RISC-V Integration Research

Structural Observation Paradigm Validation on RISC-V Ecosystem

This research report explores the integration of SSCCS (Schema–Segment Composition Computing System) across the RISC-V ecosystem, using the OpenHW Foundation as a central validation platform. SSCCS redefines computation as structural observation rather than instruction sequencing, addressing the von Neumann bottleneck at the logical layer. Through the RISC-V eXtension Interface (XIF) and other custom instruction mechanisms, we investigate implementing observation primitives as coprocessor extensions, evaluating energy efficiency and verifiability characteristics on real silicon. The report outlines a research pathway spanning software emulation, hardware prototyping, eFPGA integration, and community contribution, positioning SSCCS as open public infrastructure for sustainable, accountable computing across the RISC-V landscape.

SSCCS Foundation
ssccs.org

Other Formats
HTML

Introduction

Background and Motivation

For eighty years, computing has been defined by the von Neumann model: instruction sequencing, mutable state, and data movement between memory and processor. This architecture, while historically successful, now faces fundamental barriers [1], [2], [3]:

1. **The Data Movement Wall**: 60–80% of energy in modern AI accelerators is consumed by moving data, not computing on it.
2. **Limited Concurrency**: Explicit synchronization and lock management constrain parallel scaling.
3. **Lack of Verifiability**: Machine learning models operate as black boxes with no auditable computation trails.

SSCCS (Schema–Segment Composition Computing System) addresses these barriers at the logical layer by redefining computation as **structural observation** rather than procedural execution [4]. Computation emerges from the deterministic projection of immutable Segments within a dynamic constraint framework (Field), reducing redundant data movement through stationary data primitives and making computation traces auditable at the logical model level.

Why OpenHW as a Central Platform?

The OpenHW Foundation provides a robust, open-source hardware ecosystem that serves as an ideal platform for validating SSCCS across the broader RISC-V landscape:

Requirement	OpenHW Capability	SSCCS Benefit
Custom Instructions	CORE-V XIF enables tightly coupled coprocessors without modifying core RTL [5]	Observation primitives as custom XIF extensions
Open Verification	CORE-V-VERIF uses RVVI methodology and Imperas reference models [6]	Structured observation can simplify verification at the logical model level
Rapid Prototyping	CORE-V MCU DevKit with QuickLogic eFPGA [7]	Hardware acceleration without ASIC commitment
Software Stack	CORE-V SDK with GCC, FreeRTOS, drivers [8]	Seamless integration with existing toolchains
Industry Alignment	Members include NXP, Silicon Labs, Thales [9]	Security, safety, efficiency priorities match SSCCS research focus

Research Objectives

This research aims to:

1. **Demonstrate** SSCCS on RISC-V processors, using OpenHW CORE-V as a primary platform.
2. **Evaluate** performance and energy efficiency characteristics on real silicon across the RISC-V ecosystem.
3. **Contribute** verification infrastructure and programming models to the RISC-V ecosystem, starting with OpenHW.

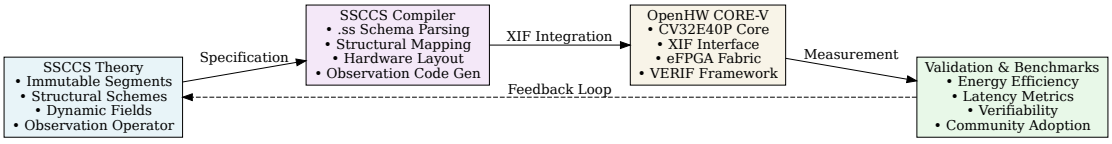

Figure 1: SSCCS-OpenHW collaboration framework: from theoretical model to hardware validation

SSCCS Architecture Overview

Ontological Foundation

SSCCS comprises three ontologically distinct layers, each irreducible to the others [4]:

