Journal homepage: https://itce.vn.ua/en

Information Technologies and Computer Engineering **Vol. 23, No. 2. 2026**

Article's History: Received: 17.02.2026 Revised: 26.05.2026
Accepted: 25.06.2026 Published: 03.08.2026

ISSN 1999-9941; e-ISSN 2078-6387
UDC 004.31:621.382:004.415.53
DOI: 10.31649/vitce/2.2026.132

Methods of functional and formal verification of digital systems: Theoretical analysis and experimental application to modules of varying complexity

Oleksandr Ostrianko*
Master of Science, Postgraduate Student
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
03056, 37 Beresteiskyi Ave., Kyiv, Ukraine
https://orcid.org/0009-0008-9107-4254

Abstract. The study aimed to provide a theoretical justification of the nature of functional and formal verification of digital systems, to compare their methods, and to determine the possibilities for their coordinated application to digital modules of varying structural complexity. Methodology was based on normative-terminological, structural-analytical, functional-verification, formal logical and comparative-analytical methods, which made it possible to distinguish between scenario-based and property-based approaches to verifying hardware logic and to conduct an experimental and demonstrative verification of three test digital modules. The study established that functional verification ensured the verification of a digital module's behavioural compliance with the specification through test inputs, expected output responses, test benches, simulation and analysis of functional mode coverage. Formal verification yielded a different type of verification conclusion, as it was aimed at confirming or refuting correctness properties, invariants, logical and temporal conditions, valid transitions and the unreachability of erroneous states. Experimental and demonstrative verification on a multiplexer, a finite-state machine and a simplified interface controller showed that, as the complexity of the digital module increased, the number of functional scenarios grew, mode coverage became more challenging, and the risk of failing to detect hidden states, deadlocks or rare signal combinations increased. For the multiplexer, both approaches yielded consistent results; for the finite-state machine, the verification of resets, states and transitions was enhanced; for the interface controller, formal verification improved the assessment of deadlocks, signal conflicts and erroneous modes. The feasibility of a combined strategy, in which functional scenarios, coverage analysis, formal properties and the interpretation of possible counterexamples complement one another, has been demonstrated. The practical significance of the results is determined by their application in digital design, hardware verification and educational and research environments for planning the verification of digital modules, refining specifications, conducting coverage analysis, interpreting counterexamples and making decisions regarding the correctness of hardware logic

Keywords: hardware logic; test scenarios; coverage analysis; correctness properties; counterexamples; control signals

Introduction

The increasing complexity of digital systems, hardware accelerators, microprocessor units, embedded controllers and programmable logic integrated circuits has led to stricter requirements for verifying their correctness at the design stage. Errors in hardware logic can manifest not only as an incorrect response to a specific combination of input signals, but also as a disruption in the sequence of states, failure to reach the correct operating mode, deadlock, a conflict between control signals, or failure to satisfy a timing condition. Under such conditions, the verification of digital systems is not limited to running a set of tests, as the increasing complexity of a module leads to a greater number of possible scenarios, states and transitions, which are difficult to cover through functional testing alone. In this regard, combination of functional and formal verification verified behavioural compliance of a digital module with the specification and the correctness properties of the hardware logic. In research, the problem of verifying digital systems is associated with the hardware design stage, during which the compliance of the digital logic with the given specification is checked. The methodology for formal verification and the design of electronic projects based

Suggested Citation:
Ostrianko, O. (2026). Methods of functional and formal verification of digital systems: Theoretical analysis and experimental application to modules of varying complexity. *Information Technologies and Computer Engineering*, 23(2), 132-144. doi: 10.31649/vitce/2.2026.132.

*Corresponding author (*oleksandrostrianko@outlook.com*)

on field-programmable gate arrays (FPGAs) was proposed by O. Odarushchenko *et al.* (2024). Within this approach, the correctness of the hardware logic is assessed not only through modelling but also through the verification of specified properties.

The development of teaching and modelling tools for digital and microprocessor systems constitutes a distinct area of research. The areas of application of machine learning for software-supported verification of hardware design were systematised by N. Wu *et al.* (2024). The study demonstrated the potential for automated verification support, but also highlighted the need to distinguish between functional, formal and property-oriented verification logic.

The application of statement-based verification for an I2C inter-integral bus module using the SystemVerilog hardware description and verification language was described by D.-W. Moon *et al.* (2025). In this approach, assertions and properties serve as a means of verifying protocol and control logic, as they can detect not only of incorrect signal responses but also deviations from the module›s expected operational sequence. The use of large language models in the design and verification of hardware systems was analysed by M. Abdollahi *et al.* (2025). The results showed that such tools support the generation and analysis of digital modules, but at the same time increase the risks of incomplete, incorrect or insufficiently verified hardware logic. U. Farooq & H. Mehrez (2021) investigated pre-physical verification of hardware design by generalising the use of multi-platform FPGA-based solutions for verifying digital systems before the physical implementation stage. The authors argued that early verification of a hardware design reduces the risk of critical errors being detected at later stages of development, but requires a combination of modelling, test scenarios and hardware-oriented validation methods. Algorithms and techniques for verifying hardware logic models as a branch of formal verification were systematised by G. Cabodi *et al.* (2024). The study established that the verification of hardware logic models makes it possible to analyse the properties of states, transitions and temporal conditions, which are difficult to fully cover using functional tests alone.

Despite substantial research on FPGA design, formal checking, statement-based verification, hardware logic model checking and software-supported verification, the issue of comparing functional and formal verification of digital modules of varying structural complexity remains insufficiently addressed. Existing research mainly focuses on specific hardware implementations, individual verification methods, or tools to support the verification process. At the same time, the logic behind the change in the effectiveness of functional and formal verification during the transition from a simple combinational module to sequential logic and a more complex control module remains less well understood. This necessitates a justification for the combined application of digital system verification methods, considering the structural complexity of hardware logic.

The study aimed to highlight the methodological differences between functional and formal verification of digital systems, to conduct an experimental and demonstrative verification of their application to digital modules of varying structural complexity, and to determine the possibilities for the combined use of these approaches. To achieve this aim, the following objectives were set: to theoretically distinguish between functional and formal verification of digital systems as different approaches to verifying hardware logic; to experimentally demonstrate the application of these approaches to a multiplexer, a finite-state machine and a simplified interface controller through test scenarios and formal properties; to compare the results obtained using a unified set of criteria and to justify a combined verification strategy for digital modules of varying structural complexity.

