Sarcouncil Journal of Engineering and Computer Sciences

ISSN(Online): 2945-3585

Volume- 04| Issue- 12| 2025

Research Article

Received: 08-11-2025| Accepted: 03-12-2025 | Published: 18-12-2025

Co-Verification of Multi-Protocol Controllers for Integrated Communication Systems

Jena Abraham

AMD, USA

Abstract: Multi-protocol controllers handling SPI, I2C, UART, and CAN interfaces need validation methods that assess hardware register-transfer logic together with software drivers and firmware at the same time. Contemporary system-on-chip designs frequently incorporate these controllers to minimize chip area while preserving communication flexibility. Verification complexity arises because multiple protocols share registers and buffers yet maintain distinct timing and signaling requirements. Co-simulation techniques enable engineers to replicate production environments during pre-silicon validation by combining hardware models with software execution contexts. Traditional isolated testing often misses register configuration errors, protocol sequence violations, and interface mismatches that co-verification readily exposes. Testbench frameworks execute realistic transaction sequences while monitoring protocol specification adherence and verifying correct handshaking between hardware state machines and software control paths. Engineers identify edge cases, error conditions, and concurrent protocol scenarios before fabrication, building confidence in controller reliability. The integrated testing process establishes functional alignment between silicon designs and driver implementations, reducing downstream integration failures. Pre-silicon validation through combined hardware-software assessment helps teams detect timing issues and interface incompatibilities early. Controllers verified through these comprehensive methods demonstrate proven reliability across operational configurations, meeting functional requirements for diverse deployment scenarios.

Keywords: Co-Verification, Multi-Protocol Controllers, Hardware-Software Integration, System-on-Chip Validation, Communication Protocols.

INTRODUCTION

System-on-chip designs now require controllers that manage multiple communication protocols within a single hardware implementation. Multi-protocol controllers support various interfaces such as SPI, I2C, UART, and CAN concurrently while reducing silicon area and power requirements. These complex designs need validation strategies that evaluate hardware register-transfer logic and software driver components together. Conventional verification methods, which test hardware and software separately, cannot detect integration problems that arise when these layers operate during real system conditions (Roth, S. *et al.*, 2023). Controllers handling multiple protocols face distinct challenges from shared resource usage, including registers, buffers, and arbitration mechanisms that must perform correctly across various timing domains and signaling specifications.

Verification engineers experience growing demands to produce dependable controller implementations as system-on-chip designs include more advanced communication features. Expanding Internet of Things devices, automotive systems, and industrial automation platforms create requirements for adaptable controllers supporting diverse protocol combinations.

Pre-silicon validation must expose defects before fabrication commits designs to expensive manufacturing processes where corrections become prohibitively costly. Co-verification methodologies address these requirements by simulating hardware behavior together with firmware and driver execution, enabling engineers to validate complete functional scenarios that reflect production deployment conditions (Petrochenkov, M. V. *et al.*, 2018). This integrated testing philosophy departs from conventional isolated verification approaches by exercising hardware-software interactions through realistic transaction sequences.

The evolution toward multi-protocol controller architectures stems from economic and technical motivations to consolidate communication functions within compact silicon implementations. System designers prefer controllers capable of reconfiguring protocol support dynamically rather than implementing separate dedicated interfaces for each communication standard. However, this architectural flexibility introduces verification complexity as controllers must maintain correct behavior across protocol modes while managing resource contention and timing constraints. Validation strategies must verify not only individual protocol compliance but also transitions between operational modes and concurrent multi-protocol scenarios. Engineers employ co-simulation platforms combining register-transfer

*Corresponding Author: Jena Abraham

DOI- https://doi.org/10.5281/zenodo.17980539

logic models with executable software to replicate these complex interactions during pre-silicon phases. The verification process encompasses protocol timing validation, register configuration correctness, and software interface functionality, establishing confidence in controller reliability before physical implementation. Contemporary validation methodologies recognize that hardware and software co-design demands corresponding co-verification approaches, ensuring seamless integration across abstraction layers.