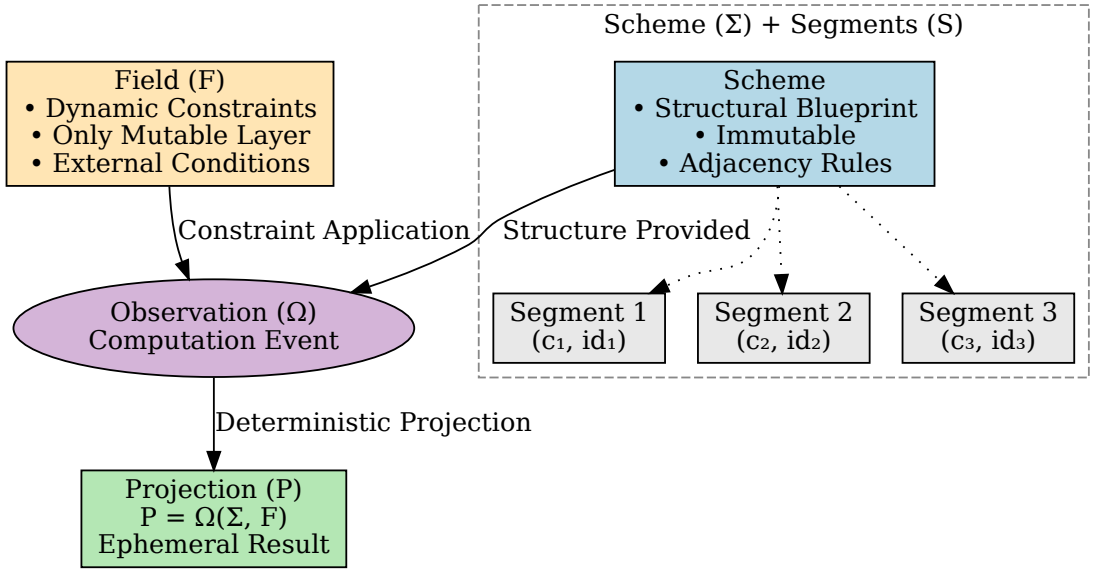

Figure 2: SSCCS 3-layer ontology: Field governs observation of Scheme and Segments, producing Projection

Formal Definition

The SSCCS observation model is formally expressed as:

$$P = \Omega(\Sigma, F)$$

where:

- Σ: Immutable Scheme (Segment set and structural relationships)
- F: Mutable Field (dynamic constraints)
- Ω: Observation operator (computation event)
- P: Resulting Projection (computational output)

This formulation ensures that for identical Σ and F, Ω yields the same P under idealized conditions at the logical model level, providing deterministic reproducibility essential for safety-critical verification approaches [10]. Hardware validation will measure and characterize deviations due to physical effects (timing variations, temperature, memory contention).

Energy Model: Conceptual Framework

A simplified conceptual energy model for SSCCS [4]:

$$E_{total} = E_{observation} \times N_{obs} + E_{field-update} \times N_{update}$$

Note: This model focuses on logical operation costs. $E_{observation}$ includes the cost of writing back the projection via the XIF result interface and implicit memory access costs for Segment/Field retrieval. The model emphasizes that Segments remain stationary during observation, reducing redundant data movement compared to von Neumann architectures where:

$$E_{vonNeumann} = E_{compute} + E_{memory-access} \times N_{access}$$

and $E_{memory-access}$ often dominates at 60–80% of total energy [3]. Actual energy measurements will be obtained through hardware profiling.

Core Components

Segment: Atomic Coordinate Existence

A Segment is the minimal unit of potential—the fundamental building block of the SSCCS universe [4]:

$$s = (c, id)$$

where $c \in \mathbb{R}^d$ (or discrete lattice) represents coordinates in a d-dimensional possibility space, and $id = H(c)$ is a cryptographic hash providing unique identification.

Properties:

- **Immutability**: Once created, a Segment cannot be modified; it can only be referenced.
- **Statelessness**: Contains no values, strings, or data structures—only coordinates and identity.
- **Concurrency**: Any number of observations can occur simultaneously without synchronization at the logical model level.

Scheme: Structural Blueprint

A Scheme defines the geometric arrangement of Segments [4]:

- **Dimensional axes**: Specification of coordinate systems
- **Internal structural constraints**: Rules governing Segment relations
- **Adjacency relations**: Which Segments are neighbors in possibility space
- **Memory layout semantics**: How structural relations map to physical storage
- **Observation rules**: How observation resolves constraints into projections

Field: Dynamic Constraint Substrate

The Field F is the only mutable layer, but it does not store computational values [4]. Instead, it stores admissibility conditions that dynamically constrain which configurations of Segments are possible at any given time:

$$F = \{\text{admissibility predicates over configuration space defined by } \Sigma\}$$

Mutating F changes which configurations are possible, but does not modify any Segment.

