Supercharge your RISC-V Designs with Higher Abstraction Shift-Left

Saswat Mund, Ayush Mewati
CircuitSutra Technologies

CIRCUIT SUTRA TECHNOLOGIES
Modelling Silicon Dreams

accellera SYSTEMS INITIATIVE™

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Supercharge your RISC-V Designs with Higher Abstraction Shift-Left

Effectively use any open-source ISS in plug & play fashion in the SystemC or any other simulation / hybrid environment.

Spike, VeerISS (Chips Alliance), Whisper (Tenstorrent),
MPACT (Google), QEMU, TEMU etc..

Use multiple ISS as reference for the fool proof verification of your RISC-V Design

Agenda

- Higher Abstraction Shift-Left
- Virtual Prototype – Digital Twins
- RISC-V & Higher Abstraction Challenges
- CST-ISS: Overview, Features, Benefits
- Live Demo: RISC-V (CST-)ISS in Action
- Success Story & Benefits
- CST-ISS Future RoadMap for RISCV
- QnA

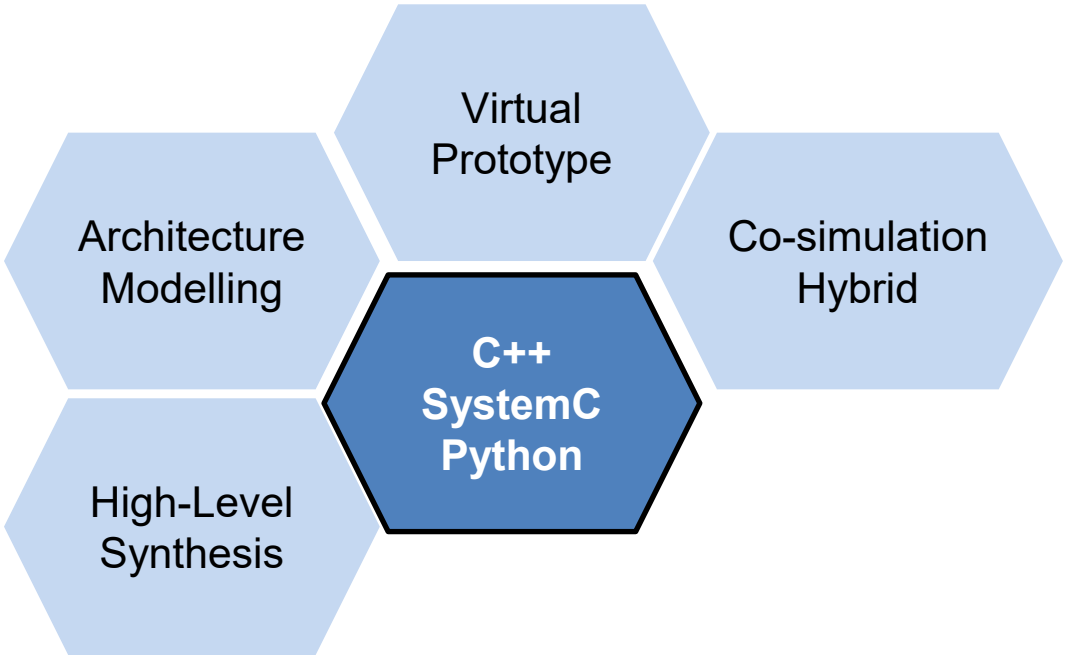
Higher Abstraction Shift Left Methodologies

Fast simulation models of SoC, IP, CPU & System at an abstraction higher than RTL using high level languages – SystemC, C++

Increasing Chip Complexity requires New Design Methodologies

Traditional RTL-GDS flow no longer sufficient

Several Design activities being performed at abstraction level above RTL



Enables

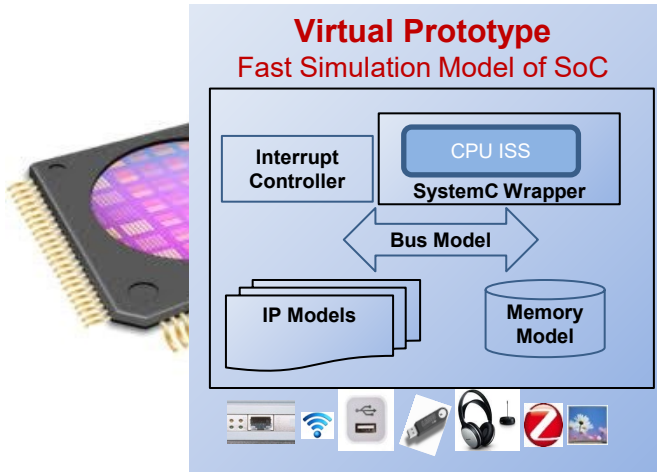
- Pre-Silicon Firmware Development & Testing
- Architecture Analysis
- Power/Performance Optimization HW-SW Partitioning
- Raising Design Abstraction above RTL - HLS
- SoC & System Level Simulation