Multi-Protocol Controller Architecture

Multi-protocol controllers implement hardware architectures supporting multiple communication standards through shared functional blocks and configurable protocol engines. These controllers typically incorporate programmable state machines, configurable clock generators, protocol-specific encoding logic, and unified buffer management systems. The architecture enables dynamic reconfiguration between protocols such as SPI, I2C, UART, and CAN by adjusting timing parameters, signal polarities, and data formatting characteristics through software-accessible registers (Roth, S. *et al.,* 2023). Hardware designers balance resource sharing against protocol-specific requirements, creating flexible implementations that minimize silicon area while maintaining compliance with diverse communication standards. Controllers employ arbitration mechanisms managing access to shared resources when multiple protocols operate concurrently or switch operational modes dynamically.

Register interfaces provide software control over protocol selection, timing configuration, interrupt management, and data transfer operations. Configuration registers determine clock dividers, bit ordering, frame formats, and error detection mechanisms appropriate for selected protocols. Buffer architectures support variable data packet sizes across different communication standards while managing flow control and handshaking requirements. State machine implementations handle protocol-specific sequence requirements, including start conditions, address phases, data transfers, and stop sequences unique to each supported standard (Halevi, Y., & Ray, A. 1988).

The architecture must accommodate timing variations spanning from high-speed SPI transfers to relatively slower I2C transactions while maintaining signal integrity and protocol compliance. Controllers integrate status registers exposing error conditions, transmission states, and buffer occupancy levels to software drivers. Interrupt generation logic provides event notification for transaction completion, error conditions, and buffer thresholds, enabling efficient software interaction patterns. The modular architecture separates protocol-specific logic from common functions such as clock management and buffer control, facilitating verification of individual components while ensuring integrated operation. Physical interface blocks manage electrical characteristics, signal conditioning, and impedance matching requirements for different communication standards. Clock domain crossing logic handles timing relationships between controller core frequencies and protocol-specific bit rates, ensuring data integrity across asynchronous boundaries. The architectural design directly impacts verification complexity as shared resources require validation across multiple operational contexts and protocol combinations.

Need for Integrated Hardware-Software Verification

Isolated hardware or software verification approaches prove insufficient for validating multi-protocol controllers due to critical dependencies between register-transfer logic behavior and driver implementations. Hardware simulation alone cannot fully exercise controllers through realistic transaction sequences that software generates during production operation. Similarly, software testing without accurate hardware models fails to expose timing-sensitive defects, register configuration errors, and protocol compliance issues (Petrochenkov, M. V. *et al.,* 2018). Integration problems frequently emerge when separately verified hardware and software components interact, manifesting as timing violations, incorrect state transitions, or data corruption that isolated testing methodologies miss. Co-verification addresses these limitations by simulating hardware and software together, enabling validation of complete functional scenarios reflecting actual system deployment conditions.

The economic implications of post-silicon defect discovery motivate comprehensive pre-silicon verification investments. Fabrication costs for advanced process nodes make silicon respins prohibitively expensive, while schedule delays from late-stage defect corrections jeopardize market opportunities. Controllers deployed in safety-critical applications such as automotive

Publisher: SARC Publisher

systems or medical devices demand rigorous validation, ensuring reliable operation across all supported protocols and operational scenarios. Regulatory compliance requirements in these domains necessitate documented verification evidence demonstrating thorough testing coverage (González, I. *et al.*, 2017). Co-verification provides this assurance by exercising hardware-software interactions through testbench environments executing production-representative workloads. The methodology validates not only individual protocol correctness but also driver interaction patterns, interrupt handling sequences, and error recovery mechanisms.

Modern development practices emphasize parallel hardware and software engineering to compress product schedules. Co-verification enables this concurrent development by providing simulation environments where software teams validate drivers against evolving hardware implementations before silicon availability. Engineers detect interface specification errors, register definition inconsistencies, and behavioral mismatches early in development cycles when corrections remain straightforward. The integrated testing approach builds confidence across engineering teams that hardware and software components will integrate successfully in physical systems. Testbench platforms combining register-transfer logic simulation with software execution establish functional alignment between layers, reducing integration risk and accelerating time-to-market objectives for complex multi-protocol controller implementations.