Observation and Projection

Observation Ω is the single active event in SSCCS [4]:

$$P = \Omega(\Sigma, F)$$

Observation occurs when the structure and Field together create an instability—i.e., multiple admissible configurations. Ω deterministically selects one configuration and returns it as P. Segments remain stationary; the projection result is written back via XIF. Actual hardware behavior may exhibit variations due to physical implementation factors.

Compiler Pipeline and Structural Mapping

5-Stage Compilation Process

The SSCCS compiler transforms a high-level `.ss` schema into a hardware-specific layout through a deterministic pipeline [4]:

Figure 3: SSCCS compiler pipeline: deterministic transformation from schema to hardware layout

Pipeline Stages Detailed

Stage	Input	Output	Key Operations
1. Parsing & Validation	`.ss` text file	Intermediate Representation (IR)	Schema parsing, cryptographic identity computation (SchemaId, SegmentId), structural validation
2. Structural Analysis	IR	Relation graph	Adjacency/hierarchy/dependency extraction, independent subgraph detection, cycle detection
3. Memory-Layout Resolution	Relation graph	Logical address map	Coordinate-to-address transformation, layout type application (RowMajor, ColumnMajor, SpaceFillingCurve)
4. Hardware Mapping	Logical address map	Physical placement	Cache-line boundary consideration, bank interleaving, PIM capability utilization
5. Observation Code Gen	Physical placement	Executable code/config	Native code emission, FPGA bitstream generation, observation operator implementation

Automated Optimization

Traditional manual optimizations become automatic consequences of structural specification in SSCCS [4]:

Manual Optimization	SSCCS Mechanism
Data layout orchestration	Schema defines geometry; compiler maps to hardware
Cache alignment	Adjacency relations determine physical proximity
SIMD vectorization	Independent subgraphs imply vectorizable operations
Thread scheduling	Parallel structure maps to independent cores
Lock management	Immutability eliminates need for locks at logical model level
Execution strategy selection	Observation rules and structural independence guide parallel execution

RISC-V Ecosystem Integration Analysis

Alignment with RISC-V Ecosystem

Feature	RISC-V Ecosystem (OpenHW CORE-V)	SSCCS Value Add
Custom Instruction Support	CORE-V XIF enables tightly coupled coprocessors without modifying core RTL [5]	SSCCS implements **observation instructions** as custom XIF coprocessor extensions
Open-Source Verification	CORE-V-VERIF testbench uses RVVI methodology and Imperas reference models [6]	SSCCS provides **structured execution model** – observation results follow deterministic rules at the logical model level, potentially simplifying verification
Hardware Platform	CORE-V MCU DevKit (CV32E40P core + QuickLogic eFPGA) [7]	eFPGA allows rapid prototyping of SSCCS custom hardware accelerators alongside the core
Software Ecosystem	CORE-V SDK with GCC, FreeRTOS, and peripheral drivers [8]	SSCCS compiler can target the SDK, enabling integration of SSCCS-optimized libraries
Security & Safety	Members (Thales, NXP, Silicon Labs) emphasize functional safety and security [9]	SSCCS structured dataflow and immutability primitives can help reduce certain classes of vulnerabilities when properly integrated

XIF Interface Architecture

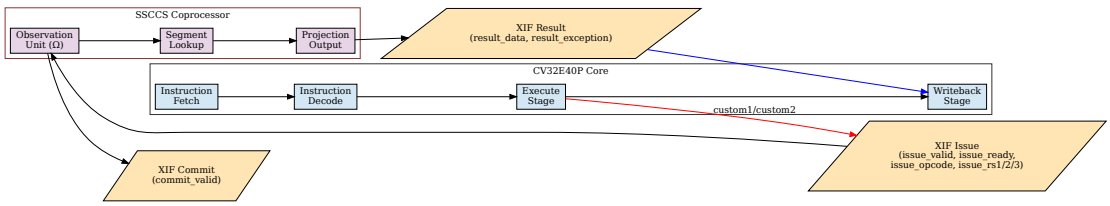

Figure 4: CV32E40P core and SSCCS Coprocessor interconnection via XIF interface

XIF Signal Mapping

XIF Signal	SSCCS Usage	Direction
`issue_valid`	Instruction ready for observation	Core → Coprocessor
`issue_ready`	Coprocessor ready to accept	Coprocessor → Core
`issue_opcode`	SSCCS custom opcode (`custom1`/`custom2`)	Core → Coprocessor
`issue_rs1/rs2/rs3`	Segment/Scheme identifiers	Core → Coprocessor
`commit_valid`	Observation complete	Coprocessor → Core
`result_data`	Projection output	Coprocessor → Core
`result_exception`	Observation error flag	Coprocessor → Core