Materials and Methods

The study was conducted using normative-terminological, structural-analytical, functional-verification, formal-logical and comparative-analytical methods. The normative-terminological method was applied to establish the conceptual framework of the research following ISO/IEC/IEEE standards. ISO/IEC/IEEE 24765:2017 (2017) was used to standardise the concepts of verification, validation, requirements and specifications. ISO/IEC/IEEE 29119-1:2022 (2022) was applied to define the logic of the test object, test data, the expected result and the deviation of actual behaviour from the specified behaviour. IEEE 1012-2024 (2025) has been used to interpret verification as the process of confirming that a digital module complies with the specified requirements. Within this framework, functional verification is regarded as the examination of a digital module's response to specified input stimuli, whilst formal verification is regarded as the proof or refutation of logical and temporal properties presented in the form of statements, constraints or invariants. The methodological framework for hardware description and verification of digital modules is based on standards for hardware description languages and property specification. IEEE 1800-2023 (2024) was used to justify the construction of testbenches, the application of assertions and coverage analysis in the SystemVerilog environment. IEEE 1800.2-2020 (2020) was addressed to characterise the Universal Verification Methodology (UVM) as an approach to organising a reusable verification environment. IEEE 1076-2019 (2019) was used as the basis for describing digital modules using the Very High Speed Integrated Circuit Hardware Description Language (VHDL), whilst IEEE 1850-2010 (2010) was used to specify properties using the Property Specification Language.

A structural-analytical method was used to select test digital modules and classify them according to the type of hardware logic and level of structural complexity. Three digital modules were selected for experimental and demonstrative verification: a multiplexer, a finite state machine and a simplified interface controller. The multiplexer

represented combinational logic, for which correctness was determined by the output corresponding to the selected input depending on the control signal. The finite state machine was considered as a sequential module, where the initial state, reset signal, permissible transitions between states and output responses were of significance. A simplified interface controller was used as an example of control logic with start conditions, acknowledgement waiting, transaction completion, possible deadlocks and control signal conflicts. The selection of these modules made it possible to trace the evolution of the verification task during the transition from simple combinational logic to sequential and control logic. The functional verification method was applied to construct test scenarios, define input stimuli and expected output responses, and verify that the behaviour of each digital module conformed to the specification. For the multiplexer, combinations of input and control signals were verified, as well as the correspondence of the output to the active channel. For the finite state machine, the test scenarios covered reset, transition from the initial state, valid transitions between states, and output responses in specific modes. For the simplified interface controller, the following were tested: initiating communication, waiting for acknowledgement, cycle completion, response to a false condition, and the consistency of control signals. The result of the functional verification was determined as "pass" or "fail" for each scenario, following a comparison of the module's actual behaviour with the expected behaviour.

The formal-logical method was used to specify the properties of correct behaviour, invariants, and logical and temporal conditions that had to be satisfied regardless of any individual test scenario. For the multiplexer, the property of logical selection was verified, i.e. the correspondence between the output and the input determined by the control signal. For the finite state machine, the properties of correct recovery after a reset, the admissibility of transitions and the unreachability of prohibited states were formalised. For a simplified interface controller, the properties of non-blocking behaviour, consistency of control signals, reachability of the final state and unreachability of the fault mode were specified. The result of formal verification was interpreted as confirmation of a property, its violation or the generation of a counterexample. The study was conducted in a sequence ranging from the normative and conceptual distinction between functional and formal verification to the selection of test digital modules, the formulation of functional scenarios, the specification of formal properties, and the comparative generalisation of the results. A comparative-analytical method was applied to compare the results of functional and formal verification according to a single set of criteria: the number of test scenarios, the completeness of mode coverage, the ability to detect typical, boundary and hidden faults, the complexity of property formalisation, the presence of counterexamples, the interpretability of the results, and the scalability of the method as the complexity of the digital module increases. The comparison criteria were defined based on studies by C. Kern & M.R. Greenstreet (1999), A.B. Mehta (2018), and H. Witharana *et al.* (2022). This comparison differentiated between scenario-based verification of the module's observable behaviours and property-based verification of logical conditions, which may lie outside the scope of individual test scenarios.

Results

Theoretical distinction between functional and formal verification and the design of the experimental test procedure. A regulatory and methodological distinction between functional and formal verification has shown that these approaches differ not only in the verification tools used, but also in the type of verification conclusion reached. Functional verification aims to establish that the behaviour of a digital module conforms to a given specification within the scope of defined test scenarios. Formal verification is geared towards confirming or refuting the correctness properties of hardware logic, in particular the validity of states and transitions, the fulfilment of logical or temporal conditions, and the impossibility of reaching erroneous states. This distinction is fundamental to the verification of digital modules, as functional verification confirms correctness only within the scope of specified input conditions and expected responses, whilst formal verification can be used for the analysis of properties that must hold regardless of any test scenario. Thus, the functional approach provides behavioural verification, whilst the formal approach provides a property-based assessment of logical correctness. The main differences between these approaches are summarised in Table 1.

Table 1. Results of the normative and conceptual distinction between functional and formal verification of digital systems

Delimiter	Functional verification	Formal verification	Values for digital module verification
Audit subject	Behaviour of digital module in specified operating modes	Properties of correct behaviour of a digital module	Distinction between behavioural and property-based verification logic
Verification basis	Test scenarios, input stimuli, expected output responses, test benches	Formal statements, invariants, logical and temporal conditions	Established varied nature of the audit result
Input data type	Test stimulus sets, signal sequences, expected responses	Property specifications, environmental constraints, permissible states and transitions	Correlation between functional testing and the completeness of test suites was identified
Expected result	Pass or fail of the test scenario	Confirmation of a property, its violation, or the construction of a counterexample	Distinction between scenario-based and evidence-based types of results was made

Table 1. Continued

Delimiter	Functional verification	Formal verification	Values for digital module verification
Method for error detection	Discrepancy between the actual outcome and the expected result	Violation of a specified property, an invalid transition or an incorrect state	Various grounds for identifying hardware logic errors have been clarified
Role of specification	Defines expected behaviour for test scenarios	Is a source of formal properties, statements and constraints	Central role of the specification in both approaches was confirmed
Nature of evidence	Limited to tried-and-tested scenarios and the coverage already achieved	Relates to proving or disproving properties	Different levels of substantiation of the audit opinion were identified
Key restrictions	Incompleteness of possible scenarios as the complexity of the digital module increases	Complexity of formalising properties, invariants and constraints of the environment	Limits of the independent application of each approach were defined

Source: compiled by the author based on ISO/IEC/IEEE 24765:2017 (2017), ISO/IEC/IEEE 29119-1:2022 (2022), IEEE 1012-2024 (2025)

An analysis of the data presented in Table 1 showed that functional verification provides a scenario-based conclusion regarding the behaviour of a digital module, whilst formal verification provides a property-based conclusion regarding the logical correctness of the hardware description. The limitation of the functional approach relates to the completeness of test scenarios, whilst that of the formal approach relates to the complexity of precisely formulating properties, invariants and environmental constraints. This confirmed the value of further comparing the two approaches on digital modules of varying structural complexity.

