Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Scoreboard Methodologies for Ensuring Data Integrity in Modern VLSI Verification

Suri Babu Talla

NXP, USA

ARTICLEINFO

ABSTRACT

Ensuring data integrity across modern system-on-chip designs has become one of the central challenges in VLSI verification. As semiconductor systems grow to incorporate multi-core processors, heterogeneous compute units, multi-protocol interconnects, chiplet-based packaging, and AI-accelerated workloads, verification approaches built around local interface monitoring can no longer provide sufficient coverage. Scoreboards address these issues by providing a scalable, reusable, and coverage-driven mechanism for validating data flow, enforcing protocol rules, checking transaction ordering, and confirming end-to-end correctness across complex systems. A new approach for the implementation of the scoreboard mechanism in a hybrid form is presented, where reference models, transaction databases, comparator mechanisms, and adaptive matching policies are all brought into one framework. End-to-end verification, predictive models, associative matching, and temporal constraint validation are some of the techniques used in this framework to enhance the accuracy of verification under heterogeneous and unordered circumstances. Protocol-aware verification using state machines is also added in addition to other methodologies like debugging and emulation. This proposal was tested on various designs like memory subsystems and artificial intelligence accelerators.

Keywords: Data Integrity, Distributed Verification, Latency Optimization, Protocol Compliance, Scoreboards, Transaction-Level Monitoring

1. INTRODUCTION

1.1 The Rising Complexity of Modern SoCs

Contemporary SoC designs are no longer single-processor systems; they combine scalar CPUs, graphics units, tensor accelerators, audio and video codecs, and specialized memory controllers into one package, each block serving a workload profile that the others cannot efficiently handle [1]. PCIe and CXL connectivity extend the system boundary outward to plug-in cards and disaggregated memory, while chiplet integration brings multiple semiconductor dies together within a single package, effectively turning one chip into a small distributed system.

What makes verifying these designs difficult is not the behavior of any single block but the behavior that emerges when blocks interact [1]. Independent clock domains allow each block to run at its own frequency, supporting fine-grained power optimization through clock gating, but they also mean that every inter-block interface must handle data crossing a frequency boundary. Power islands add another dimension: circuitry that is powered down and later restored must re-establish the correct state without corrupting any in-flight transactions. Protocols AMBA AXI, ARM CHI, and CXL each define rules for how blocks communicate, covering channel handshakes, coherency messaging, and cross-package memory access [2]. In automotive contexts, ISO 26262 ASIL-D requirements layer on top of all of this, demanding that the design independently detect failures, reach a safe state, and recover in a verified way [1]. These checkers are valuable but limited, because a bug that spans multiple components will not be visible at any single boundary [2]. The increase in concurrency, variations in protocols, and the introduction of chiplets makes such a need even more pressing, especially in the face of out-of-order execution and interaction at several levels. It is thus important to adopt a more holistic validation process that utilizes several scoreboarding techniques in one cohesive framework.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1.2 Why Data Integrity Matters

When a data integrity bug escapes pre-silicon verification and reaches a shipped product, the consequences scale with the size of the deployment. Field failures triggered by customer workloads produce data loss and productivity disruption, and the vendor absorbs both warranty cost and the slower, harder-to-quantify cost of damaged reputation [3]. At volume, even a low failure rate across millions of units generates significant financial exposure. When a respin is necessary, the cost compounds further; where a software fix is possible, the engineering and logistics effort is still substantial. The practical pressure this creates is to find data integrity bugs before tape-out, under conditions that actually stress the design. Those conditions are harder to create than they appear. Most data integrity failures go undetected under typical verification workloads; they require specific combinations of stress, concurrency, and timing that routine tests rarely hit [4]. Memory bandwidth saturation loads both the control logic tracking outstanding transactions and the datapaths moving data, and when multiple cores hit the same memory bank during a refresh cycle, the interactions between address decoders, timing logic, and precharge circuitry become difficult to predict.

Cache coherency failures are particularly costly because they are silent: a core reads a stale value, a computation proceeds on wrong data, and the error may propagate far from its origin before anything detectable occurs [3]. Power state transitions introduce another window of vulnerability, as shifting frequencies and voltages temporarily relax timing margins. Rare state machine bugs may require billions of cycles to trigger, far beyond what most simulation runs cover [4]. Catching these requires running the design through sustained stress across billions of transactions, which is where extended verification with reference models and scoreboards becomes necessary [3]. A golden reference model provides the basis for that comparison. Written in straightforward sequential code against the architectural specification, the reference model predicts the correct output for any given input independently of the implementation. A scoreboard then compares what the design actually produced against what the reference model expected, and any difference identifies a specification violation [4]. Thus, there is a need for an approach that could link transactions among different subsystems.

