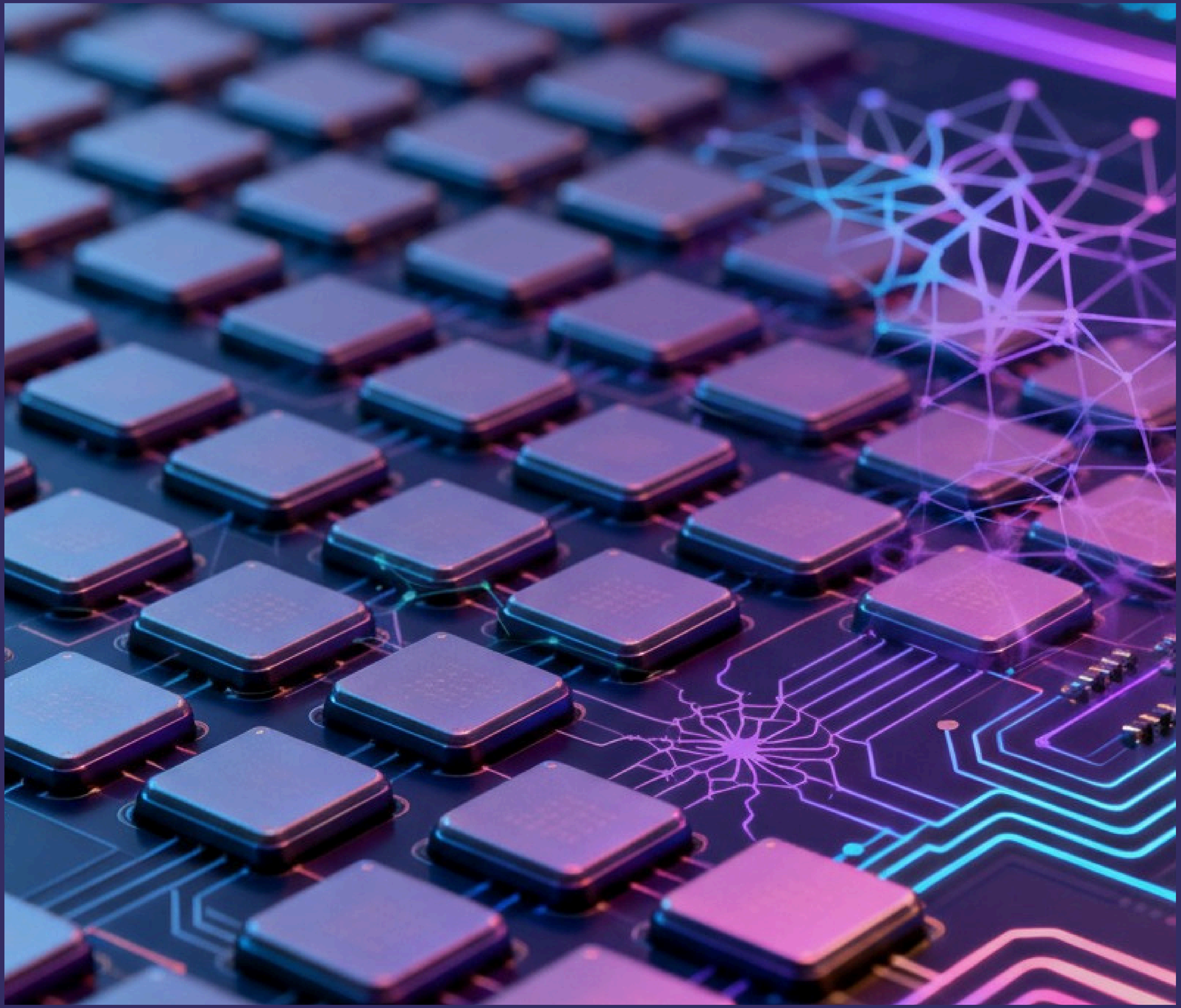
VERIFICATION CHALLENGES AND STRATEGIES FOR RISC-V BASED NEAR-MEMORY AI ACCELERATORS

SUBMISSION CATEGORY: ASIC / SoC Verification, RISC-V Verification

AUTHOR NAME: Vinayak V P

UNIVERSITY NAME: Vidyavardhaka College of Engineering, Mysuru



EXECUTIVE SUMMARY

Near-memory AI acceleration changes how computation and memory interact in modern SoCs. While it reduces data movement and power, it introduces complex verification challenges due to custom RISC-V instructions and parallel execution.