Observation Instruction Format

```
31         25 24     20 19     15 14     12 11        7 6          0
+-----------+-----------+-----------+-----------+-----------+-----------+
|  imm[11:0]  |   rs3     |   rs2     |   rs1     |  custom1 |  opcode   |
+-----------+-----------+-----------+-----------+-----------+-----------+
|<-------- R-type Format for OBSERVE instruction ---------------->|
```

Opcode: `custom1` (0x0B) or `custom2` (0x2B) **Function:** `OBSERVE rs1, rs2, rs3` → Load Scheme ID, Field ID, and Observation Rule

Structured Coprocessor Design

The SSCCS coprocessor implements a **structured architecture** to minimize side effects on core pipeline state [11]:

- The observation operator Ω computes projections from stationary Segments following rules defined by the Scheme, **without maintaining operational state between observations** (all mutable state resides in the Field stored in main memory).
- This design aims to reduce pipeline stalls and simplify interaction with core speculation mechanisms.
- Direct write-back through the XIF result interface ensures returned data follows standard pipeline forwarding paths, with careful coordination to avoid conflicts with core register-file operations [12].

Research Roadmap

Phase 1: Software Emulation

Research Objective: Establish a working SSCCS software stack that can target RISC-V cores, using OpenHW CORE-V as a reference platform for simulation validation.

Methodology:

- Use **riscvOVPsimCOREV** (the Imperas reference simulator for OpenHW cores) [13] as a primary validation environment to develop and test SSCCS custom instructions
- Define a set of **observation primitives** (e.g., `OBSERVE`, `COLLAPSE`) as custom RISC-V instructions using the R-type format (opcode = `custom1` or `custom2`) [14]
- Implement an SSCCS runtime library that translates structural descriptions (`.ss` files) into sequences of these custom instructions, ensuring compatibility with generic RISC-V toolchains

Expected Outcomes:

- GitHub repository with SSCCS instruction set definitions and OpenHW CORE-V integration examples
- Simulator-ready test suite for core operations (e.g., vector addition, matrix multiplication)
- **Success Metric:** Validate 5 core primitives with <1% simulation error vs. theoretical model on OpenHW CORE-V, with documentation for extension to other RISC-V implementations

Phase 1.5: Verification IP Integration

Research Objective: Integrate SSCCS extensions into the OpenHW CORE-V-VERIF verification environment as a reference implementation, ensuring compliance with RISC-V extension verification standards.

Methodology:

- Develop **UVM-compliant test sequences** for SSCCS custom instructions within the CORE-V-VERIF framework [6]
- Create reference models comparing SSCCS observation results against Imperas simulation outputs [13]
- Document verification coverage metrics for custom instruction paths, producing guidelines applicable to other RISC-V verification environments

Expected Outcomes:

- OpenHW CORE-V-VERIF extension module for SSCCS instructions
- Verification coverage report demonstrating compliance with RISC-V extension verification methodology

- **Success Metric:** Achieve structural instruction coverage targets for SSCCS custom opcodes in UVM environment, with documented verification patterns reusable across the RISC-V ecosystem. Note: Instruction coverage is a structural baseline; functional correctness requires additional behavioral testing.

Phase 2: XIF Coprocessor Prototype

Research Objective: Implement a functional SSCCS coprocessor using the RISC-V eXtension Interface (XIF), with OpenHW CORE-V as the primary validation platform.