1.3 Scoreboards as the Verification "Truth Oracle"

Data value checking confirms that bits arriving at a destination match what the source sent after passing through whatever transformation pipeline lies between them. Ordering checks enforce sequencing rules where the specification requires transactions to complete in a defined order. Protocol compliance checking confirms that control signals follow handshake rules at every interface. Latency tracking catches responses that arrive outside their specified time bounds. For components like ECC encoders, compression engines, and encryption blocks, the scoreboard validates the transformation itself, confirming that encoded data decodes correctly, that compressed data recovers without loss, and that encrypted data can be recovered with the correct key [4].

The level of abstraction at which a scoreboard operates depends on what it needs to catch. Transaction-level scoreboards work at protocol boundaries and treat the hardware between those boundaries as a black box; they record a request going in and check the response coming out without monitoring intermediate pipeline states [3]. This approach scales well and covers the majority of functional correctness requirements. Cycle-accurate scoreboards trade that scalability for precision, observing every clock edge, and are reserved for timing-dependent checks where a transaction-level view would miss the violation. Hybrid designs combine both, applying transaction-level checking broadly and dropping to cycle accuracy where the specification demands it [4]. Scoreboards are also structured to grow with the design hierarchy. At the IP block level, a scoreboard checks one component against its own specification. At subsystem level, it checks a cluster of integrated blocks. At a full SoC level, it checks the complete chip against top-level architectural requirements [3]. Because interface standards like AXI and CHI are reused across many designs, a scoreboard methodology built for one project largely transfers to the next, spreading the development investment across multiple programs. Reference models, kept separate from any particular implementation, can themselves be tested for correctness as the specification evolves, giving the verification program a stable and independently maintained source of truth [4].

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1.4 Contributions of This Article

A unified hybrid scoreboard architecture is introduced to support data integrity validation across heterogeneous system-on-chip environments. The architecture integrates end-to-end validation, predictive modeling, associative matching, and temporal constraint checking within a single scalable framework. An adaptive transaction correlation mechanism is defined to support out-of-order execution and multi-protocol communication scenarios, enabling consistent validation across distributed subsystems. A protocol-aware validation structure incorporating embedded state tracking is established to improve detection of sequencing violations and cross-interface inconsistencies. A comparative validation perspective is provided through representative subsystem scenarios, demonstrating improved scalability, coverage efficiency, and robustness relative to conventional isolated scoreboard implementations.

2. SCOREBOARD ARCHITECTURE

2.1 Core Components

A production-grade scoreboard is not a single checking mechanism but a collection of cooperating components, each responsible for a distinct aspect of correctness verification. Understanding what each component does and why it is structured the way it is provides the foundation for building scoreboards that hold up under the stress conditions modern SoC verification demands. The reference model sits at the input end of the scoreboard and receives stimulus transactions directly from the testbench. Its job is to process those transactions according to the architectural specification and produce the expected output that a correct hardware implementation would generate [5]. What that processing looks like depends heavily on the subsystem being verified. A memory controller reference model must track all outstanding transactions and apply whatever reordering rules the specification permits so that the set of valid response sequences can be determined. A cache hierarchy reference model must implement the coherency protocol well enough to predict, at any point, what data value a given observer should see. Cryptographic engine verification calls for a software implementation of the algorithm running at full precision, against which hardware results are compared. DSP pipeline verification requires the reference to compute in floating-point while accounting for the precision loss that fixed-point hardware introduces [11]. In each case the reference model is the specification made executable, and its accuracy directly determines how much trust can be placed in a passing scoreboard result.

Predicted and observed transactions are both held in a transaction database, which must support retrieval patterns that vary with the ordering behavior of the interface under test [12]. Strictly ordered interfaces allow a simple queue, since transactions arrive and depart in a known sequence and matching is positional. Out-of-order interfaces require associative containers, typically hash tables or balanced trees, so that any transaction can be located by identifier or address in logarithmic time regardless of arrival order. Interfaces where more than one expected transaction could legitimately match a given observed transaction require multi-map structures that return all candidates rather than the first hit. Choosing the wrong data structure at this layer degrades scoreboard performance under high-transaction-rate conditions and can mask bugs by introducing lookup failures that are mistaken for design errors.

The comparator engine works across both sides of the database, retrieving candidate expected transactions that correspond to an observed output and then checking them field by field [12]. The matching key depends on the protocol: AXI interfaces are typically matched on transaction identifier, address-based protocols on starting address and transfer length, and interfaces that carry no stable identifier may require matching on data content directly. Once a candidate set is retrieved, the comparator checks every attribute of the transaction, confirms that the response arrived within its specified latency window, and verifies that ordering relationships with other transactions have been respected. Protocol state is tracked in parallel so that handshake sequence violations can be reported in the same context as data mismatches rather than being caught separately. When a mismatch is detected, the error reporting unit is responsible for producing output that allows an engineer to diagnose the root cause without replaying the simulation from scratch [5]. A log entry that records only the mismatching field is rarely sufficient; useful error reports include the full observed transaction, the complete set of expected candidates that were considered, and the specific reason each candidate was rejected. Temporal context matters as well: the transactions immediately before and after the failing event often reveal whether the mismatch is a genuine design bug or an artifact of the monitoring logic itself. Protocol state at the moment of failure and the relationship of the failing transaction to coverage metrics

