of GLS to prove all low-level protocol and functional operations are still intact (this is supported by equivalence checking), but we do need to know if tests are working correctly in order to be sure that the test ran correctly.

In order to minimize the burden of GLS on the tight timescales, we applied a strategy to minimize testbench effort and reduce maintenance between release candidates. Specifically, since all our tests are self-checking (they all call result and status sequences to validate important aspects of the results), we implement a mechanism to disconnect all internal passive UVC interfaces completely during GLS. All external UVCs (such as SPI, CLK, RST, and PWR) remain connected, as too does the ADC interface (since it is the primary input for most tests) and the primary data-path scoreboard and a few critical clock and power checks. This results in a need to maintain far fewer internal connects to the netlist, which in turn provides the back-end team with more flexibility in terms of flattening and processing the netlist (i.e., we have far fewer nets that we need to find in the gate-level netlist).

This technique allows us to run almost all tests out of the box with very little maintenance overhead. The same suite of tests is run on exactly the same UVM environment, where the only difference is that most UVCs are disconnected, and some key remaining nets need alternative paths to be defined for the netlist. Note that in general

the HDL-paths for the register model connections are no longer valid in GLS after netlist optimization, so a few tests that rely on these paths to do backdoor operations are not executed in GLS regressions.

4) Analog/Mixed-Signal Simulation

As mentioned above, the digital simulations are run with real-number models for the analog macros instantiated within the DUV. These models are provided by the analog design team. In order to run high-level analog and Analog/Mixed-Signal (AMS) simulations on the real analog netlist, we simulate dedicated AMS usecase tests on the digital UVM simulation environment and export the corresponding Value Change Dump (VCD) file to our analog team. They use these VCD files as direct digital stimuli for the full DUV with analog netlist in an appropriate analog simulator setup. In this respect, the UVM environment supports AMS indirectly.

B. Formal Verification

Formal verification uses technologies that mathematically analyze the space of possible behaviours of a design, rather than computing results for particular values [7]. It is an exhaustive verification technique that uses mathematical proof methods to verify whether the design implementation matches design specifications [11].
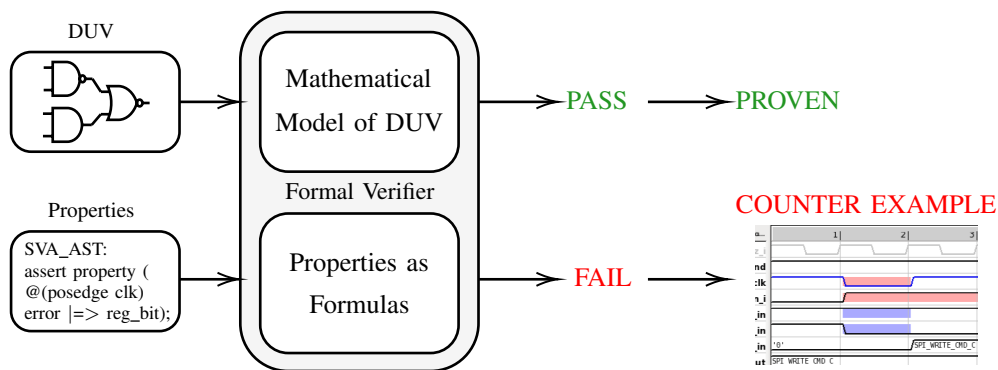

Fig. 4: Formal verifier [11]

Fig. 4 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, it is said to have failed, 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 [12]. To meet different verification requirements, we used different formal verification techniques and apps from Cadence Jasper. 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 behaviour of the registers. 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. This generation is based on a metamodel-based automation framework explained in [13]. Since this approach is fully automated and a push-button solution, it also helps design engineers verify their RTL for every register change before releasing the next version of the RTL.