Comparison of three test digital modules showed that the structural complexity of the hardware logic directly influenced the nature of the verification task. The multiplexer represented combinational logic, for which the main criterion of correctness was that the output corresponded to the selected input depending on the control signal. At this level, functional testing had a limited and controlled scope of test cases, whilst formal verification served to confirm the basic property of logical selection. A finite state machine characterised sequential logic, in which the result depended on the initial state, the reset signal, permissible transitions and output responses. For this module, the verification task shifted from checking individual signal combinations to analysing states, transitions and the correctness of recovery after a reset. Simplified interface controller modelled control logic with branching behaviour, start conditions, acknowledgement waiting, exchange completion, possible deadlocks and conflicts between control signals. At this level, running through functional scenarios did not eliminate the risk of hidden states, deadlocks or conflicts between control signals; consequently, the importance of formal properties increased. The resulting comparison demonstrated a gradual increase in the complexity of the verification process: from checking the logical selection in the multiplexer to evaluating valid transitions in a finite state machine and analysing deadlocks, reachability of the final state and signal consistency in the interface controller. A summary of the digital modules used in the experimental demonstration verification is presented in Table 2.

Table 2. Specifications of the digital modules used in the experimental demonstration test

Complexity level	Digital module	Logic type	Key verification elements	Potential errors	Expected verification difficulty
Simple	Multiplexer	Combinational logic	Input combinations, control signal, correspondence between the output and the selected input	Incorrect channel selection, an error in the control logic, or a mismatch between the output signal and the expected combination	Low
Average	Finite automaton	Sequential logic	Initial state, clock signal, reset signal, permitted transitions, output responses in individual states	Incorrect transition, prohibited state, incorrect response to reset, violation of state sequence	Average
Complex	Simplified interface controller	Branched control logic	Control signals, transition conditions, exchange scenarios, waiting states, blocking, cycle completion	Unreachable state, hidden error, deadlock, signal conflict, communication protocol violation	High

Source: compiled by the author based on IEEE 1076-2019 (2019), IEEE 1800.2-2020 (2020), IEEE 1800-2023 (2024)

The analysis presented in Table 2 showed that the increasing complexity of the digital module altered not only the number of test scenarios but also the very nature of the verification task. For the multiplexer, the key requirement remained the confirmation of a one-to-one correspondence between the control signal and the selected input; therefore, both approaches had a relatively simple application structure. For the finite-state machine, the focus of verification shifted to states, transitions and responses to resets, which increased the importance of formalised properties. For the interface controller, the determining factors were control sequences, signal blocking and conflicts – that is, those aspects which are difficult to cover fully using functional scenarios alone. This confirmed the value of further comparing functional and formal verification on the same test objects.

Functional and formal verification of test digital modules: Scenarios, properties and results. The results of the functional verification showed that the scenario-based

approach provided sufficient verification of the specified operating modes for all three digital modules; however, its completeness decreased as the structural complexity of the hardware logic increased. For the multiplexer, the verification confirmed that the output corresponded to the active input for all specified control combinations. For the finite state machine, the correctness of the reset condition, the transition from the initial state, valid transitions and output responses in the specified modes was confirmed. For the simplified interface controller, "start", "wait", "ack", "done" and "error" scenarios confirmed the correctness of the main exchange sequence, in particular the transition from start to wait, acknowledgement reception, cycle completion and error handling. However, scenario-based testing did not eliminate the risk of hidden states, deadlocks or control signal conflicts, which might not have manifested themselves in the specified test suite. The summarised results of the functional verification of digital modules of varying structural complexity are presented in Table 3.

Table 3. Test scenarios and results of functional verification of digital modules of varying complexity

Digital module	Number of scenarios	Test scenarios	Verified modes	Result of functional verification	Limitations of scenario-based approach
Multiplexer	4	Selection of each input by a control signal; checking that the output corresponds to the active channel	Basic logic selection modes; change in the control signal; output corresponding to the selected input	Scenarios were completed; actual output matched expected active channel	Constraints are minimal, as the space of input combinations is small and manageable
Finite automaton	5	Reset; transition from initial state; permissible transitions between states; checking output reactions; returning to a specified state	Initial state; reset; permitted transitions; output reactions in individual states	Test cases were run for the specified sequences; the correctness of typical transitions and responses to a reset has been verified	Completeness depends on the coverage of significant sequences; rare or unexpected transitions may remain outside the test set
Simplified interface controller	6	Start; wait; ack; done; error; verification of consistency of control signals	Initiating the exchange; awaiting confirmation; receiving an ACK; completing the cycle (done); responding to an error; control signal interaction	Test cases were run for the specified modes; the correctness of the main exchange sequence and the response to errors has been confirmed	Risk of unaccounted-for signal combinations, hidden states, deadlocks or conflicts that are not apparent in the specified scenarios

Source: compiled by the author

The data in Table 3 showed that the effectiveness of functional verification depended directly on the structural complexity of the digital module. For the multiplexer, the scenario-based approach ensured complete coverage of the basic modes, as the verification boiled down to checking the logical selection. In the finite-state machine, the scope of verification increased due to the need to account for states, reset conditions and valid transitions; consequently, the completeness of the result depended on the quality of the sequence selection. The greatest limitation of the scenario-based approach was found to be for the interface controller: even after running the main "start", "wait", "ack", "done" and "error" scenarios, there remained a risk of hidden states, deadlocks or control signal conflicts. This confirmed that functional verification is sufficient for checking the intended operating modes, but as the complexity of the module increases, it needs to be supplemented by formal property verification.

Results of formal verification showed that property-based verification made it possible to extend the assessment of the correctness of digital modules beyond functional scenarios. Whilst functional verification confirmed the module's behaviour only in specified modes, formal verification made it possible to assess the logical conditions that had to be satisfied for valid states, transitions and control sequences. For the multiplexer, the property of logical selection of the active input was confirmed; for the finite state machine, the correctness of the reset and the validity of transitions were confirmed; and for the simplified interface controller, the significance of verifying the completion of the exchange cycle, the absence of deadlock and the consistency of control signals was confirmed. The summarised results of the formal verification are presented in Table 4.