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

are both included, since knowing whether a failure occurred on a heavily exercised path or a rarely reached corner case changes the priority assigned to the investigation. Coverage collection runs alongside comparison throughout the simulation, tracking progress across several orthogonal dimensions [10]. Functional coverage confirms that each specified design feature has been exercised at least once. Scenario coverage tracks whether combinations of features have been tested together, since many bugs appear when two behaviors interact. Data pattern coverage ensures the verification has not relied on a narrow set of values, particularly where boundary conditions are known to stress implementation logic. Error injection coverage records whether the design's own detection and correction mechanisms have been exercised, not merely its normal operation paths. Ordering coverage confirms that diverse transaction sequencing patterns have been generated rather than the same canonical order being repeated. Taken together, these metrics provide the basis for a defensible closure argument, moving verification sign-off from a judgment call to a measurable outcome.

Subsystem	Reference Model Approach	Primary Validation Goal
Memory Controller	Transaction tracking with specification-defined reordering rules	Confirm valid response sequences under out-of-order completion
Cache Hierarchy	Coherency protocol implementation predicting per-observer data values	Detect stale reads and coherency violations across multiple cores
Cryptographic Engine	Software algorithm execution at full precision against known test vectors	Bit-exact output match for every input pattern
DSP Pipeline	Floating-point computation with fixed-point precision loss accounted for	Output correctness within specified numerical error bounds

Table 1: Reference Model Strategies Across Subsystem Types [5, 11]

2.2 Scoreboard Data Flow

The path a transaction takes through the scoreboard from observation to verdict follows a structure that has been refined across many generations of verification practice. Interface monitors observe signal activity on the design's ports and reconstruct the logical transaction that the signals represent, then broadcast those transactions through UVM analysis ports to any subscriber that has registered interest [12]. This broadcast model keeps monitors independent of the scoreboard implementation so that the same monitor can feed a scoreboard, a coverage collector, and a protocol checker simultaneously without any of those components needing to know about the others.

On the expected side, transactions produced by the reference model are written into the database as soon as they are generated. If an observed transaction has already arrived and is waiting in a pending queue, the database immediately attempts a match; otherwise the expected transaction sits in the database until its counterpart is observed. On the observed side, each arriving transaction triggers a database query against the expected set [12]. When a matching expected transaction is found and all fields compare correctly, both records are retired, and the result is logged as a successful match. When candidates are found but comparison fails on one or more fields, the error reporting unit is invoked with the full context of both the observed transaction and every candidate that was considered. When no candidates are present at all, the observed transaction is held in a pending queue, and the match attempt is deferred until the corresponding expected transaction arrives.

This two-sided buffering strategy is what allows the scoreboard to handle both ordered and out-of-order interfaces within the same framework. In a strictly ordered system, the expected and observed streams arrive in the same sequence, and matches resolve immediately, keeping the database shallow. In an out-of-order system either side may run ahead of the other, and the database absorbs that skew by holding unmatched records until their counterparts appear [5]. The depth to which the database must grow under stress conditions is itself a diagnostic signal: a database that grows without bound during a simulation indicates either a design bug producing unreturned transactions or a scoreboard configuration mismatch, both of which require investigation before coverage closure can be

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

claimed.Separate treatment of expected and received stream sequences may result in partial correlation in the context of multiple subsystem or protocol domains involved in a transaction.

3. SCOREBOARD TECHNIQUES FOR DATA INTEGRITY

3.1 End-to-End Scoreboarding

An end-to-end scoreboard treats the entire data path from request entry to response delivery as a single verification unit rather than checking each interface in isolation [1]. The value delivered to the requesting core must match the contents of the addressed memory location, and confirming that requires a checker with visibility into both ends of that path simultaneously. This form checks surfaces' failure modes that boundary monitors cannot reach. Pipeline hazards arise when two operations on the same data are in flight at the same time, and a read stage observes an intermediate value rather than the committed result of the preceding write [1]. Network-on-chip arbitration logic can silently drop a packet when a routing decision coincides with a particular contention pattern, and no boundary monitor on either side of the affected link will record the loss. Replay mechanisms intended to recover from errors can duplicate a transaction when the credit counter state is inconsistent, delivering the same data twice without any individual interface checker detecting the repetition.

Reordering buffers can complete transactions in an order that violates a read-after-write dependency, producing a result that is wrong but locally consistent at every interface it crosses [4]. ECC encoding logic can miscalculate a checksum in a way that passes basic sanity checks but fails to protect against the error patterns the code was designed to catch. End-to-end scoring catches all of these because the comparison is made between what entered the system and what came out, not between what entered one stage and left it. The technique is particularly well suited to memory subsystems with reordering logic, network fabrics where routing is non-deterministic, and cache hierarchies where coherency must be maintained across multiple observers, since all three involve transformations that span too many components for any local checker to validate alone [4].