Table 1: Benefits of Multi-Protocol Controller Co-Verification (Roth, S. *et al.*, 2023; Petrochenkov, M. V. *et al.*, 2018)

Benefit Category	Description
Early Defect Detection	Co-verification identifies protocol handling errors and register configuration issues during pre-silicon phases, preventing costly post-fabrication corrections and reducing development cycle time
Hardware-Software Alignment	Simultaneous testing establishes functional consistency between register-transfer logic behavior and software driver expectations, eliminating integration mismatches
Accelerated Development	Parallel validation of hardware and software components enables concurrent engineering activities, shortening time-to-market schedules for system-on-chip products
Protocol Compliance Assurance	Integrated testing verifies adherence to communication protocol specifications across SPI, I2C, UART, and CAN interfaces before physical implementation
Resource Optimization	Combined verification reduces redundant testing efforts by validating shared controller resources and buffer mechanisms across multiple protocol modes simultaneously
Production Scenario Coverage	Testbench environments executing realistic transaction sequences ensure controllers operate correctly under actual deployment conditions and use cases

HARDWARE SIMULATION AND RTL VERIFICATION

Hardware simulation forms the foundation of multi-protocol controller validation by modeling register-transfer logic behavior before physical silicon fabrication. Simulation environments execute digital designs represented in hardware description languages, enabling engineers to observe signal transitions, state machine progressions, and data flow patterns across clock cycles. Register-transfer logic models capture controller functionality at a level where individual registers, combinational logic blocks, and sequential elements interact according to design specifications (Halevi, Y., & Ray, A. 1988). Verification teams construct testbenches that apply input stimuli to controller models while monitoring output responses, comparing observed behavior against expected results derived from protocol specifications and architectural requirements. Simulation provides detailed visibility into internal controller states that remain inaccessible in physical implementations, facilitating root cause analysis when defects appear.

The simulation process encompasses functional verification, confirming correct logical operation and timing verification, and validating setup and hold requirements across clock domains. Engineers employ waveform analysis tools to examine signal relationships, detect race conditions, and identify timing violations that could cause functional failures in manufactured silicon. Register-transfer logic simulation enables

Publisher: SARC Publisher

exploration of corner cases and error scenarios difficult to reproduce in physical hardware testing (Bayılmış, C. *et al.,* 2022). Testbench architectures generate protocol transactions exercising controller state machines through various operational sequences, including normal operations, boundary conditions, and fault injection scenarios. Coverage metrics guide verification completeness by tracking exercised code paths, visited states, and toggled signals, identifying gaps requiring additional test scenarios.

Simulation platforms support hierarchical verification strategies where individual controller modules undergo isolated testing before integration into complete system models. Protocol engines, buffer management units, clock generation circuits, and interrupt controllers receive focused validation, ensuring correct standalone operation. Integration testing validates interactions among modules, verifying correct handshaking, data transfer, and error propagation between module interfaces. Hardware simulation supports iterative design refinement as engineers implement modifications to resolve identified defects, while regression testing confirms corrections avoid introducing additional issues. The methodology provides rapid feedback cycles compared to physical prototyping, accelerating design convergence toward functional correctness while maintaining detailed observability into controller behavior across diverse operational scenarios and protocol modes.

Register-Transfer Logic Validation

Register-transfer logic validation verifies the correct implementation of controller functionality at the architectural level, where data movements between registers and transformations through combinational logic define system behavior. Verification engineers analyze register definitions, ensuring proper bit field allocations, access permissions, and reset values align with specification documents. Configuration registers controlling protocol selection, timing parameters, and operational modes require validation confirming software writes produce intended hardware responses (González, I. *et al.,* 2017). Status registers reflecting controller states, error conditions, and transaction progress need verification to ensure accurate real-time hardware state reporting to software drivers. Register access validation includes checking read-write permissions, reserved bit handling, and side effects such as interrupt flag clearing or transaction triggering.

Combinational logic verification confirms correct data path operations, including multiplexing, encoding, decoding, and arithmetic functions supporting protocol operations. Sequential logic validation examines state machine implementations managing protocol sequences, buffer controls, and arbitration mechanisms. Engineers verify state transitions occur correctly based on input conditions, ensuring controllers progress through protocol phases properly (Lian, K. Y. *et al.,* 2013). Reset behavior validation confirms controllers initialize to known states, with all registers assuming specified default values, enabling deterministic startup behavior. Clock domain crossing verification checks data transfer mechanisms between asynchronous clock domains, validating synchronization circuits to prevent metastability and data corruption.