A major benefit of using 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. On the other hand, a failure in simulation based setup may happen after several clock cycles, making the debugging of the root cause difficult. However, there are also limitations associated with this approach. Usually, the properties generated by the tool are encrypted and cannot be accessed. On the other hand, 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 app from Cadence Jasper offers formal based connectivity verification for straight and conditional connections. Traditional simulation based connectivity checking does not exercise a brute force approach; however, formal verification gives complete coverage due to its exhaustive nature. In our design, we have a lot of digital multiplexer (DMUX) connections that need to be verified. We have several analog blocks that connect to the digital top. We verify end-to-end connectivity for such connections. We also have several overwrite bits in the test registers that are used during debug mode. These overwrite bits are also a good candidate for such a connectivity check that we performed for the project.

Similar to the DMUX, the DUV also has many Analog Multiplexer (AMUX) which we wanted to verify with the formal approach. However, since most modern formal tools only support synthesizable designs and not analog/AMS models, it was not possible to verify the AMUX using this approach. We had to use simulation based approach to verify the design in this case. Although there are several techniques to also verify connectivity in analog/AMS designs, we are currently evaluating them and will present the results in another paper.

3) Formal Property Verification

FPV is the most commonly used app to verify the correct behaviour of the design. We have used FPV to verify several design features, such as FIFO and arbiters. Using a formal based approach helped expose several design bugs that would have been hard to find using the simulation based approach. The design also has several scan isolation based requirements where several signals need to be isolated to either logic 0 or logic 1 in the scan mode. This requirement was also verified with the FPV approach which gave us confidence in the design implementation.

The DUV also has an AHB bus that connects SPI to the registers. Due to time limitations, we did not prepare a dedicated UVC for the AHB since the correctness of it is implicitly verified with several UVM testcases. However, since AHB is an important aspect of the design and must comply with standards, we used Cadence ABVIP to verify its correctness. The AHB DUV acts as a subordinate and the ABVIP acts as a manager in the verification setup. Using this approach, we were able to verify the AHB bus.

4) Clock Domain Crossing Verification

The Clock Domain Crossing (CDC) app from Cadence Jasper offers formal based CDC and Reset Domain Crossing (RDC) verification. The tool is used to verify both structural as well as functional checks that are automatically generated based on several synchronizer schemes used in the design. Later, these checks/properties are also verified in the effect of metastability by using a Metastability Injection (MSI) flow. The real benefit of using the CDC app is that the user can also write custom checks/properties that can be proved in the metastability effects. Later, these properties and the MSI model can be exported to the simulation setup and reused while doing the verification. A detailed explanation of a pragmatic formal CDC verification is described in [14].

5) Unreachability Analysis

The UNR app from Cadence Jasper is used to find out parts of the RTL code that are unreachable. This analysis helps to identify code coverage holes in an early cycle of RTL development. The analysis is shared with the design engineers to make sure either to fix the RTL code or waive them off if not needed. In the RADAR project, we were able to find a significant amount of unreachable code that was waived off before the tape-out.

C. Verification Management

The project uses a single point for the traceability of all requirements. All requirements and verification items (such as tests, checks and coverage) are added to the requirements management tool. Later, these verification items are mapped to the corresponding requirements. A script is used to export the vPlan in Cadence vManager readable format that is later used to map all the tests, checks, and coverage items from the regression runs. The requirements management tool acts as the one-stop shop for all traceability of requirements and their verification items. The regression setup in the project is divided on the basis of different strategies. We use a general regression setup for UVM simulations and others for CDC jitter, UPF, GLS and formal verification. All these regressions are combined for overall metrics analysis. A script was set to trigger automatic weekly regressions on the weekend and even nightly regressions basis at a point when we were very close to the tape-out.

To increase the verification throughput, we used an ML based regression approach that is offered by Cadence Xcelium ML tool. The tool generates an optimized Verification Session Input File (VSIF) based on the learning

models derived out of the regression runs which contains a list of testcases that contribute the most to the coverage metrics. The results of the analysis to improve functional and code coverage with a similar or fewer number of regression runs are discussed in [4]. We also observed that the style of tests resulted in certain choices of tests in the optimized VSIF. The testcases are implemented to test either a particular functionality in isolation or a complex testcase that covers the isolation ones as well. In a usual regression, if the complex testcase fails, the isolation one would also fail. This helps to identify the actual root cause rather than debugging a more complex testcase. However, since the ML models are trained to pick up the best (and more complex) testcase that contributes more to the coverage, it naturally picks up the more complex testcases rather than the isolation ones. This does not limit the capabilities of ML, but rather something that is affected by the styling of tests. For future possibilities, we are also exploring the capabilities of Xcelium ML to expose bugs or unique failure signatures in less turnaround time.