3.2 Predictive Scoreboards

Not every design block has a behavioral specification that can be expressed as a set of protocol rules. Many blocks implement algorithms drawn from signal processing, cryptography, or linear algebra, and the correct output for a given input is determined by the algorithm rather than by a state machine [5]. Scoreboards for these blocks work by running the same algorithm in software and comparing hardware outputs against software results. The choice of reference model implementation strategy affects both what can be detected and how long verification takes. Transaction-accurate models capture input-output behavior without internal pipeline detail, running faster while still catching functional errors. Algorithmic models implement the mathematical specification with no architectural detail at all, offering the best simulation performance and sufficient accuracy for blocks where the specification is defined entirely in terms of input-output behavior [11]. State-based models capture protocol state machines and are best suited to control plane verification, where the state transitions themselves are what need to be checked.

3.3 Associative Matching

Out-of-order systems present a fundamental problem for scoreboards that rely on sequential matching: the first response to arrive may correspond to the last request that was issued, making FIFO-based comparison incorrect by design [19]. Associative matching addresses this by identifying corresponding request-response pairs through content rather than position. Transaction IDs offer the most reliable matching key when the interface provides them. Each request carries an ID that propagates through the design and appears on the corresponding response, allowing the scoreboard to retrieve the correct expected transaction regardless of what order responses arrive in [19]. In cases of absent or ambiguous transaction IDs, address-based matching can be used, grouping all transactions within a specified address range. Out-of-order memory systems benefit directly from this approach because multiple outstanding transactions are the normal operating condition rather than an edge case. ARM CHI fabric designs benefit because coherency messages can resolve in any order depending on snoop responses and directory state [19]. Multi-channel I/O interfaces benefit because each channel processes its transactions independently, with no ordering relationship to the others.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.4 Temporal Scoreboarding

Some correctness requirements are expressed in time rather than in data values, and a scoreboard that only compares content will miss violations of these requirements entirely [5]. Response latency bounds, quality-of-service guarantees, minimum bandwidth provisioning for specific transaction classes, and hard deadlines in real-time scheduling are all requirements that demand temporal checking alongside functional checking. Latency validation records the arrival time of each request and the arrival time of its corresponding response, computes the interval, and flags any case where that interval exceeds the architectural maximum [10]. Timeout detection is a related but distinct check: it confirms that every transaction that enters the system eventually produces a response, catching deadlock conditions where a request is accepted but forward progress halts.

Scoreboard Technique	Target Interface	Failure Class Detected
End-to-End Scoreboarding	Cache hierarchies, reordering memory subsystems	Pipeline hazards, silent packet loss, transaction duplication
Predictive Scoreboarding	FFT, AES, DEFLATE, matrix multiply accelerators	Algorithmic output deviation, precision error, encoding fault
Associative Matching	Out-of-order memory, ARM CHI fabric, multi-channel I/O	Mismatched response pairing, ID propagation failure
Temporal Scoreboarding	Real-time DSP pipelines, automotive ASIL-D systems	Latency-bound violation, deadline miss, bandwidth starvation

Table 2: Scoreboard Techniques Mapped to Interface Types and Failure Classes [5, 10, 19]

3.5 Data Integrity Validation

Several protection mechanisms embedded in hardware designs require their own dedicated validation beyond what general transaction checking provides [6]. A scoreboard validating ECC must confirm both that the encoding logic computes the correct code for each input pattern and that the decoding logic correctly identifies and repairs injected errors rather than passing corrupted data silently. The scoreboard must confirm that the checksum computed over each burst matches the value the specification requires for that input [6]. Coherency validation tracks every access to memory locations that more than one core or processing element can reach. A write issued by one core must be visible to any other core that subsequently reads the same address; receiving a cached value from before the write is a coherency failure regardless of whether the data path is otherwise functioning correctly. When a read completes, the returned value is checked against the most recent write that preceded the read in the access history, and any discrepancy is flagged as a violation [6]. Scrambling mechanisms used for security randomize data using a key, and validation must confirm two things independently: that scrambling followed by descrambling with the same key recovers the original data exactly, and that scrambling with different keys produces outputs that bear no recoverable relationship to one another, confirming that the mechanism provides the isolation it was designed to provide [6].

3.6 Constraints in Existing Scoreboard Techniques

Scoreboarding schemes, individually, exhibit robust validation capabilities within their specific contexts; however, their efficiency is limited by their application to complex SoC configurations. End-to-end scoreboards guarantee correctness throughout entire data paths, but they do not account for interim protocol states and interdependencies between events. Predictive scoreboards deliver accurate validation at algorithmic blocks, but they depend on the accuracy of the corresponding reference model and cannot be efficiently deployed for multiple protocols simultaneously. Associative matching allows correct transaction matching even when the order of completion is inconsistent, but it can create uncertainties if there are overlapping address spaces or similar data structures. Temporal scoreboards incorporate latency and timing considerations but disregard data correctness in many

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

instances. With regard to today's SoC systems with parallel processing, distributed subsystems, and chiplet-based architecture, the above approaches are used independently, restricting them from giving unified visibility to interacting components. The lack of unified validation in areas, data, time, and protocol domains leads to their inefficiency during high-throughput operations and cross-domain operations.