Table 4. Properties, invariants and results of formal verification of digital modules

Digital module	Formalised properties	Verified invariants or conditions	Result of formal verification	Counterexample or a limitation on interpretation	Values for module testing
Multiplexer	Output must correspond to the input specified by the control signal	For each value of the control signal, only the corresponding input is active; the output does not change outside the selection logic	Property of logical selection was confirmed	No counterexample is expected provided the selection scheme is described correctly	Basic logical compliance of the combinational module with the specification has been confirmed

Table 4. Continued

Digital module	Formalised properties	Verified invariants or conditions	Result of formal verification	Counterexample or a limitation on interpretation	Values for module testing
Finite automaton	After a reset, the initial state is reached; transitions occur only between permissible states; a forbidden state is unreachable	Initial state is defined; following a reset signal, the automaton returns to the specified state; invalid transitions are not executed	Correctness of the reset and the standard permitted transitions has been verified; the unreachability of the forbidden state requires a complete description of the set of states	Possible counterexample in the case of an incomplete transition table or an incorrectly defined transition condition	Overall logic of states and transitions outside the scope of individual functional scenarios has been assessed
Simplified interface controller	Exchange cycle must be completed under valid conditions; control signals must not conflict; no deadlock must occur; an erroneous state must not be attainable without the relevant condition	Waiting state does not turn into an infinite deadlock; the start, confirmation and completion signals do not conflict with one another; the final state is reachable	Some of the properties were verified for the given conditions; the properties of non-blocking and non-failure-inducing behaviour require the constraints of the environment to be precisely specified	Possible counterexample arising from an incomplete description of the external conditions, in particular the absence of confirmation or an incorrect sequence of control signals	Detection of hidden logical inconsistencies, which cannot be guaranteed by scenario-based functional testing alone, has been enhanced

Source: compiled by the author based on C. Kern & M.R. Greenstreet (1999), IEEE 1850-2010 (2010), A.B. Mehta (2018), H. Witharana *et al.* (2022)

An analysis of Table 4 showed that the diagnostic value of formal verification increased in line with the structural complexity of the digital module. For the multiplexer, formal verification served as a rigorous confirmation of basic logical correctness, whilst for the finite state machine, verification assessed the integrity of the system of states and transitions. Formal verification proved to be most significant for the simplified interface controller, where the result depended not only on the execution of the main scenarios, but also on the correctness of environmental constraints, the absence of deadlocks and the consistency of control signals. Thus, formal verification complemented functional verification in cases where the scenario-based approach failed to fully detect hidden states, invalid transitions or potential counterexamples.

Comparative analysis and combined strategy for verification of digital modules of varying complexity. Comparison of the results of functional and formal verification showed that both approaches ensured the correctness of digital modules but produced different types of verification conclusions. Functional verification was best suited to confirming the behaviour of the module in specified modes, whilst formal verification made it possible to assess correctness properties, invariants, unreachable states and the conditions under which counterexamples arise. Using the examples of a multiplexer, a finite state machine and a simplified interface controller, the difference between the approaches was found to increase with the growing structural complexity of the hardware logic. The generalised results of the comparison are presented in Table 5.

Table 5. Comparative results of functional and formal verification of digital modules

Comparison criteria	Functional verification	Formal verification	Summary of audit findings
Number of test scenarios	Increased from 4 scenarios for the multiplexer to 6 scenarios for the simplified interface controller	Not determined by the number of scenarios, as the verification was performed through properties, invariants and constraints	As the complexity of the module increased, the scenario-based approach became more labour-intensive and needed to be supplemented with property-based testing
Comprehensiveness of coverage of operating modes	Highest for the multiplexer; for the finite-state machine and the controller, it depended on the choice of sequences and the verification conditions	Depended on the completeness of the properties defined and the correctness of the environment constraints	Thoroughness of the verification was determined by the quality of the specification but was implemented using various methods
Identification of common mistakes	Ensured the identification of discrepancies between actual behaviour and the expected response in a given scenario	Determined whether a given property or invariant was violated	Functional test was effective for the specified modes, whilst the formal test was effective for verifying the correctness conditions
Detection of hidden errors	Limited to the scenarios included in the test suite	Covered impermissible transitions, unreachable states, deadlocks and conflicts between control signals	For sequential and control modules, formal verification had greater diagnostic value
Verification of logical properties	Conducted indirectly via expected output reactions	Conducted directly by means of the properties of logical selection, transition validity and signal consistency	Formal verification provided direct confirmation or refutation of the properties of hardware logic

Table 5. Continued

Comparison criteria	Functional verification	Formal verification	Summary of audit findings
Checking temporal and sequential conditions	Conducted via predefined sequences of reset, transition, start, wait, acknowledgement and cycle completion	Conducted based on the conditions of transition admissibility, reachability of the final state and the absence of blocking	Combination of both approaches proved to be the most appropriate for modules involving states and control sequences
Counterexamples	Atypical result; the error was recorded as a failure to complete the scenario	Could be caused in the event of a breach of the property or an incomplete description of the environmental constraints	Counterexamples enhanced the explanatory value of the formal verification but required interpretation
Difficulty of application	Increased based on the number of states, transitions and control signals	Increased in tandem with the increasing complexity of formalising the properties and constraints of the environment	Both approaches had limitations, so it is advisable to use them in combination
Scalability	Decreased as the number of possible scenarios and signal combinations increased	Limited by complexity of the state space and the precision of the formal properties	The most comprehensive conclusion was provided by a combined verification strategy

Source: compiled by the author

The most comprehensive conclusion was provided by the combined strategy. The summary presented in Table 5 showed that functional verification best confirmed the behaviour of the digital module in the specified operating modes, whilst formal verification assessed logical properties that could not be reduced to individual scenarios. For the multiplexer, both approaches yielded consistent results, as the verification space was limited. For the finite-state machine and the simplified interface controller, the difference between the approaches became more significant: functional verification confirmed the expected transitions and exchange sequences, whilst formal verification highlighted the need to check for unreachable states, deadlocks, possible counterexamples and control signal conflicts. Consequently, the comparison results confirmed that functional and formal verification do not duplicate one another but rather form a complementary verification strategy for digital systems of varying complexity.

Results of the functional and formal verification showed that neither approach, taken in isolation, provided an equally comprehensive verification conclusion for all types of digital modules. Functional verification proved sufficiently effective for confirming the behaviour of the module in specified modes, particularly at the multiplexer level. At the same time, for the finite state machine and the simplified interface controller, its completeness depended on the selection of sequences, transition conditions and verification scenarios. Formal verification complemented this result, as it was used to verify correctness properties, invariants, the unreachability of erroneous states, the absence of deadlocks, and the consistency of control signals. Based on the verification results for the three test objects, a combined verification strategy was developed for digital modules of varying complexity. Its logic is based on the sequential combination of functional verification of the intended operating modes with formal verification of properties that cannot be fully confirmed by scenario-based testing alone. For simple combinational modules, functional verification provides the bulk of the conclusion, whilst formal verification serves to confirm the basic logical property. For sequential and control modules, formal verification becomes increasingly relevant, as this makes it possible to assess the validity of transitions, the reachability of final states, the absence of deadlocks and the possibility of counterexamples. A generalised diagram illustrating the combined use of functional and formal verification is shown in Figure 1.

Figure 1 summarises a combined verification strategy for digital modules, in which functional and formal verification are applied as complementary stages in reaching a verification conclusion. The verification sequence covers the transition from the module specification and expected operating modes to functional scenarios, formalised properties and the analysis of possible counterexamples. If the functional scenarios pass and the formal properties are verified without counterexamples, the verification result can be considered consistent with the given specification. If a scenario fails, a property violation is detected, or a counterexample is found, the specification, test suite, formal properties or environmental constraints require refinement. Consequently, combined verification has a sequential yet iterative structure, as the verification results may cause the process to revert to earlier stages of refining the digital module. Thus, the combined strategy has made it possible to link the behavioural verification of a digital module with a property-based assessment of its logical correctness. For modules of low complexity, it primarily confirmed the results of functional verification, whilst for more complex sequential and control modules, it enhanced the detection of hidden states, deadlocks, unreachable states and signal conflicts. Therefore, the combined use of functional and formal verification is a sound approach to verifying digital systems of varying structural complexity.