IV. Results

With the verification approach mentioned in this paper, we were successfully able to verify the chip within a short time frame. We achieved acceptable requirements coverage for the first stage, and the rest were waiveivable. Note that the extensive protocol checks were not part of the verification and are something planned for the next phase of the project. We also do not have exhaustive functional coverage for the first stage, which is planned for the next phase. Simulation based verification exhibited a high degree of coverage across various functional, performance, and stress tests. Connectivity verification confirmed the integrity of data paths and signals, while CDC verification mitigated potential metastability issues. The integration of ML based regression optimization contributed to achieving higher coverage in a shorter time frame. Using the approach we discussed, we achieved acceptable code coverage as well. Although the first stage of our verification environment had less comprehensive protocol checks and functional coverage, this level of code coverage compensates for those gaps. Overall, this approach strikes a good compromise between time and exhaustiveness.

V. Conclusion

The paper presents a real case study of one of our recent projects based on RADAR sensors, and the goal was to efficiently verify the design in an aggressive timeline with high confidence. Our work highlights the importance of a multi-faceted verification strategy to tackle the challenges posed by complex SoC designs and presents the lessons learnt during the course of verification. A requirement-driven flow was also used to track coverage and backup verification holes with real coverage holes. The synergy between formal and simulation based methods, coupled with ML-driven optimization, provides a powerful framework for achieving comprehensive and efficient verification, ensuring the robustness and reliability of modern SoCs in the fast-paced consumer electronics landscape.

References

[1] Saverio Trotta et al. "2.3 SOLI: A Tiny Device for a New Human Machine Interface". In: *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. 2021, pp. 42–44. DOI: 10.1109/ISSCC42613.2021.9365835.

[2] Keerthikumara Devarajegowda et al. "A Mutually-Exclusive Deployment of Formal and Simulation Techniques Using Proof-Core Analysis". In: DVCon Europe, 2017.

[3] Serrie-Justine Chapman, Darren Galpin, and Mike Bart. "Requirements driven Verification methodology (for standards compliance)". In: DVCon Europe, 2014.

[4] Narayan Gadde Deepak et al. "Improving Simulation Regression Efficiency using a Machine Learning-based Method in Design Verification". In: DVCon Europe, Dec. 2022.

[5] Harry Foster. *2022 Wilson Research Group Functional Verification Study*. Tech. rep. Mentor, A Siemens Business, Oct. 2022.

[6] Harry Foster. "Guidelines for creating a formal verification testplan". In: 2006.

[7] Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar. *Formal Verification, An Essential Toolkit for Modern VLSI Design*. Morgan Kaufmann Publishers, 2015.

[8] Mark Litterick and Marcus Harnisch. "Advanced UVM Register Modeling". In: DVCon US, 2014.

[9] Mark Litterick, Jeff Vance, and Jeff Montesano. "To Infinity And Beyond - Streaming Data Sequences In UVM". In: DVCon US, 2021.

[10] Mark Litterick, Jeff Vance, and Jeff Montesano. "Be a Sequence Pro to Avoid Bad Con Sequences". In: DVCon US, 2019.

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

[12] Aman Kumar and Sebastian Simon. "A Semi-Formal Verification Methodology for Efficient Configuration Coverage of Highly Configurable Digital Designs". In: DVCon US, 2021.

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

[14] Aman Kumar, Muhammad Ul Haque Khan, and Bijitendra Mittra. "Pragmatic Formal Verification Methodology for Clock Domain Crossing (CDC)". In: DVCon Europe, 2023.