Timing analysis at the register-transfer level identifies critical paths limiting maximum operating frequencies and verifies setup and hold time requirements across all register transfers. Engineers examine propagation delays through combinational logic chains, ensuring signals stabilize before subsequent clock edges capture values. Timing constraints derived from protocol specifications translate into register-transfer logic requirements that validation confirms. Assertion-based verification embeds properties directly in hardware descriptions, providing continuous monitoring for invariant violations during simulation. Formal verification techniques mathematically prove properties hold across all possible input combinations, complementing simulation-based validation with exhaustive correctness guarantees for critical controller functions. Register-transfer logic validation establishes confidence that hardware implementations match architectural specifications before proceeding to gate-level synthesis and physical implementation stages.

Protocol Compliance Verification

Protocol compliance verification validates that multi-protocol controllers adhere to communication standard specifications defining electrical characteristics, timing requirements, and transaction sequences. Each supported protocol imposes distinct rules governing signal levels, bit timing, frame formats, and error handling that controllers must implement correctly. SPI protocol verification confirms correct clock polarity, phase relationships, and data serialization matching master or slave operational modes (Bayılmış, C. *et al.,* 2022). I2C validation checks start and stop

Publisher: SARC Publisher

condition generation, acknowledge bit handling, clock stretching support, and multi-master arbitration mechanisms. UART verification examines baud rate accuracy, start bit detection, parity generation, stop bit timing, and flow control signaling. CAN protocol validation assesses bit stuffing, cyclic redundancy checks, arbitration procedures, and error frame generation.

Verification testbenches generate protocol-compliant transactions exercising controllers through nominal operational sequences and error conditions specified in protocol standards. Engineers verify timing parameters, including setup times, hold times, bit periods, and inter-frame gaps, match specification requirements across supported data rates (Li, Y. *et al.*, 2025). Signal integrity verification confirms output drivers produce valid logic levels with appropriate transition times and impedance characteristics. Input receiver validation checks controllers correctly interpret signal levels, tolerate noise margins, and handle edge cases such as glitches or voltage variations. Protocol state machine verification ensures controllers sequence through defined states properly, generating correct responses to received commands and handling exceptional conditions per protocol specifications.

Compliance testing includes validating error detection and recovery mechanisms that each protocol defines. Controllers must generate appropriate error signals when detecting violations such as framing errors, parity mismatches, acknowledge failures, or checksum inconsistencies. Error handling verification confirms controllers respond correctly to fault conditions, either retrying transactions, reporting errors to software, or entering safe states depending on protocol requirements. Interoperability testing validates that controllers communicate successfully with reference implementations and commercial devices implementing the same protocols. Waveform comparison against golden reference captures from protocol analyzers provides an objective compliance assessment. Coverage analysis tracks exercised protocol features, ensuring testbenches explore all operational modes, data patterns, and error scenarios defined in communication standards, establishing comprehensive compliance validation before controller deployment in production systems.

Table 2: Multi-Protocol Communication Standards and Characteristics (Bayılmış, C. *et al.*, 2022; Lian, K. Y. *et al.*, 2013).

Protocol Standard	Key Characteristics and Verification Requirements
Serial Peripheral Interface (SPI)	Full-duplex synchronous communication supporting high-speed data transfers up to several megabits per second, with configurable clock polarity and phase, requiring validation of master-slave configurations and multi-slave selection mechanisms
Inter-Integrated Circuit (I2C)	Multi-master synchronous serial protocol using two-wire interface supporting address-based device selection with clock stretching and arbitration mechanisms requiring verification of acknowledge signaling and bus contention resolution
Universal Asynchronous Receiver-Transmitter (UART)	Asynchronous serial communication with configurable baud rates, data bits, parity, and stop bits requires validation of frame timing accuracy and flow control mechanisms across various configuration combinations
Controller Area Network (CAN)	Multi-master protocol designed for automotive and industrial applications, supporting message-based communication with priority arbitration requiring verification of bit stuffing, cyclic redundancy checks, and error frame handling
Protocol Timing Validation	Each communication standard imposes distinct timing constraints, including setup times, hold times, bit periods, and inter-frame gaps that verification must confirm across all supported data rates and operational modes
Error Detection Mechanisms	Protocols implement various error detection approaches, including parity checks, cyclic redundancy codes, and acknowledgment mechanisms, requiring validation of correct error identification and recovery procedures

SOFTWARE STACK INTEGRATION AND TESTING

Software stack integration validates that firmware and driver implementations interact correctly with multi-protocol controller hardware through defined programming interfaces. Driver software manages controller configuration, initiates protocol transactions, handles interrupts, and processes data transfers between applications and hardware

Publisher: SARC Publisher

interfaces. Firmware running on embedded processors coordinates controller operations within larger system contexts, implementing protocol stacks and managing communication sequences (Lian, K. Y. *et al.*, 2013). Integration testing exercises these software components against hardware models or physical prototypes, verifying functional correctness across realistic usage scenarios. The validation process confirms register access sequences produce intended hardware behaviors, interrupt handlers respond appropriately to controller events, and data buffers transfer information reliably between software and hardware domains.