4. PROTOCOL-AWARE SCOREBOARDS

Protocol specifications exist because distributed hardware components need a shared contract governing when signals are valid, in what order transactions must proceed, and what constitutes an illegal sequence [3]. A scoreboard that checks only data values will pass a transaction that carries the right data but was delivered through an illegal handshake sequence. Protocol-aware scoreboards close that gap by embedding a state machine derived from the specification itself so that every transition the design makes is evaluated against what the protocol actually permits.

AMBA AXI structures read transactions through a defined progression of states. Before any data can transfer, the initiator must assert ARVALID and hold it until the target responds with ARREADY, at which point the address and control information transfer and the transaction advances to the data phase [3]. A protocol-aware scoreboard tracking this progression will catch conditions that a data-only checker ignores entirely, including data beats that appear on the read data channel before any address handshake has been completed or a transaction that stalls indefinitely in an intermediate state because one side dropped its handshake signal at the wrong moment [3].

ARM CHI operates at a higher level of complexity because it coordinates not just two endpoints but a home node, one or more requesting nodes, and any nodes currently holding a copy of the requested data [19]. The scoreboard must track all of these message flows in parallel and confirm that the home node does not declare a transaction complete before all required snoop responses have arrived, since doing so would allow two nodes to hold conflicting copies of the same cache line without either being aware of the inconsistency [19]. PCIe introduces a different class of ordering requirement, one based not on coherency but on the sequencing rules governing how different transaction types interact with one another [1]. Posted write requests must be observed at their destination before non-posted transactions to the same address range are processed, because a design that reorders these can produce a state where a read returns a value that a preceding write was supposed to have changed. The scoreboard tracks both the ordering invariants and the replay buffer state, treating a duplicate transmission of a completed transaction as a protocol violation in its own right.

5. DEBUGGING WITH SCOREBOARDS

5.1 Root-Cause Isolation

When a scoreboard flags a mismatch, the information it captures determines how quickly an engineer can find the underlying cause. A field-by-field comparison between expected and actual transactions immediately narrows the problem to a specific attribute, whether data value, burst length, timing, or protocol field, rather than leaving the engineer to determine what went wrong from a generic failure message [9]. Temporal traces across multiple interfaces add another dimension by synchronizing transaction logs so that concurrent activity on different interfaces can be examined together [12]. Many data corruption bugs only trigger when a write on one interface coincides with a read on another at a specific phase relationship, and that relationship is invisible when interfaces are examined separately. Protocol-level context captures the state of handshake signals at the exact moment a mismatch occurs, which is particularly useful for AXI interfaces where the interaction between valid and ready signals determines whether a transaction was accepted or silently stalled [9].

5.2 Common Failure Patterns

Certain categories of failure recur across different designs. Pipeline hazards occur when a write and a read target the same resource at the same time, and the read captures an intermediate rather than committed value [4]. ECC encoding faults generate checksums that appear valid under light inspection but break down against the exact error patterns the encoding scheme was designed to protect, leaving the hardware with a false sense of integrity coverage.

Dropped network-on-chip packets point to arbitration or routing logic that discarded a transaction somewhere in the fabric rather than forwarding it toward its destination, with the loss often going undetected until a downstream

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

consumer times out waiting for a response that will never arrive. Clock domain crossing faults produce transient data corruption when a signal sampled at a frequency boundary has not yet settled, and while the metastability event itself is not directly observable, the corrupted value it produces is caught by data comparison [16].

Failure Pattern	Triggering Condition	Scoreboard Detection Mechanism
Pipeline Hazard	Concurrent read and write to same resource while write is uncommitted	End-to-end data comparison across both operations
Dropped NoC Packet	Arbitration or routing logic discards in-flight transaction under contention	Entry-exit packet tracking with timeout on unmatched entries
ECC Encoding Fault	Checksum computed incorrectly for specific input patterns	Error injection followed by decoder output comparison
Clock Domain Crossing Fault	Signal sampled before settling at frequency boundary	Data value comparison exposing transient corruption at crossing point

Table 3: Recurring Failure Patterns, Triggering Conditions, and Detection Methods [4, 16]

6. SCOREBOARDS IN EMULATION AND HYBRID VERIFICATION

Emulation platforms accelerate verification by running designs on reconfigurable hardware, reaching execution speeds several orders of magnitude faster than software simulation can achieve, but that speed comes at the cost of visibility [18]. Hybrid scoreboards address this by dividing the workload: the design runs on the emulator at full speed while the reference model and comparison logic execute on a connected host workstation, with monitors on the emulator side capturing transactions and forwarding them across a high-speed interface to the host for checking [18]. This split allows firmware-driven tests to run for billions of cycles over hours rather than weeks, covering design state space that shorter simulation runs never reach. Coverage analysis across those extended runs produces a much more complete picture of which scenarios have actually been exercised, and the data integrity checks that would be impractical to sustain in simulation remain active throughout, catching correctness failures that only surface under prolonged stress conditions [18].