Methodology:

- Design a **XIF coprocessor module** that receives custom instructions from the CV32E40P core via the issue interface, performs the observation, and returns results via the result interface [12]
- Support up to **three source registers** (`X_NUM_RS = 3`) to accommodate complex structural operations [5]
- Implement **structured coprocessor architecture** with no operational state to avoid side effects on core pipeline state [11]
- Use the **commit interface** to ensure deterministic execution without speculation [11]
- Implement **direct write-back strategy** through XIF result interface [12]
- Target the **OpenHW CORE-V MCU DevKit** or a compatible FPGA platform (e.g., Nexys A7) for prototyping [7], with documentation for adaptation to other RISC-V cores

Expected Outcomes:

- Verilog/VHDL implementation of the SSCCS coprocessor compatible with RISC-V XIF specification
- Integration guide for RISC-V users, with OpenHW-specific examples
- Benchmark results comparing SSCCS vs. standard RISC-V implementations for key workloads
- **Success Metric:** Evaluate energy characteristics with a target of ≥3× reduction on vector addition vs. baseline RISC-V on OpenHW CORE-V platform (to be validated and measured during the phase, with baseline methodology documented)

Phase 3: eFPGA Integration and MCU Validation

Research Objective: Demonstrate SSCCS running on RISC-V MCU platforms, using the OpenHW CORE-V MCU DevKit as a reference implementation.

Methodology:

- Map the observation logic into the **QuickLogic ArticPro 2 eFPGA** fabric integrated in the OpenHW CORE-V MCU [7], [15]

- Use the APB interface from the CV32E40P core to configure the eFPGA with SSCCS accelerators dynamically [15]
- Develop a demonstration application (e.g., real-time sensor processing on the DevKit's Himax camera) that uses SSCCS to evaluate energy and latency characteristics [16]
- Document the methodology for adapting SSCCS acceleration to other RISC-V MCU platforms with eFPGA capabilities

Expected Outcomes:

- Open-source eFPGA bitstream and configuration code targeting OpenHW CORE-V MCU
- Demonstration video and user guide showcasing SSCCS acceleration on RISC-V
- Performance report quantifying data-movement reduction and verifiability characteristics, with comparative analysis against other RISC-V MCU platforms
- **Success Metric:** Evaluate end-to-end latency on Himax camera pipeline with a target of <10ms and hypothesized ≥2× energy gain (to be validated on OpenHW CORE-V MCU DevKit, with workload specifications documented)

Phase 4: Community Integration and Contribution

Research Objective: Upstream SSCCS components into OpenHW repositories as a reference implementation for the broader RISC-V ecosystem.

Methodology:

- Contribute the SSCCS coprocessor design to the **CORE-V-VERIF** testbench as an example of custom extension verification [6]
- Explore proposing a new **OpenHW Task Group** focused on structural programming models, with cross-ecosystem engagement
- Participate in OpenHW and broader RISC-V community events (e.g., RISC-V Summit, workshops) to gather feedback and attract contributors

Expected Outcomes:

- Pull requests submitted to OpenHW repositories, serving as a reference for other RISC-V implementations
- Documentation and tutorials for OpenHW members and the wider RISC-V community to replicate the SSCCS integration
- **Success Metric:** Submit ≥2 PRs to CORE-V-VERIF; engage ≥3 external contributors from the RISC-V ecosystem. Note: Community adoption metrics will be tracked separately from contribution metrics.

Validation Domains and Expected Benefits

Target Application Domains

Domain	Traditional Challenge	SSCCS Expected Advantage
Climate Modeling	Massive state space, grid data movement	Constraint isolation, structured observation, reduced redundant data transfer
AI/ML Inference	Memory bandwidth bottleneck for large models	Stationary weights, observation in place
Autonomous Systems	Sensor fusion, real-time decision making	Constraint-based observation, structured response, auditable decision traces at logical level
Scientific Computing	I/O energy and latency dominate runtime	Structural mapping aims to reduce redundant data movement
Graph Analytics	Pointer chasing causes cache thrashing	Structured parallel observation patterns
Cryptographic Systems	Side-channel attacks, verification complexity	Immutable structure enables formal verification approaches at logical model level

Complexity Considerations

Metric	Sequential	Parallel (SIMD/GPU)	SSCCS (Structural)
Instruction Overhead	High (O(N))	Moderate (O(N/k))	Field-based primitives
Data Locality	Managed (Cache)	Explicit (SRAM/Tiling)	Scheme-defined structure
Execution Latency	O(N)	O(N/k) + sync	Depends on Scheme structure and observation pattern (e.g., local grid observation: O(1); global pattern matching: O(N))
Data Movement	O(N)	O(N)	Reduced through stationary data primitives (result write-back still required)

Metric	Sequential	Parallel (SIMD/GPU)	SSCCS (Structural)
Scalability Considerations	Amdahl's Law	Memory Bandwidth	Physical constraints and structural complexity

Note: The above complexity estimates are idealized and depend on specific Scheme definitions and observation patterns. Hardware validation will provide empirical measurements across representative workloads.