Testing methodologies encompass unit testing of individual driver functions, integration testing of complete driver stacks, and system-level validation incorporating application workloads. Engineers verify initialization sequences and properly configure controllers for selected protocols, setting timing parameters, enabling interrupts, and allocating buffer resources (Wen, D. *et al.*, 2024). Runtime testing validates that transaction initiation, data transmission, reception handling, and error recovery procedures operate correctly under various load conditions. Software validation environments range from simulation platforms executing firmware against register-transfer logic models to hardware-in-the-loop configurations running production code on physical controllers, establishing confidence that firmware and drivers reliably control multi-protocol hardware across diverse operational contexts.

Table 3: Co-Simulation Platform Components and Capabilities (Gielis, J. *et al.*, 2022)

Platform Component	Functionality and Verification Support
Hardware Simulation Engine	Executes register-transfer logic models capturing controller behavior at clock-cycle accuracy while providing signal-level visibility for waveform analysis and timing verification across all hardware states
Software Execution Environment	Runs firmware and driver code through instruction-set simulators or virtual platforms, enabling software development and validation before physical hardware availability, with debugging capabilities
Transaction-Level Modeling	Abstracts low-level signal timing while maintaining functional accuracy to accelerate simulation performance for complex scenarios involving lengthy protocol sequences and multi-transaction operations
Testbench Automation Framework	Implements Universal Verification Methodology or SystemVerilog constructs providing reusable verification components, including drivers, monitors, scoreboards, and coverage collectors for systematic validation
Constrained Random Generation	Creates diverse test patterns through randomization within specified constraints, exploring broader state spaces than directed testing while maintaining protocol compliance and scenario relevance
Coverage Analysis Tools	Tracks code coverage, functional coverage, and assertion coverage, identifying verification gaps and guiding test development toward comprehensive validation of controller features and operational modes
Debug and Analysis Capabilities	Provides waveform viewers, signal tracers, breakpoint functionality, and cross-layer debugging, enabling efficient root cause analysis for defects spanning hardware-software boundaries in integrated environments

CO-SIMULATION PLATFORMS AND TESTBENCH ENVIRONMENTS

Co-simulation platforms provide integrated environments where hardware register-transfer logic models execute concurrently with software, firmware, and drivers, enabling comprehensive validation of multi-protocol controllers. These platforms bridge abstraction levels by connecting hardware simulators with software execution environments, allowing realistic system-level scenarios during pre-silicon development phases. Engineers employ platforms such as SystemVerilog testbenches, Universal Verification Methodology frameworks, and commercial tools, including Synopsys VCS and Cadence Xcelium, for creating sophisticated verification environments (Gielis, J. *et al.*, 2022). Testbench architectures generate protocol transactions, monitor interface signals, check functional correctness, and measure coverage metrics, guiding verification completeness.

Publisher: SARC Publisher

Platform capabilities include transaction-level modeling, abstracting low-level signal details while maintaining functional accuracy, and accelerating simulation performance for complex scenarios. Constrained random generation produces diverse test patterns exploring state spaces more thoroughly than directed testing alone. Functional coverage tracking identifies untested controller features requiring additional scenarios (Gielis, J. *et al.,* 2022). Assertion-based verification embeds properties, monitoring invariants, and protocol compliance continuously during simulation. Debug capabilities provide waveform visualization, signal tracing, and breakpoint functionality, facilitating defect analysis across hardware-software boundaries. The integrated co-simulation environment enables validation teams to exercise multi-protocol controllers through realistic workloads, detecting integration issues, timing violations, and functional defects before silicon fabrication.