7. BEST PRACTICES

7.1 Keep Reference Models Simple

Reference model complexity has a direct bearing on the quality of the verification program, and the relationship runs counter to what engineers might expect. A model that tries to replicate the internal pipeline structure of the design introduces its own defects, and when a false failure fires, the team spends debug time on the model rather than the hardware [11]. Simpler models that capture the behavioral specification without mirroring implementation detail are faster to validate, easier to maintain across design revisions, and far less likely to produce misleading results. That narrower scope is both easier to reason about and quicker to execute during simulation [5].

7.2 Use Transaction IDs

In out-of-order systems, transaction IDs are the most reliable basis for matching actual outputs to expected outputs. Without them, the scoreboard must fall back on address-based or content-based matching, both of which introduce ambiguity when multiple outstanding transactions share similar attributes, and that ambiguity produces false failures that erode confidence in the verification results [11]. Assigning explicit IDs at the stimulus generator and propagating them through the design gives the comparator an unambiguous key, regardless of how the design reorders responses internally.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

7.3 Build Scalable Data Structures

Scoreboard performance degrades sharply when the underlying data structures are not chosen with transaction volume in mind. High-bandwidth designs can issue thousands of transactions per clock cycle, and a scoreboard that uses sequential search through a queue will become the bottleneck in simulation long before the design itself is stressed [12]. Associative containers like hash tables and balanced search trees keep lookup time constant or logarithmic regardless of how many transactions are outstanding, which means scoreboard overhead stays proportionally small even as the transaction count grows into the tens of thousands. Selecting the right container at architecture time avoids having to restructure the scoreboard later under schedule pressure.

7.4 Support Error Injection

A scoreboard that can only validate correct behavior covers only part of the verification requirement. Robustness validation depends on confirming that the design detects and recovers from faults, and the scoreboard must be capable of setting the correct expectation when an error is deliberately introduced [15]. Single-bit ECC errors should produce a corrected output with an error flag; the scoreboard should expect both. Packet drops, timeouts, and parity violations each have defined responses in the architectural specification, and verifying that those responses occur requires the scoreboard to anticipate them rather than treat them as unexpected failures.

7.5 Integrate Coverage

Scoreboard activity is a natural source of coverage data, and connecting the two gives the verification program a more complete picture of what has actually been tested [10]. Functional coverage confirms that individual design features have been exercised. Data-pattern coverage checks that the design has been tested against diverse input values rather than a narrow range of boundary cases. Error-injection coverage records which fault scenarios have been validated through the mechanisms described in the previous section. Taken together, these dimensions give the team a defensible basis for declaring verification closure rather than relying on simulation time alone as a proxy for thoroughness.

8. CASE STUDIES

8.1 PCIe Gen5 Root Complex

A PCIe Gen5 root complex managing traffic across multiple endpoint devices and bridges must process transaction layer packets at gigahertz rates while preserving ordering, flow control, and error detection guarantees across every protocol layer [1]. CRC integrity checks run across all protocol layers, confirming that the error detection mechanisms built into the standard are functioning as the specification requires. In addition to integrity verification, scoreboard verification is needed for monitoring the transaction replay characteristics caused by retry features at the data link layer.

The retry feature could cause transactions to be retransmitted, leading to duplication and resending of transaction completion, particularly in cases where multiple devices attempt to acquire simultaneous access to the common resource. Credit flow control verifies that both the transmission buffer and receive buffers will not overflow and, at the same time, checks whether credit return procedures maintain consistency throughout the layered protocols. The scoreboard also checks the accuracy of the reconstruction process in case of multiple lanes, where skew compensation and deskew buffer processes will ensure packets are reconstructed in order at the receiver.

8.2 Automotive NoC Fabric

In automotive NoC architectures, verification requirements extend beyond functional correctness to include strict guarantees on determinism, safety, and timing predictability. Quality-of-service validation checks that priority levels produce the expected latency separation between traffic classes, confirming that high-priority requests are not delayed behind lower-priority ones. Guaranteed bandwidth validation confirms that minimum allocation commitments are maintained under load, and arbitration analysis checks for starvation conditions where lower-priority traffic is indefinitely blocked, a scenario that can produce priority inversion with system-level consequences in a safety-critical context. In addition to this, scoreboard validation makes sure that packet-level consistency is upheld while traversing the routing paths even when congestion and arbitration are present.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Silent drops of packets, erroneous routing of the transactions, and time inconsistencies, which occur only in the case of the worst traffic conditions, are identified using this approach. As the automotive applications are usually bound by ISO 26262 specifications, the scoreboard also validates fault containment properties of the system to make sure that faults are not propagated across the local subsystem. This combination of structural, temporal, and safety-oriented validation is essential in NoC fabrics where correctness cannot be evaluated at a single interface boundary but must be inferred across multiple interacting nodes.