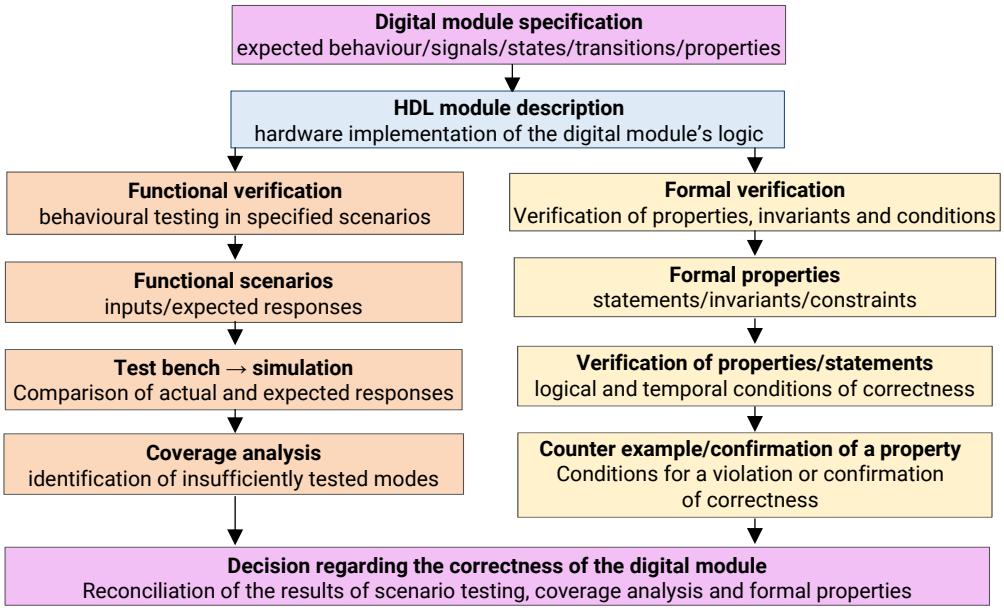

Figure 1. Generalised diagram of the combined use of functional and formal verification

Note: diagram is based on the results of functional and formal verification of the multiplexer, the finite-state machine and the simplified interface controller

Source: compiled by the author

Discussion

A comparison of functional and formal verification showed that the difference between these approaches was determined not only by the verification tools, but also by the type of verification conclusion. Functional verification confirmed the expected behaviour of the digital module in specified scenarios, whilst formal verification was aimed at verifying properties such as correctness, the admissibility of transitions, the unreachability of erroneous states and the absence of deadlocks. As the structural complexity of the digital module increased, scenario-based verification gradually lost its comprehensiveness, as it could not cover all possible combinations of states, transitions and control signals. Therefore, it proved more appropriate not to contrast the functional and formal approaches, but to combine them within a single verification strategy. This finding was consistent with the research by X. Lai *et al.* (2019), in which the verification of hardware systems is viewed as a multidimensional process in which functional and non-functional characteristics cannot be completely separated. This approach explained why the verification of the digital module was not limited to establishing compliance with the expected logic but also involved analysing the conditions under which the module remained correct under various environmental constraints, transition sequences and operating modes. This explained the result obtained, according to which a simplified interface controller proved to be a more complex object for formal analysis than a multiplexer or a finite-state machine: for the controller, it was not only the individual responses to input signals that mattered, but also the control sequence, waiting states, conditions for the completion of an exchange, and the absence of deadlock.

The research findings were also compared with the work of A. Dobis *et al.* (2023), in which the ChiselVerify toolchain is considered as a means of combining dynamic and formal verification of digital designs. In contrast to the study, the presented research did not aim to develop or evaluate a new verification environment. The focus was on how the choice between functional and formal verification depended on the type of digital module. The results showed that, for combinational logic, scenario-based verification of expected responses could be sufficient, whilst for sequential and control logic, there was a growing need to formalise properties. In this respect, the findings of A. Dobis *et al.* confirmed the general trend towards the integration of scenario-based and property-based verification methods.

In the context of complex control structures, the results obtained were similar to the findings of R. Ma *et al.* (2025), who investigated the automation of Network-on-Chip verification within the framework of the UVM. For complex communication structures, conventional scenario-based coverage does not always ensure the detection of hidden combinations of states, transitions and signals. A similar pattern was observed in this study: a simplified interface controller could be functionally verified according to the specified scenarios; however, this was insufficient for a complete analysis of deadlocks, signal conflicts and the reachability of correct final states. A separate aspect of the problem was related to the quality of the formalisation of properties. The paper by V. Radu *et al.* (2024) examines the application of generative artificial intelligence to generate SystemVerilog statements in UVM-based functional verification. For the present study, this approach was relevant not because of the use of artificial intelligence per

se, but because of the problem of correctly defining the property that the system is intended to verify. If a property is formulated incompletely or does not correspond to the actual logic of the module, the verification may yield a formally acceptable but methodologically limited result. Therefore, the effectiveness of formal verification depended not only on the verification tool, but also on the accuracy of the specification of properties, invariants and environmental constraints.

The practical aspect of modelling using the SystemVerilog language was explored in a study by N. Georgoulopoulos *et al.* (2024), which addressed the design and verification of a Successive Approximation Register Analogue-to-Digital Converter model. In contrast to this applied approach, the present study utilised simplified demonstration digital modules: a multiplexer, a finite-state machine and a simplified interface controller. This selection of components isolated the problem of functional-formal correspondence from the subject-matter complexity of the analogue-to-digital model and focused on the change in the verification task during the transition from combinational logic to sequential and control logic. A study by H. Guzmán-Miranda *et al.* (2025) was methodologically similar to the issues under consideration; it proposed a Formal Verification Methodology (FVM) for projects using the VHDL. In this approach, formal verification is regarded not as a one-off procedure, but as an ordered process of checking the hardware description. This clarified the role of the preliminary definition of properties, invariants and correctness conditions in the formal verification of digital modules. At the same time, the study did not replicate the full FVM procedure for VHDL code, as the main focus was on explaining for which types of digital modules formal verification logic carried greater analytical weight. The broader industrial context of formal methods was presented in the publication by M.H. ter Beek *et al.* (2025), where formal methods are considered as a set of tools for the specification, development, analysis and verification of software and hardware systems. In this context, the results of the comparison should not be interpreted as a ready-made industrial verification methodology for a specific hardware project. The study outlined a methodological framework for selecting a verification strategy depending on the complexity of the module and the nature of potential errors. This approach was consistent with the view that formal methods in engineering practice are not applied in isolation, but within the broader process of development, specification and correctness checking.