Table 4: Challenges in Multi-Protocol Controller Co-Verification (Halevi, Y., & Ray, A. 1988; González, I. *et al.*, 2017)

Challenge Category	Description
Test Environment Complexity	Managing integrated testbench architectures that combine hardware simulation models with software execution contexts requires sophisticated coordination mechanisms and debugging capabilities
Synchronization Issues	Coordinating timing between hardware state machines operating at gate-level speeds and software processes executing at system-level rates introduces verification complications
Performance Bottlenecks	Co-simulation platforms face computational overhead when executing both register-transfer logic models and firmware code concurrently, affecting validation throughput
Abstraction Layer Debugging	Identifying defects spanning hardware-software boundaries demands expertise across multiple engineering domains and specialized debugging tools for cross-layer analysis
Shared Resource Conflicts	Controllers managing multiple protocols through common registers and buffers require careful verification of resource arbitration and access sequencing across operational modes
Protocol Timing Variations	Different communication standards impose distinct signaling requirements and timing constraints that must be validated simultaneously within unified test frameworks

CONCLUSION

Integrated hardware-software validation of multi-protocol controllers produces system-on-chip designs with demonstrated reliability across communication interfaces. Testing register-transfer logic behavior together with software interaction patterns prevents misalignments that typically emerge during system integration. Controllers managing multiple protocols through shared resources present unique verification challenges that unified testing methodologies effectively address. Finding functional defects, timing violations, and interface incompatibilities during simulation prevents costly silicon revisions and shortens product deployment cycles. Testbench architectures combining hardware models with executable software reflect actual usage patterns, providing thorough validation coverage for target applications. Protocol compliance checks, register access validation, and application interface testing establish confidence before physical prototyping. Integrated circuit designs continue evolving toward greater complexity with expanded protocol support and constrained resources, making unified verification essential for guaranteeing functional correctness and maintaining interoperability between hardware and software layers. Multi-protocol controllers validated through these methods meet performance criteria while delivering reliable operation across industrial automation, automotive networking, consumer electronics, and enterprise communication platforms. Verified functional behavior and confirmed protocol adherence result from comprehensive pre-silicon testing that exercises realistic scenarios. Teams deploying these validation strategies deliver controllers capable of supporting demanding communication requirements with predictable, reliable performance characteristics across diverse application domains.

REFERENCES

1. Roth, S., Karacora, Y., Chaccour, C., Sezgin, A., & Saad, W. "Integrated Communication and Control Systems: A Data Significance Perspective." *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, (2023).

Publisher: SARC Publisher

2. Petrochenkov, M. V., Mushtakov, R. E., & Shpagilev, D. I. "Verification of system on chip integrated communication controllers." *Труды Института системного программирования РАН* 30.3 (2018): 195-206.
3. Halevi, Y., & Ray, A. "Integrated communication and control systems: Part I—Analysis." (1988): 367-373.
4. González, I., Calderón, A. J., Barragán, A. J., & Andújar, J. M. "Integration of sensors, controllers and instruments using a novel OPC architecture." *Sensors* 17.7 (2017): 1512.
5. Bayılmış, C., Ebleme, M. A., Çavuşoğlu, Ü., Küçük, K., & Sevin, A. "A survey on communication protocols and performance evaluations for Internet of Things." *Digital Communications and Networks* 8.6 (2022): 1094-1104.
6. Lian, K. Y., Hsiao, S. J., & Sung, W. T. "Intelligent multi-sensor control system based on innovative technology integration via ZigBee and Wi-Fi networks." *Journal of network and computer applications* 36.2 (2013): 756-767.
7. Li, Y., Zhang, X., Zhang, Z., Zhang, J., Wu, Y., & Guan, Y. "Design and application of modular multi-channel integrated scalable control device." *Proceedings of the 2025 5th International Conference on Control and Intelligent Robotics*. (2025).
8. Wen, D., Zhou, Y., Li, X., Shi, Y., Huang, K., & Letaief, K. B. "A survey on integrated sensing, communication, and computation." *IEEE Communications Surveys & Tutorials* (2024).
9. Gielis, J., Shankar, A., & Prorok, A. "A critical review of communications in multi-robot systems." *Current robotics reports* 3.4 (2022): 213-225.

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Abraham, J. " Co-Verification of Multi-Protocol Controllers for Integrated Communication Systems." *Sarcouncil Journal of Engineering and Computer Sciences* 4.12 (2025): pp 83-90.

Publisher: SARC Publisher