01 **Near-memory AI accelerators integrated with RISC-V processors reduce data movement but introduce complex verification challenges.**

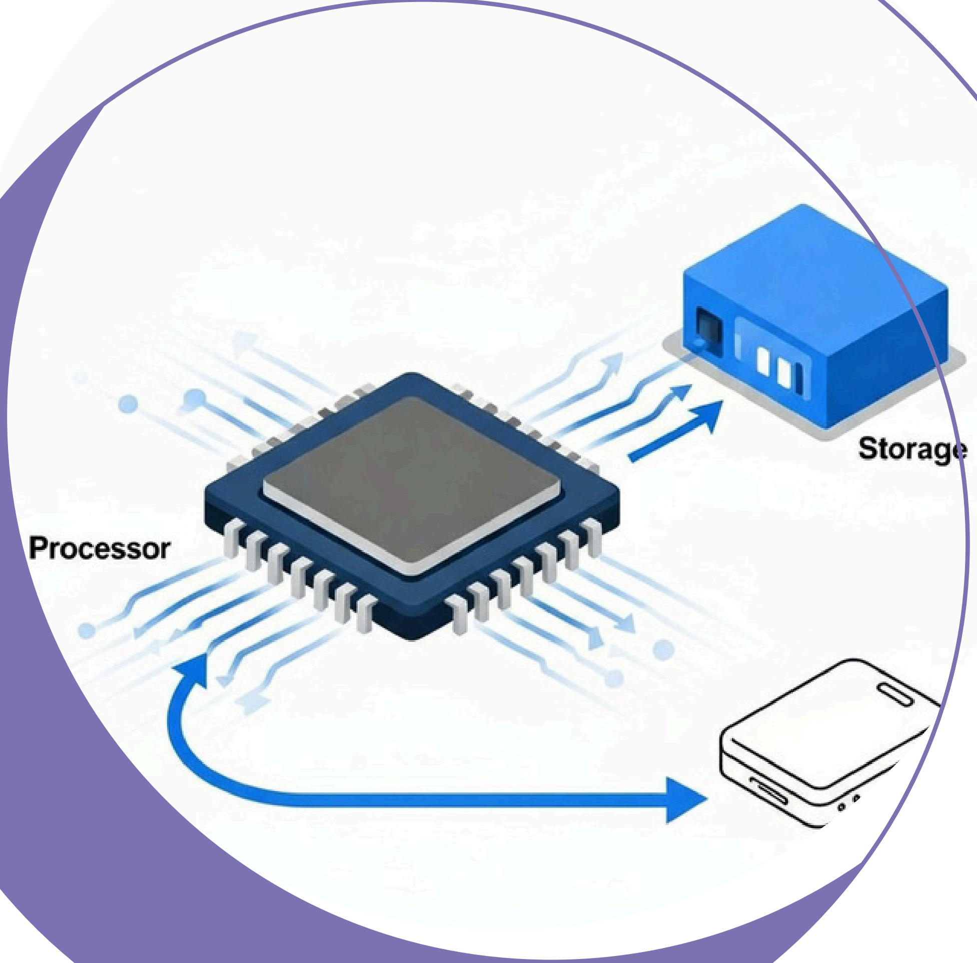
02 **Custom RISC-V instructions, tightly coupled memory-compute behavior, and parallel execution make traditional verification insufficient.**

03 **This work analyzes key verification challenges and proposes layered verification strategies for near-memory AI architectures.**

04 **The approach emphasizes functional correctness, memory consistency, and coverage-driven verification for scalable AI SoCs.**

Problem statement

VERIFICATION CHALLENGES IN NEAR-MEMORY RISC-V AI ARCHITECTURES



- Custom RISC-V extensions lack mature reference models
- Near-memory computation violates conventional memory visibility and ordering assumptions
- Parallel dataflows introduce race conditions and corner cases
- Functional coverage closure is challenging in parallel AI accelerators

LEVEL 3 VERIFICATION CHALLENGES AND SOLUTIONS

01

- **Challenge: Lack of Golden Reference Models for Near-Memory Compute**

Solution:

- Use hierarchical reference models instead of a single golden model
- Instruction-level C/Python models for custom RISC-V instructions
- Block-level functional models for near-memory MAC units

02

- **Challenge: Memory Visibility and Ordering Violations**

Solution:

- Assertion-based verification for ordering and visibility rules
- Stress tests combining load/store operations with near-memory compute