Assertion-based verification represented a middle ground between scenario-based and property-based verification. This aspect was explored by M.W. Anwar *et al.* (2020), proposing a unified model-driven framework for static and dynamic assertion-based verification. In this approach, assertions served as a linking element between functional scenarios, test coverage and formal properties. Consequently, the combined verification strategy was not merely a matter of mechanically appending formal verification to functional verification. It involved organising the analysis in such a way that scenarios, coverage, properties and counterexamples were used as interrelated grounds for concluding that the digital module was correct. The issue of confidence in the results of formal verification was clarified in the study by Z. Yu *et al.* (2019), which addressed certification of verification results for hardware logic models. The study highlighted the problem of confirming the correctness of the obtained results, which was directly linked to the conclusions of this research. Formal verification cannot be regarded as an automatically infallible verification method, as its evidential strength depends on the correctness of the model, the formulation of properties, environmental constraints and the interpretation of results. Therefore, in the comparison conducted, formal verification was characterised not only by its ability to detect hidden errors, but also by the complexity of formalisation and the need to analyse counterexamples. The study by D. Beyer *et al.* (2024) contributed to the discussion by raising the question of the transferability of algorithmic ideas between hardware and software verification. For this study, this conclusion did not imply that hardware and software verification were the same. Rather, it highlighted the flexibility of verification tools and the possibility of combining different methods depending on the object under verification. The results also demonstrated that the boundaries between functional verification, assertion-based verification, model checking and formal property specification are not rigid. Combining these approaches was justified when the choice of method was linked to the structural complexity of the digital module, the type of logic and the nature of the expected faults. In the work by R. Mukherjee *et al.* (2017), formal techniques were applied to the joint verification of hardware-software solutions, where the software component interacted with hardware logic described at the Register-Transfer Level (RTL). For this study, the paper explained the extension of the verification problem: the complexity of a digital module can increase not only due to the number of states or control signals, but also due to the interaction between hardware and software behaviour. In contrast to the joint verification of hardware-software solutions, the analysis in this article was limited to digital modules of varying structural complexity. This differentiated between scenario-based and property-based verification logic. In the context of critical systems, formal verification became not only a methodological tool but also a means of establishing trust. W. Khan *et al.* (2020) examined the formal specification and verification of hardware components, in particular memory blocks and adders, using the Coq theorem prover to work with Boolean algebra and digital circuits. A comparison with this approach clarified that, for hardware components of critical systems, scenario-based verification of expected behaviour is not always sufficient. If a fault in an arithmetic or memory component can affect the operation of a critical system, then proving correctness properties becomes a separate condition for trust in the project. In the present study, this conclusion was

extended to the level of structural complexity of digital modules: as the complexity of a module's logic increased, so did the need not only for scenario-based coverage but also for a formalised description of properties.

The comparative aspect of formal methods was outlined in a study by R. Karmakar (2022), which examined the Z, B and Event-B techniques in the context of critical systems design. This approach demonstrated that formal verification is not a uniform procedure that can be mechanically contrasted with functional testing. It encompasses various methods of specification, modelling and proof, and its effectiveness depends on the soundness of the choice of representation for the properties. For this reason, the comparative table characterised formal verification not only in terms of its ability to detect hidden errors, but also in terms of the complexity of formalisation and its dependence on the quality of the initial specification. Functional verification, despite its inherent limitations, has retained a practical role in engineering tasks. This was evident in the study by C. Elakkiya *et al.* (2017), which addressed the functional verification of the Joint Test Action Group interface based on functional coverage and UVM. This example maintained a methodological balance: functional verification was not regarded as a less suitable verification method but was treated as a tool for reproducing expected operating modes. Its limitations became apparent when the number of possible scenarios, states and transitions exceeded the capabilities of full manual or semi-automated coverage. Therefore, the combined strategy did not exclude a functional coverage-oriented approach but rather defined the conditions under which it should be supplemented by property verification. The automation of formal verification of module interactions was linked to the problem of the high threshold for applying formal verification. M. Orenes-Vera *et al.* (2021) proposed AutoSVA as a basis for the automatic generation of formal verification test environments that analyse the survivability and safety properties of control logic in module interactions at the register-transfer level. In the study, this approach explained the dual nature of formal verification: it offered greater diagnostic power for complex module interactions, but at the same time required the precise formulation of properties, invariants and environment constraints. This was consistent with the finding that a simplified interface controller required a broader application of property-based analysis than a multiplexer or a finite state machine.

The challenge of transitioning from a description at the register-transfer level to a formal model was explored in detail in the study by Y. Hsiao *et al.* (2021). The authors proposed a methodology and the rtl2μspec tool for the automatic synthesis of μspec models from processor designs written in Verilog, for subsequent use in verifying the Memory Consistency Model. This approach demonstrated that the formal verification of complex hardware logic often requires an intermediate representation between the initial register-transfer-level description and the property to be proven. In the study conducted, this problem was not solved instrumentally, but it did define the limits of the experimental and demonstrative verification: the comparison of a multiplexer, a finite-state machine and an interface controller provided a methodological explanation for the increase in complexity, whilst real-world industrial verification required additional means of transformation, abstraction and formalisation of the hardware description. Security verification of processors extended the discussion beyond the general correctness of a digital module's behaviour. C.S. Chuah *et al.* (2023) formally verified the security-critical functions of a commercial processor based on the Reduced Instruction Set Computer Five (RISC-V) architecture, using formal verification based on model checking. This example illustrated why formal verification becomes necessary in cases where an error has not only functional but also safety implications. In contrast to domain-specific safety verification, the study considered a generalised model of digital modules of varying complexity. At the same time, the example of RISC-V processors confirmed that, for complex control logic, it is not sufficient to limit oneself to checking expected scenarios, as some critical violations may only manifest through properties related to access, privileges, states or invalid sequences.

A review of the approaches discussed has shown that functional and formal verification should not be interpreted as mutually exclusive methods of verifying digital modules. Functional verification provided a link to specific operating scenarios, test coverage and expected responses, whilst formal verification reinforced the analysis of properties, invariants, valid transitions and potential counterexamples. The most sound approach proved to be their combined use, whereby the choice of verification tools depended on the structural complexity of the module, the nature of possible errors and the requirements for the provability of the verification conclusion. This approach highlighted verification not as a set of isolated procedures, but as a coordinated strategy for assessing the correctness of hardware logic.