8.3 AI Accelerator Pipeline

In AI accelerator pipelines, correctness is defined not only by functional accuracy but also by numerical stability across deep computational graphs. Quantization validation checks that the fixed-point conversion from floating-point input handles rounding modes and saturation boundaries according to the specification rather than truncating in an implementation-defined way. Verification via scoreboards, in this case, involves element-by-element comparison of matrix calculations' outputs against software models and guarantees that precision losses are not exceeding acceptable threshold levels. It is especially vital for deep pipeline processing where intermediate calculation values are used across several processing cycles, thus making minor numerical differences grow and accumulate. In addition to this, scoreboard checking is essential during mixed-precision computation, wherein different types of formats such as FP32, FP16, and INT8 are used by various layers during their computation. This helps in determining whether the format conversion process is semantically equivalent to the values generated. In particular, it is relevant for inference tasks where the output accuracy might not be zero but would still affect task performance measures like accuracy and loss convergence.

Case Study	Core Scoreboard Responsibility	Primary Verification Coverage
PCIe Gen5 Root Complex	Flow control credit tracking and multi-layer CRC integrity checking	Lane reversal reassembly, replay buffer duplicate detection
Automotive NoC Fabric	Entry-exit packet matching with zero tolerance for silent loss	QoS latency separation, starvation and priority inversion detection
AI Accelerator Pipeline	Element-by-element matrix result comparison against software reference	Quantization rounding modes, saturation boundary handling

Table 4: Case Study Scoreboard Responsibilities and Coverage Priorities [1, 5, 10]

9. PROPOSED HYBRID SCOREBOARD ARCHITECTURE

Machine learning models trained on known-good design behavior will increasingly work alongside rule-based checkers, catching output deviations that fixed assertions would not flag because the failure mode was never anticipated when the testbench was written [9]. Chiplet integration is moving from a niche packaging strategy to a mainstream design approach, and scoreboards will need to follow, extending correctness checks across die-to-die interfaces and treating the full multi-die assembly as the verification unit rather than stopping at the boundary of a single die. Formal and simulation-based methods will be applied in combination more deliberately than they are today, with mathematical proofs covering complete execution spaces and simulation providing empirical confirmation across representative workloads [14]. Generative AI tools will contribute stimulus generation and coverage-driven assertions built from learned behavioral models, supplementing what engineers produce manually rather than replacing the underlying verification judgment those engineers apply [9].

9.1 Architecture Overview

The architecture is built up of four major layers that include transaction acquisition, prediction and modeling, correlation and matching, and validation and reporting blocks. The transaction acquisition blocks are interfaced to the design monitors to get the expected and observed transactions. The prediction and modeling modules calculate

the expected output values by applying corresponding reference models to architectural descriptions. The correlation and matching blocks match the observed and predicted transactions through adaptive criteria.

9.2 Integrated Validation Mechanisms

Predictive validation aids in the development of algorithms through the creation of reference outputs using software reference models. Associative matching provides for proper association of transactions during out-of-order executions through identifiers, addresses, and other matching processes based on contents. Latency, timeout, and quality-of-service controls can be enforced through temporal validation.

9.3 Adaptive Transaction Correlation

Transaction correlation is performed using a hierarchical matching strategy that prioritizes transaction identifiers when available, followed by address-based grouping and content-based matching as secondary criteria. This adaptive mechanism reduces ambiguity in high-concurrency environments where multiple transactions share overlapping attributes. The dynamic buffers accommodate both anticipated and actual transactions, facilitating synchronization despite the different patterns.

9.4 Protocol-Aware Validation Integration

Protocol compliance is incorporated through embedded state tracking mechanisms aligned with protocol specifications AXI, CHI, and PCIe. State transitions can be checked in tandem with transactions, thus making it possible to detect any issues relating to sequence ordering, incomplete handshaking, and invalid state progression. The combination allows for the extension of verification from just verifying data integrity to verifying control plane activity.

9.5 Scalability and Multi-Domain Support

The architecture supports scalability across increasing transaction volumes and subsystem complexity through the use of associative data structures and modular validation components. Distributed validation nodes can be deployed across subsystems, with centralized coordination maintaining global consistency. Multi-domain support includes processing of asynchronous clock domains, power domain transitions, and interconnects within chiplets. In addition to this, the architecture ensures consistency through transaction integrity regardless of any differences in time taken during processing and changes in protocol behaviors.

CONCLUSION

The verification demands placed on modern SoC designs have grown in proportion to their architectural ambition. Heterogeneous compute integration, multi-protocol interconnects, chiplet disaggregation, and AI workloads have collectively raised the bar for what functional verification must cover, and scoreboards have moved from a useful addition to a foundational requirement for data integrity assurance across these systems. Their value comes from the ability to observe behavior across deeply parallel execution environments and protocol-rich interfaces simultaneously, catching corruption, duplication, ordering violations, and timing anomalies that local checkers positioned at individual boundaries cannot see. Beyond detection, the scalability of the scoreboard model means the same architectural approach applies at the IP level, subsystem level, full SoC integration, and emulation regression without fundamental redesign. Reference models maintained independently of any specific implementation provide verification teams a stable correctness baseline that survives across tape-outs and specification revisions.