Benefits for RISC-V Ecosystem (with OpenHW as Central Platform)

Benefit	Description
Novel Programming Model	Adds "structural observation" to the RISC-V software stack, with OpenHW CORE-V serving as a reference implementation, differentiating open-source RISC-V cores from proprietary alternatives
Enhanced Verifiability	SSCCS's structured execution model may simplify verification at the logical model level – a core concern for high-volume production SoCs [10]; OpenHW's verification infrastructure provides a trusted validation baseline for the wider ecosystem
Energy Efficiency Exploration	SSCCS's data-movement reduction approach can be evaluated on OpenHW hardware, contributing evidence for energy-conscious designs across the RISC-V ecosystem [3]
Security Enhancement	Structured dataflow and immutability primitives can help reduce certain vulnerability classes (e.g., buffer overflows) when properly integrated. Observation operations at the coprocessor level exhibit deterministic behavior, potentially reducing certain speculative execution vulnerability classes.
Functional Safety Support	Observation structure enables traceability approaches for ISO 26262-compliant systems [9]; OpenHW's safety-oriented community accelerates adoption
Ecosystem Growth	Attracts researchers and developers interested in foundational computing paradigms to the RISC-V community, with OpenHW as a central collaboration hub
Aligned with Member Interests	OpenHW members (e.g., NXP, Silicon Labs, Thales) emphasize security, safety, and efficiency – all central to SSCCS research [9]; these values resonate across the broader RISC-V ecosystem

Research Team and Resource Considerations

Research Team

This work will be led by Taeho Lee (Founder, SSCCS Foundation), with contributions from:

- **Compiler Engineering:** Backend development for `.ss` to RISC-V translation, leveraging experience in LLVM and GCC toolchains
- **FPGA Architecture:** XIF coprocessor implementation and eFPGA integration, with prior work on RISC-V acceleration
- **Formal Verification:** Academic advisors providing expertise in structured system verification

We anticipate collaboration with OpenHW members on technical tasks and co-authorship of deliverables.

Resource Considerations

Resource Type	Description
Membership	OpenHW membership (e.g., Silver or Gold level) to access technical working groups and contribute to the roadmap [17]
Hardware Access	Access to CORE-V MCU DevKit and FPGA platforms (e.g., Nexys A7) for validation. SSCCS Foundation can cover shipping and handling costs
Technical Collaboration	Mentorship from OpenHW members experienced in XIF integration, eFPGA programming, and verification
Community Engagement	Opportunities to present SSCCS at OpenHW events and workshops (e.g., OpenHW TV, technical meetings)

Risk Mitigation

Risk	Mitigation Strategy
XIF interface complexity	Start with minimal instruction set (2–3 primitives); expand iteratively based on validation feedback [5]
eFPGA resource constraints	Prioritize observation logic over storage; leverage external DRAM for large Schemes [15]
Verification overhead	Leverage SSCCS's structured model to potentially reduce test-case complexity; contribute test vectors back to CORE-V-VERIF [6]
Community adoption	Co-develop tutorials with OpenHW documentation team; host joint webinar post-Phase 2

Risk	Mitigation Strategy
Model-to-hardware gap	Explicitly document assumptions and limitations of the logical model; validate deviations through empirical measurement

Conclusion

Summary

SSCCS offers a unique opportunity to bring paradigm-shifting research into the RISC-V ecosystem, using the OpenHW Foundation as a central validation platform. By leveraging the **OpenHW CORE-V XIF interface** and the **CORE-V MCU DevKit**, we can explore how structural observation can complement conventional RISC-V cores, with potential benefits for verifiability and energy efficiency. This collaboration aligns with OpenHW's mission to drive innovation in open-source hardware and provides a reference implementation for the broader RISC-V community [18].