Conclusions

The study provided a theoretical justification for the nature of functional and formal verification of digital systems and identified specific features of their application to digital modules of varying structural complexity. The study established that functional verification is focused on scenario-based verification of a digital module's behaviour through test inputs, expected output responses, test benches, simulation and analysis of functional mode coverage. Formal verification is aimed at verifying the correctness properties of hardware logic, invariants, logical and temporal conditions, valid transitions and the unreachability of erroneous states. The results obtained confirmed that these approaches are not interchangeable, as functional verification yields a behavioural conclusion within the scope of specified scenarios, whilst formal verification yields a property-based conclusion regarding the logical correctness of the hardware description. The normative and methodological distinction between functional and

formal verification provided grounds for clarifying that the verification of digital modules should not be equated with general software testing. The effectiveness of digital module verification depended not only on the number of test scenarios, but also on the completeness of the specification, the accuracy of property formalisation, the correctness of environmental constraints, and the ability to cover boundary or hidden operating modes.

Experimental and demonstration-based verification on a multiplexer, a finite-state machine and a simplified interface controller showed that the nature of the verification task changed as the complexity of the digital module increased. For the multiplexer, functional testing ensured coverage of the basic modes of logical selection, whilst formal verification confirmed the property that the output corresponds to the active input. For the finite state machine, the significance of testing the reset condition, valid transitions, output responses and the unreachability of prohibited states increased. For the simplified interface controller, the "start", "wait", "ack", "done" and "error" scenarios confirmed the correctness of the main exchange sequence and the response to an error condition, but did not eliminate the risk of hidden states, deadlocks and control signal conflicts. A comparison of the results of functional and formal verification showed that functional verification was effective in confirming the module's behaviour in the specified modes; however, its completeness diminished as the complexity of the hardware logic increased. Formal verification strengthened the verification conclusion through the analysis of properties, invariants, the admissibility of transitions, the unreachability of false states, the absence of deadlocks, the consistency of control signals, and the possibility of generating counterexamples in the event of a violation of the specified conditions. At the same time, the result of formal verification depended on the accuracy of the specification, the completeness of the formalisation of properties, and the correct specification of environmental constraints.

A combined strategy, in which functional scenarios, coverage analysis, formal properties and the interpretation of possible counterexamples complement one another, has been identified as the most sound approach to verifying digital modules of varying structural complexity. For simple combinational modules, this strategy predominantly confirmed the results of scenario-based verification, whilst for sequential and control modules, it improved detection of hidden states, unreachable states, deadlocks and signal conflicts. The limitations of the study stem from its experimental and demonstrative nature, as the results were obtained on three typical digital modules. Further research should focus on testing the proposed combined strategy on more complex hardware systems, industrial HDL projects and full-scale verification environments. Other promising avenues include expanding the set of test modules, utilising UVM environments, analysing the automated generation of formal assertions, and conducting an in-depth study of counterexamples in complex interface and control structures.

Acknowledgements
None.

Funding
The study received no funding.

Conflict of Interest
None.

References
[1] Abdollahi, M., Yeganeh, S.F., Baharloo, M.A., & Baniasadi, A. (2025). Hardware design and verification with large language models: A scoping review, challenges, and open issues. *Electronics*, 14(1), article number 120. doi: 10.3390/electronics14010120.

[2] Anwar, M.W., Rashid, M., Azam, F., Naeem, A., Kashif, M., & Butt, W.H. (2020). A unified model-based framework for the simplified execution of static and dynamic assertion-based verification. *IEEE Access*, 8, 104407-104431. doi: 10.1109/ACCESS.2020.2999544.

[3] Beyer, D., Chien, P.-C., Jankola, M., & Lee, N.-Z. (2024). A transferability study of interpolation-based hardware model checking for software verification. *Proceedings of the ACM on Software Engineering*, 1, article number 90. doi: 10.1145/3660797.

[4] Cabodi, G., Camurati, P.E., Palena, M., & Pasini, P. (2024). Hardware model checking algorithms and techniques. *Algorithms*, 17(6), article number 253. doi: 10.3390/a17060253.

[5] Chuah, C.S., Appold, C., & Leinmueller, T. (2023). Formal verification of security properties on RISC-V processors. In *MEMOCODE '23: Proceedings of the 21st ACM-IEEE international conference on formal methods and models for system design* (pp. 159-168). New York: Association for Computing Machinery. doi: 10.1145/3610579.3611085.

[6] Dobis, A., Laeufer, K., Damsgaard, H.J., Petersen, T., Rasmussen, K.J.H., Tolotto, E., Andersen, S.T., Lin, R., & Schoeberl, M. (2023). Verification of Chisel hardware designs with ChiselVerify. *Microprocessors and Microsystems*, 96, article number 104737. doi: 10.1016/j.micpro.2022.104737.

[7] Elakkiya, C., Murty, N.S., Babu, C., & Jalan, G. (2017). Functional coverage – driven UVM based JTAG verification. In *2017 IEEE international conference on computational intelligence and computing research* (pp. 1-7). New York: IEEE. doi: 10.1109/ICCIC.2017.8524556.

[8] Farooq, U., & Mehrez, H. (2021). Pre-silicon verification using multi-FPGA platforms: A review. *Journal of Electronic Testing*, 37, 7-24. doi: 10.1007/s10836-021-05929-1.