Benefits of Higher Abstraction Shift-Left

- **Identification & Resolution of Flaws Early in the cycle**
  Architecture Flaws, Functional Specs, Functionality
- **Early Software Development (in parallel to chip design)**
- **Optimize the SoC Architecture (Performance / Power ..) for target application**
- **Faster design iterations -> More Innovation**
- **Reduce Cost. Reduce Time to Market**
- **Early Engagement with potential customers**
- **Easy to roll out future variants of the SoC**

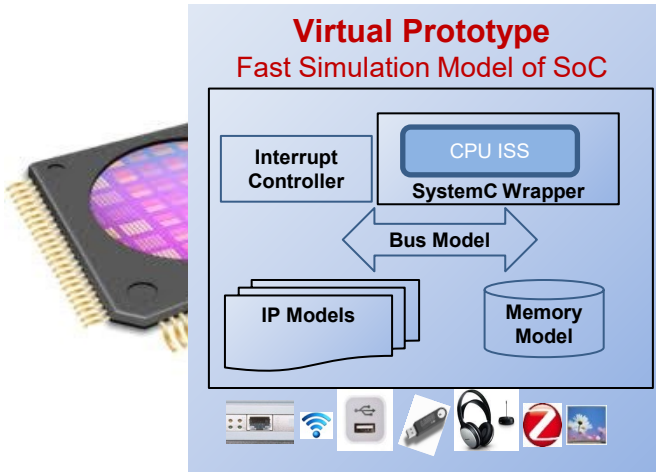
Significantly Enhance the Probability of Success

Virtual Prototype



- Building a functional digital twin before physical hardware
- Fast Executable model of SoC
- Runs un-modified software binary
- Integrates with standard debug tools
- Improved control & debug visibility
- Supports fault injection

Virtual Prototype



Model of CPU

- Instruction Set Simulator (ISS)
- Developed using C / C++
- Wrapper is needed for integrating to SystemC / Other simulation env
- Should support tracing, debugger integration, Semi-hosting etc..

Model of other blocks of SoC

IP Blocks, Interconnects, Memory, External interfaces etc..

- Register accurate, Loosely Timed fast simulation models
- Developed using SystemC, TLM2.0, Custom TLM implementations
- Hardware level debug & analysis features are desirable

RISC-V Instruction Set Architecture (ISA)

Open & free: Unlike proprietary ISAs, RISC-V is open to all.

Standardized foundation: Ensures interoperability across vendors.

Extensible & modular: Tailor instructions for specific domains (AI, IoT, HPC).

Growing global ecosystem: Academia, startups, and industry leaders driving adoption.

Robust, High Quality: Modelling Infrastructure for RISC-V is the need of the hour

RISC-V Instruction Set Simulator (ISS) – Fast Simulation Model

Several Open-Source ISS are available: Spike, VeerISS (Chips Alliance), Whisper (Tenstorrent), MPACT (Google), QEMU, TEMU etc..