LEVEL 3 VERIFICATION CHALLENGES AND SOLUTIONS

03

- **Challenge: Parallel Execution and Race Conditions**

Solution:

- Constrained-random stimulus targeting parallel execution
- Synchronization and completion assertions

04

- **Challenge: Functional Coverage Closure Difficulty**

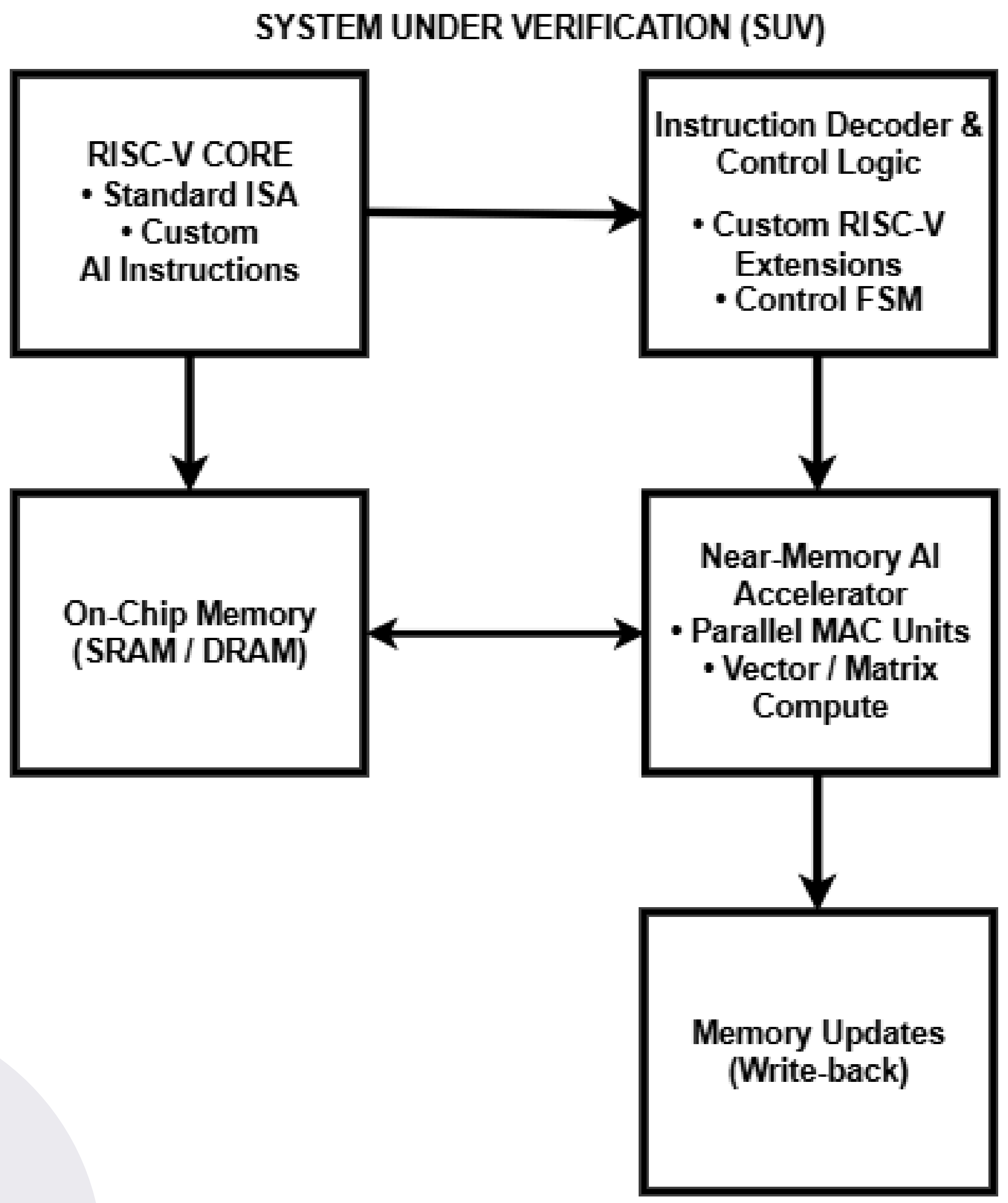
Solution:

- Coverage-driven verification targeting parallel dataflow corner cases

Verification Challenges:

WHY NEAR-MEMORY RISC-V ARCHITECTURES ARE HARD TO VERIFY

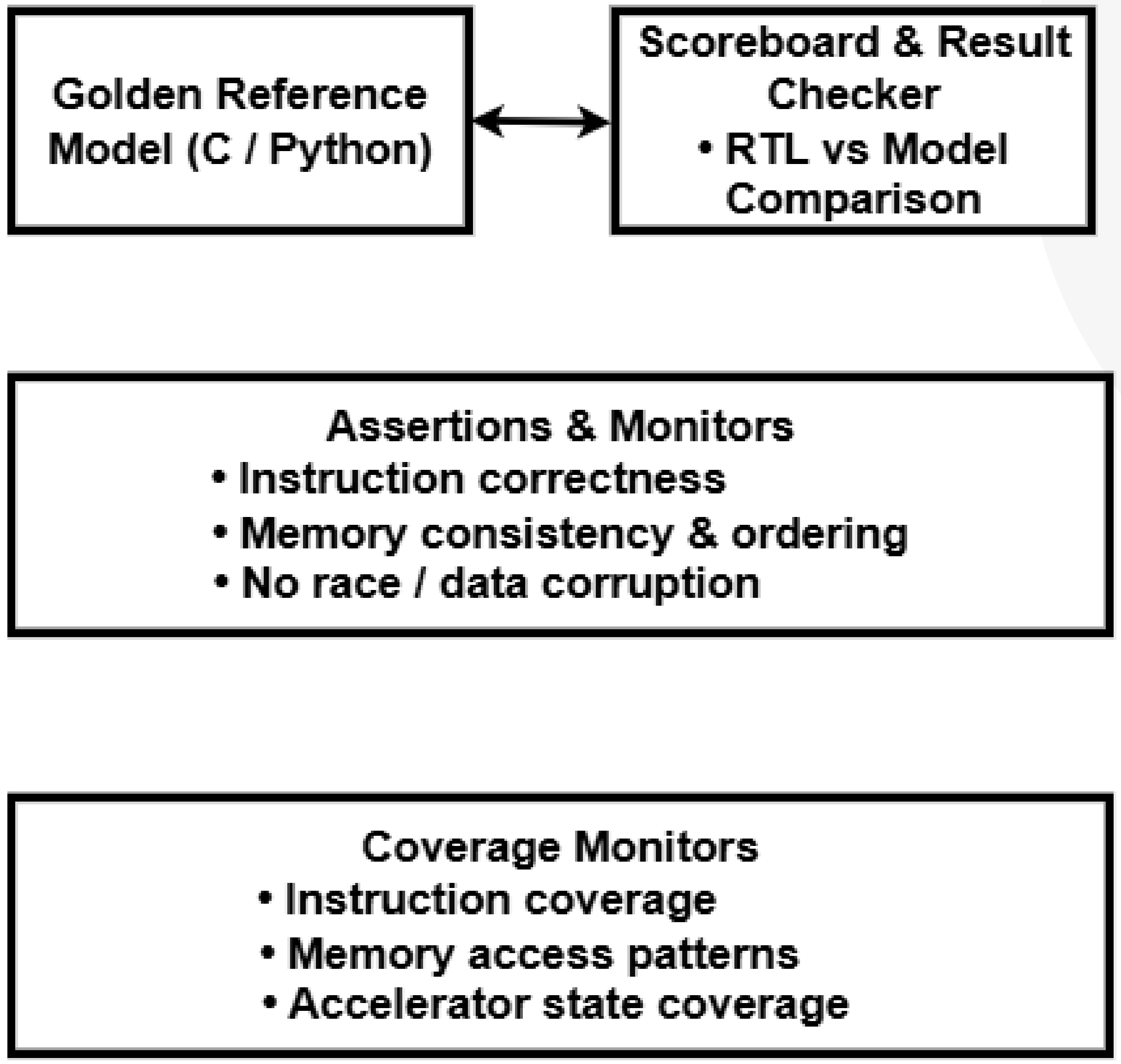
This diagram illustrates the tight interaction between the processor and memory in near-memory architectures.
While computation moves closer to data, it also reduces observability and breaks traditional assumptions, making verification more challenging compared to conventional SoCs.


SYSTEM UNDER VERIFICATION (SUV)
RISC-V CORE
• Standard ISA
• Custom AI Instructions
Instruction Decoder & Control Logic
• Custom RISC-V Extensions
• Control FSM
On-Chip Memory (SRAM / DRAM)
Near-Memory AI Accelerator
• Parallel MAC Units
• Vector / Matrix Compute
Memory Updates (Write-back)

Verification Challenges and Solutions

VERIFICATION VIEW OF A RISC-V BASED NEAR-MEMORY AI ACCELERATOR

This diagram presents a verification-centric view of the system under verification, consisting of a RISC-V core, custom instruction control, near-memory AI accelerator, and memory.
Verification is performed using reference models, assertions, and coverage monitors to validate instruction correctness, memory consistency, and parallel execution behavior.