[9] Georgoulopoulos, N., Mamali, T., & Hatzopoulos, A.A. (2024). Design and verification of a SAR ADC SystemVerilog real number model. *Journal of Electronic Testing*, 40, 315-328. doi: 10.1007/s10836-024-06124-8.
[10] Guzmán-Miranda, H., García, M.L., & Aguado, A.U. (2025). FVM: A formal verification methodology for VHDL designs. *IEEE Open Journal of the Computer Society*, 6, 1920-1933. doi: 10.1109/OJCS.2025.3625468.
[11] Hsiao, Y., Mulligan, D.P., Nikoleris, N., Petri, G., & Trippel, C. (2021). Synthesizing formal models of hardware from RTL for efficient verification of memory model implementations. In *MICRO-54: Proceedings of the 54th annual IEEE/ACM international symposium on microarchitecture* (pp. 679-694). New York: Association for Computing Machinery. doi: 10.1145/3466752.3480087.
[12] IEEE 1012-2024. (2025). *IEEE standard for system, software, and hardware verification and validation*. Retrieved from https://standards.ieee.org/ieee/1012/7324/.
[13] IEEE 1076-2019. (2019). *IEEE standard for VHDL language reference manual*. Retrieved from https://surl.lt/dzmkxp.
[14] IEEE 1800.2-2020. (2020). *IEEE standard for Universal Verification Methodology language reference manual*. Retrieved from https://standards.ieee.org/ieee/1800.2/7567.
[15] IEEE 1800-2023. (2024). *IEEE standard for SystemVerilog – unified hardware design, specification, and verification language*. Retrieved from https://standards.ieee.org/ieee/1800/7743.
[16] IEEE 1850-2010. (2010). *IEEE standard for Property Specification Language (PSL)*. Retrieved from https://standards.ieee.org/ieee/1850/4383/.
[17] ISO/IEC/IEEE 24765:2017. (2017). *Systems and software engineering – vocabulary*. Retrieved from https://www.iso.org/standard/71952.html.
[18] ISO/IEC/IEEE 29119-1:2022. (2022). *Software and systems engineering – software testing. Part 1: General concepts*. Retrieved from https://www.iso.org/standard/81291.html.
[19] Karmakar, R. (2022). Formal verification techniques: A comparative analysis for critical system design. In A. Abraham, N. Gandhi, T. Hanne, T.-P. Hong, T.N. Rios & W. Ding (Eds.), *21st international conference on intelligent systems design and applications (ISDA 2021): Intelligent systems design and applications* (pp. 93-102). Cham: Springer. doi: 10.1007/978-3-030-96308-8_9.
[20] Kern, C., & Greenstreet, M.R. (1999). Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 4(2), 123-193. doi: 10.1145/307988.307989.
[21] Khan, W., Kamran, M., Naqvi, S.R., Khan, F.A., Alghamdi, A.S., & Alsolami, E. (2020). Formal verification of hardware components in critical systems. *Wireless Communications and Mobile Computing*, 2020(1), article number 7346763. doi: 10.1155/2020/7346763.
[22] Lai, X., Balakrishnan, A., Lange, T., Jenihhin, M., Ghasempouri, T., Raik, J., & Alexandrescu, D. (2019). Understanding multidimensional verification: Where functional meets non-functional. *Microprocessors and Microsystems*, 71, article number 102867. doi: 10.1016/j.micpro.2019.102867.
[23] Ma, R., Huang, J., Zhang, S., Xie, Y., & Luo, G. (2025). NoCFuzzer: Automating NoC verification in UVM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(1), 371-384. doi: 10.1109/TCAD.2024.3430195.
[24] Mehta, A.B. (2018). *ASIC/SoC functional design verification: A comprehensive guide to technologies and methodologies*. Cham: Springer. doi: 10.1007/978-3-319-59418-7.
[25] Moon, D.-W., Pyo, S.-H., Hong, D.-K., Bataa, O., & Norinpel, E. (2025). Assertion-based verification of I2C module using SystemVerilog. *Electronics*, 14(8), article number 1687. doi: 10.3390/electronics14081687.
[26] Mukherjee, R., Purandare, M., Polig, R., & Kroening, D. (2017). Formal techniques for effective co-verification of hardware/software co-designs. In *Proceedings of the 54th annual design automation conference 2017* (article number 35). New York: Association for Computing Machinery. doi: 10.1145/3061639.3062253.
[27] Odarushchenko, O., Letychevskyi, O., Shamanskyi, V., Babeshko, I., Peschanenko, V., & Striuk, O. (2024). Information technology of formal verification and design of FPGA electronic projects. In *2024 14th international conference on dependable systems, services and technologies* (pp. 1-6). New York: IEEE. doi: 10.1109/DESSERT65323.2024.11122210.
[28] Orenes-Vera, M., Manocha, A., Wentzlaff, D., & Martonosi, M. (2021). AutoSVA: Democratizing formal verification of RTL module interactions. In *2021 58th ACM/IEEE design automation conference* (pp. 535-540). New York: IEEE. doi: 10.1109/DAC18074.2021.9586118.
[29] Radu, V., Dranga, D., Dumitrescu, C., Tabirca, A.I., & Stefan, M.C. (2024). Generative AI assertions in UVM-based System Verilog functional verification. *Systems*, 12(10), article number 390. doi: 10.3390/systems12100390.
[30] ter Beek, M.H., *et al*. (2025). Formal methods in industry. *Formal Aspects of Computing*, 37(1), article number 7. doi: 10.1145/3689374.
[31] Witharana, H., Lyu, Y., Charles, S., & Mishra, P. (2022). A survey on assertion-based hardware verification. *ACM Computing Surveys,* 54(11), article number 225. doi: 10.1145/3510578.
[32] Wu, N., Li, Y., Yang, H., Chen, H., Dai, S., Hao, C., Yu, C., & Xie, Y. (2024). Survey of machine learning for software-assisted hardware design verification: Past, present, and prospect. *ACM Transactions on Design Automation of Electronic Systems*, 29(4), article number 59. doi: 10.1145/3661308.

[33] Yu, Z., Biere, A., & Heljanko, K. (2019). Certifying hardware model checking results. In Y. Ait-Ameur & S. Qin (Eds.), *21st international conference on formal engineering methods: Formal methods and software engineering* (pp. 498-502). Cham: Springer. doi: 10.1007/978-3-030-32409-4_32.

Методи функціональної та формальної верифікації цифрових систем: теоретичний аналіз та експериментальне застосування до модулів різної складності

Олександр Острянко

Магістр, аспірант
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
03056, просп. Берестейський, 37, м. Київ, Україна
https://orcid.org/0009-0008-9107-4254

Анотація. Метою дослідження було теоретичне обґрунтування сутності функціональної та формальної верифікації цифрових систем, порівняння їхніх методів і визначення можливостей узгодженого застосування до цифрових модулів різної структурної складності. Методологія ґрунтувалася на нормативно-термінологічному, структурно-аналітичному, функціонально-верифікаційному, формально-логічному та порівняльно-аналітичному методах, що дало змогу розмежувати сценарний і властивісний підходи до перевірки апаратної логіки та здійснити експериментально-демонстраційну перевірку трьох тестових цифрових модулів. Установлено, що функціональна верифікація забезпечувала перевірку поведінкової відповідності цифрового модуля специфікації через тестові впливи, очікувані вихідні реакції, тестбенчі, симуляцію та аналіз покриття функціональних режимів. Формальна верифікація формувала інший тип перевірочного висновку, оскільки була спрямована на підтвердження або спростування властивостей коректності, інваріантів, логічних і часових умов, допустимих переходів та недосяжності помилкових станів. Експериментально-демонстраційна перевірка на мультиплексорі, скінченному автоматі та спрощеному інтерфейсному контролері показала, що зі зростанням складності цифрового модуля збільшувалася кількість функціональних сценаріїв, ускладнювалося покриття режимів і зростав ризик невиявлення прихованих станів, блокувань або рідкісних комбінацій сигналів. Для мультиплексора обидва підходи дали узгоджений результат; для скінченного автомата зросло значення перевірки скидання, станів і переходів; для інтерфейсного контролера формальна перевірка посилила оцінку блокувань, конфліктів сигналів і помилкових режимів. Обґрунтовано доцільність комбінованої стратегії, у якій функціональні сценарії, аналіз покриття, формальні властивості та інтерпретація можливих контрприкладів взаємно доповнюють один одного. Практичне значення результатів полягає в їх використанні в цифровому проєктуванні, апаратній верифікації та навчально-дослідних середовищах для планування перевірки цифрових модулів, уточнення специфікацій, аналізу покриття, інтерпретації контрприкладів і прийняття рішення щодо коректності апаратної логіки

Ключові слова: апаратна логіка; тестові сценарії; аналіз покриття; властивості коректності; контрприклади; керувальні сигнали