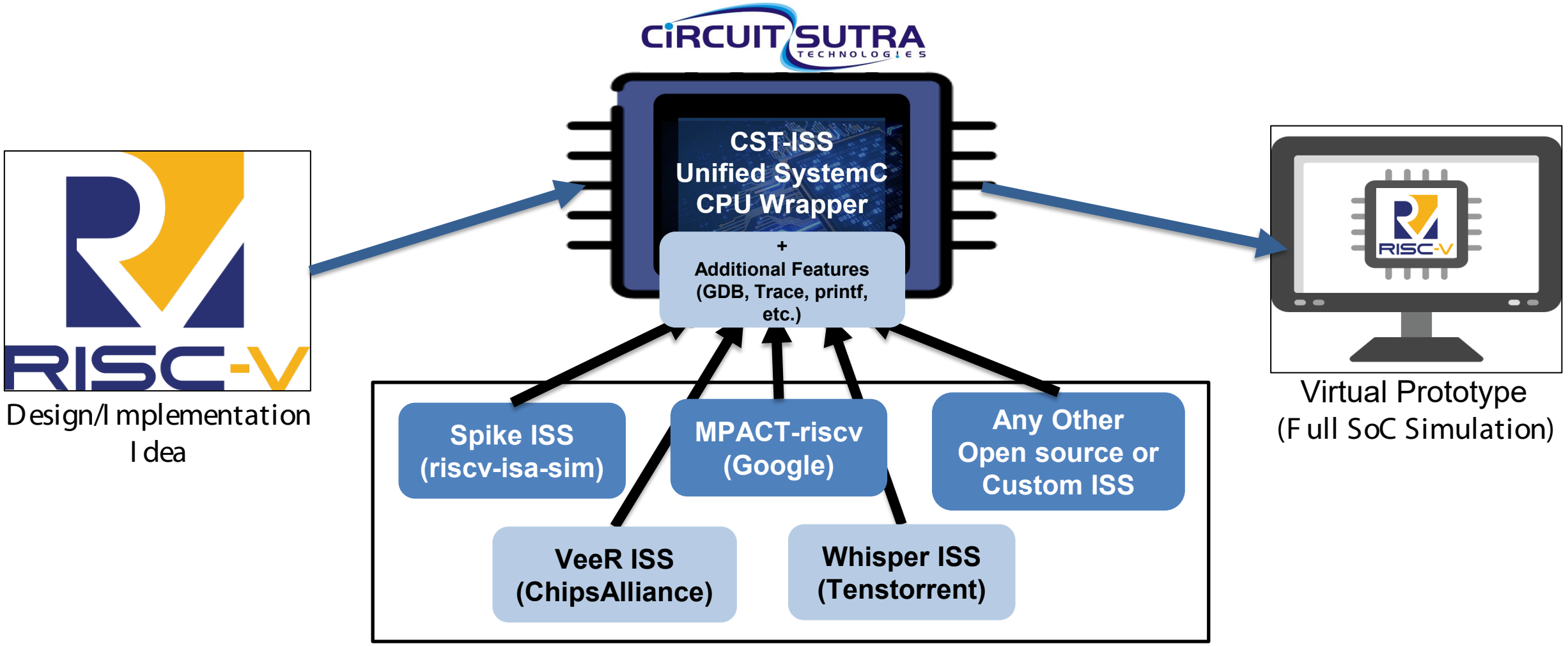
Challenges

Each of these have some pros / cons in terms of feature availability, quality, performance etc..

Difficult to use these in the SystemC and other simulation environment

Difficult to integrate with debug / analysis tools

The “Unified” Vision



Introduction to CST-ISS

What is CST-ISS

- CircuitSutra's proprietary SystemC CPU wrapper interface
- Can be integrated with any ISS (ARM, RISCV, etc.) developed using C/C++.
- Result after integration is a SystemC based functional CPU model