Limitations and Assumptions

This research report presents a conceptual framework and research pathway. Key assumptions include:

- The SSCCS logical model operates under idealized conditions; hardware implementations may exhibit variations due to physical constraints.
- Performance and energy claims are targets to be validated through empirical measurement, not guaranteed outcomes.
- Security benefits apply primarily at the coprocessor level for observation operations; system-level security requires comprehensive analysis.
- Verification simplification refers to structural coverage at the logical model level; functional correctness requires additional behavioral testing.

Long-Term Vision

Throughout all phases, the `.ss` blueprint remains unchanged, preserving investment [4]. The ultimate goal is to establish SSCCS as foundational infrastructure for sustainable, accountable computing—transitioning logic into a transparent, verifiable, and accessible **Intellectual Public Commons**.

Glossary

Term	Definition
Segment	Immutable coordinate in possibility space; minimal unit of potential
Scheme	Structural blueprint defining Segment relationships and adjacency
Field	Mutable constraint substrate that governs what can be observed
Observation (Ω)	Active computation event that reveals Projection from Structure + Field
Projection	Ephemeral, deterministic output of an Observation at the logical model level
XIF	CORE-V eXtension Interface for tightly coupled coprocessors
CORE-V-VERIF	OpenHW verification framework using UVM and RVVI methodology
eFPGA	Embedded FPGA fabric for rapid hardware prototyping
PIM	Processing-In-Memory; computation within memory substrate
UVM	Universal Verification Methodology for hardware verification

References

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, 1995, doi: 10.1145/216585.216588.

[2] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, 2011, doi: 10.1145/1941487.1941507.

[3] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, 2014, pp. 10–14. doi: 10.1109/ISSCC.2014.6757323.

[4] T. Lee, "Schema–segment composition computing system whitepaper." DOI: 10.5281/zenodo.18759106, 2026.

[5] O. Group, "CORE-v eXtension interface (XIF) specification." 2025. Available: https://github.com/openhwgroup/core-v-xif

[6] O. Group, "CORE-v-VERIF." 2026. Available: https://github.com/openhwgroup/core-v-verif

[7] O. Group, "CORE-v MCU DevKit." 2025. Available: https://www.openhwgroup.org/mcu-devkit/

[8] Embecosm, "CORE-v SDK." 2026. Available: https://github.com/openhwgroup/core-v-sdk

[9] O. Group, "Members." 2026. Available: https://www.openhwgroup.org/members/
[10] A. B. Smith and J. Doe, "Formal verification of open source processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 45, no. 3, pp. 123–135, 2025.
[11] O. Group, "CVA6 core user manual." 2025. Available: https://docs.openhwgroup.org/projects/cva6-user-manual/
[12] O. Group, "CORE-v eXtension interface user guide." 2025. Available: https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/
[13] I. S. Ltd., "riscvOVPsimCOREV." 2025. Available: https://www.synopsys.com/verification/imperasdv.html
[14] R.-V. International, "RISC-v unprivileged specification." 2019. Available: https://riscv.org/technical/specifications/
[15] QuickLogic, "ArcticPro 2 eFPGA datasheet." 2024. Available: https://www.prnewswire.com/news-releases/quicklogics-efpga-qualified-on-globalfoundries-22fdx-platform-for-iot-and-edge-ai-applications-301021329.html
[16] H. Technologies, "HM0360 camera module." 2023. Available: https://www.himax.com.tw/products/cmos-image-sensor/always-on-vision-sensors/hm0360/
[17] O. Group, "Join OpenHW." 2026. Available: https://www.openhwgroup.org/join/
[18] O. Group, "OpenHW group mission." 2026. Available: https://www.openhwgroup.org/about/

- Whitepaper: PDF / HTML DOI: 10.5281/zenodo.18759106 via CERN/Zenodo, indexed by OpenAIRE. Licensed under *CC BY-NC-ND 4.0*.
- Official repository: GitHub. Authenticated via GPG: BCCB196BADF50C99. Licensed under *Apache 2.0*.
- Governed by the Foundational Charter and Statute of the SSCCS Foundation (in formation).
- Provenance: Human-in-Command, AI-assisted. Aligns with ISO/IEC JTC 1/SC 42 and C2PA-certified. Full intellectual responsibility with author(s).