Golden Reference Model (C / Python)
Scoreboard & Result Checker
• RTL vs Model Comparison
Assertions & Monitors
• Instruction correctness
• Memory consistency & ordering
• No race / data corruption
Coverage Monitors
• Instruction coverage
• Memory access patterns
• Accelerator state coverage

VERIFICATION-RELEVANT ARCHITECTURAL FEATURES

- Custom RISC-V instructions require instruction-level functional verification
- Near-memory computation impacts memory visibility and ordering assumptions
- Parallel execution introduces synchronization and race-condition risks
- Modular RTL enables isolated block-level and integration verification

Custom RISC-V Instruction Extensions

Near-Memory Compute & Data Visibility

Parallel MAC Execution

Modular RTL Partitioning

Clock Gating & Pipeline Stages

CHALLENGE–SOLUTION–RESULT: GOLDEN REFERENCE MODEL GAP

Challenge:

Lack of a complete architectural golden reference model for near-memory AI compute.

Solution Applied:

- Hierarchical reference modeling instead of a single golden model
- Instruction-level C/Python reference models for custom RISC-V instructions
- Block-level functional models for near-memory MAC operations
- End-to-end memory consistency checks

Results Achieved:

- Enabled functional verification without a full golden model
- Instruction decode and execution mismatches identified
- Memory consistency corner cases exposed

RESULTS: IMPACT OF VERIFICATION STRATEGIES ON LEVEL-3 CHALLENGES

01 **Golden Reference Model Challenge:**
Hierarchical reference modeling revealed instruction-level mismatches

02 **Memory Visibility & Ordering Challenge:**
Assertion-based checks exposed ordering and synchronization corner cases

03 **Parallel Execution Challenge:**
Functional coverage improved for parallel dataflow scenarios

04 **System-Level Observability Challenge:**
C-based SoC tests improved debug observability across compute–memory boundaries

TECHNICAL APPROACH

Verification Methodology

01 Instruction-Level Verification:

- Validate custom RISC-V instruction decode and execution
- Use directed tests and reference model comparison

02 Memory Subsystem Verification:

- Verify data consistency, visibility, and ordering
- Check correct interaction between compute and memory

03 Block-Level Verification:

- Verify functional correctness of systolic MAC units
- Validate parallel execution and synchronization behavior

04 SoC-Level Integration Verification:

- Validate interactions between RISC-V core, accelerator, and memory
- Verify control, data flow, and system-level correctness

Technical viability

- Availability of RISC-V Verification Ecosystem
- Reusable Reference Models for AI Operations
- Assertion-Based Verification for Memory Behavior

Verification Feasibility & Practicality

- Compatibility with UVM-Based Verification Flows
- Scalable Verification from Block to SoC Level
- Relevance to Emerging Edge-AI SoC Designs

Residual Challenges and Future Work

- ✓ Availability of open-source RISC-V cores and verification environments
- ✓ Existing RTL and functional reference models for MAC and vector operations
- ✓ FPGA and SoC platforms supporting accelerator prototyping and testing
- ✓ Mature EDA toolchains for simulation, assertions, and coverage

Scaling hierarchical reference models to larger near-memory systems

Managing state-space explosion in highly parallel AI workloads

Improving functional coverage metrics for complex AI dataflows

Debug scalability across tightly coupled compute–memory interfaces

REFERENCES:

1. A. Waterman, Y. Lee, D. Patterson, and K. Asanović, “The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA,” RISC-V International, 2019.
2. D. Kroening and O. Strichman, “Decision Procedures: An Algorithmic Point of View,”Springer, 2016.
3. J. Bergeron, “Writing Testbenches: Functional Verification of HDL Models,” Springer, 2nd Edition, 2003.
4. H. Foster, A. Krolnik, and D. Lacey, “Assertion-Based Design for System-on-Chip Verification,” Springer, 2nd Edition, 2004

APPLICATIONS

- Verification of AI accelerators in edge SoCs
- RISC-V based memory-centric architectures
- Power-efficient accelerators for low-cost devices
- Validation of near-memory compute in SSD/DRAM controllers