REFERENCES

[1] Jacob Ryan Maas et al., "End-to-End Formal Verification Strategies for IP Verification" https://dvcon-proceedings.org/wp-content/uploads/end-to-end-formal-verification-strategies-for-ip-verification.pdf

[2] Lubis-Eda, "Formal Verification vs. Simulation—Data ordering and integrity," August 12, 2024, https://lubis-eda.com/formal-scoreboarding-for-verifying-data-ordering-and-integrity/

[3] Gordon Allan, "Keeping Score: Architectural Considerations for Scoreboard Design." https://dvcon-proceedings.org/wp-content/uploads/keeping-score-part-1-of-2-architectural-considerations-for-scoreboard-design.pdf

[4] Dr. Mike Bartley, Transitioning to Advanced Verification Techniques for FPGAs https://www.aldec.com/en/company/blog/87--transitioning-to-advanced-verification-techniques-for-fpgas-catch-22

[5] Xinbing Zhou and Yi Man, "ASIP performance enhancement by hazard control through scoreboard," Micromachines (Basel), vol. 15, no. 11, p. 1287, Oct. 2024. https://pmc.ncbi.nlm.nih.gov/articles/PMC11596897/

[6] Vijaypal Singh Rathor, Soykot Podder, and Shivam Dubey, "An ensemble learning model for hardware Trojan detection in integrated circuit design," Comput. Electr. Eng., vol. 123, no. PB, p. 110090, Apr. 2025. https://dl.acm.org/doi/10.1016/j.compeleceng.2025.110090

[7] Pietro Nannipieri et al., "VLSI design of advanced-features AES cryptoprocessor in the framework of the European Processor Initiative," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 30, no. 1, pp. 1–10, Dec. 2021. https://www.researchgate.net/publication/356698082

[8] Iwona Grobelna, "Formal verification of control modules in cyber-physical systems," Sensors, vol. 20, no. 18, p. 5154, Sep. 2020. https://www.mdpi.com/1424-8220/20/18/5154

[9] Xiong Luo and Manman Yuan, "Machine-learning-assisted intelligent processing and optimization of complex systems," MDPI, Aug. 2023. https://mdpi-res.com/bookfiles/book/8167/MachineLearningAssisted_Intelligent_Processing_and_Optimization_of_Complex_Systems.pdf

[10] Romain Jacob, "Challenges and recent advances in the design of real-time wireless cyber-physical systems," BenchCouncil Trans. Benchmarks, Stand. Eval., vol. 2, no. 1, p. 100036, Mar. 2022. https://www.sciencedirect.com/science/article/pii/S2772485922000230

[11] Pranuti Pamula et al., "Verification of SoC using advanced verification methodology," Eng. Proc., vol. 34, no. 1, p. 12, Mar. 2023. https://www.mdpi.com/2673-4591/34/1/12

[12] Valentin Radu et al., "Generative AI assertions in UVM-based SystemVerilog functional verification," Systems, vol. 12, no. 10, p. 390, Sep. 2024. https://www.mdpi.com/2079-8954/12/10/390

[13] Hyun Woo Oh and Seung Eun Lee, "The design of optimized RISC processor for edge artificial intelligence based on custom instruction set extension," IEEE Access, vol. 11, May 2023.

[14] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10124773

[15] Shadi Banitaan et al., "Foundation models in software engineering: A taxonomy, systematic review, and in-depth analysis of testing support," Information, vol. 17, no. 1, p. 73, Jan. 2026. https://www.mdpi.com/2078-2489/17/1/73

[16] Mehrdad Moradi, Bert Van Acker, and Joachim Denil, "Failure identification using model-implemented fault injection with domain knowledge-guided reinforcement learning," Sensors, vol. 23, no. 4, p. 2166, Feb. 2023. https://www.mdpi.com/1424-8220/23/4/2166

[17] Mariusz Węgrzyn, Orest Kochan, and Ihor Maikiv, "Fault injection tool for FPGA reliability testing: A novel method and discovery of LUT-specific logical redundancies," Electronics, vol. 14, no. 23, p. 4600, Nov. 2025. https://www.mdpi.com/2079-9292/14/23/4600

[18] Heni Belgacem, Mohammad Abuabiah, and Ines Chihï, "Fault diagnosis framework for mechatronics systems using digital model and machine learning," Procedia Comput. Sci., vol. 277, pp. 1009–1018, 2026. https://www.sciencedirect.com/science/article/pii/S1877050926002577

[19] [18] Luigi Arena et al., "A co-simulation approach to the validation of communication protocols for smart shires environments," Simul. Modelling Pract. Theory, vol. 148, p. 103261, Apr. 2026. https://www.sciencedirect.com/science/article/pii/S1569190X26000109

[20] [19] Yiqiang Zhao et al., "Research on cache coherence protocol verification method based on model checking," Electronics, vol. 12, no. 16, p. 3420, Aug. 2023. https://www.mdpi.com/2079-9292/12/16/3420