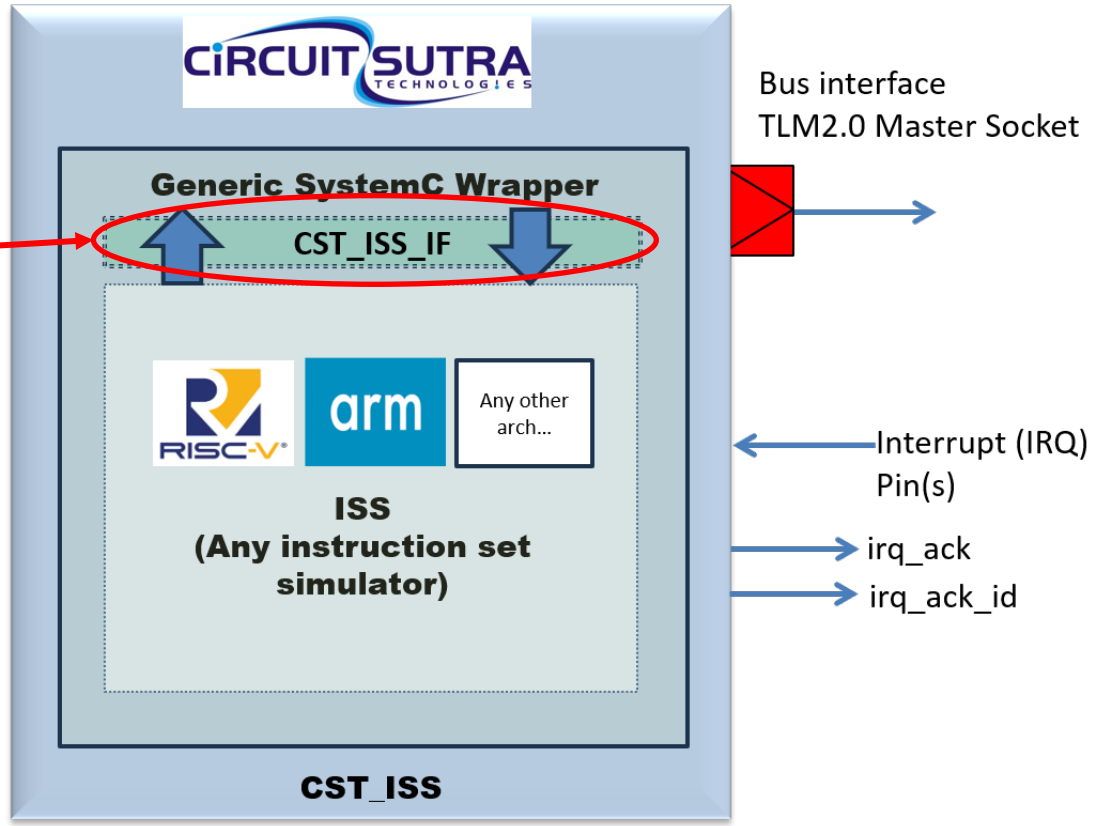
Application areas

- Virtual Prototyping
- Architectural Exploration
- FW development and testing
- Reference model for RTL Verification
- …..

CST-ISS: Architecture overview

CST_ISS_IF:

The interface that enables communication between the SystemC wrapper and the ISS.
It also enables utilities like GDB, Trace, etc.

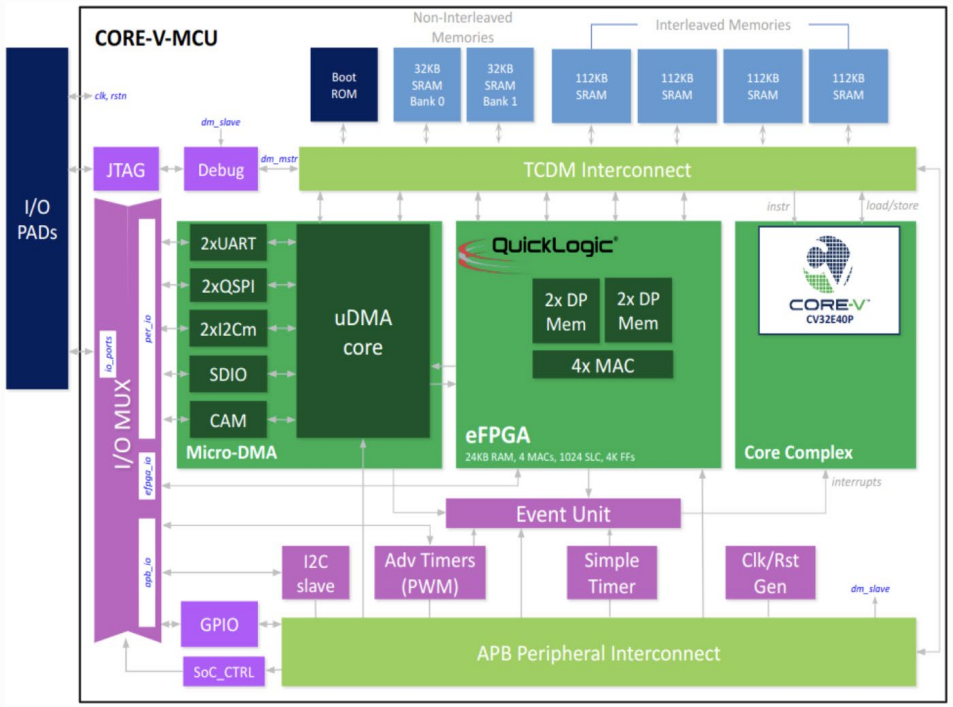


TLM2.0 Master socket to access the IP models/memory in the SystemC world.

Interrupt (IRQ) pin(s) to receive interrupts from the SystemC models

Interrupt Ack and Ack ID pins to acknowledge the IRQs, send the id of the acknowledged IRQ, respectively.

Case Study: Core-V MCU VP



CORE-V-MCU Block Diagram

Register Accurate, Loosely timed model

Developed using CircuitSutra Modelling Methodology (CSTML, VP-QSP)

- Primary Use Case:
  - Pre-Silicon FirmWare Development
  - CI/CD flow for firmware
- Abstraction
  - Timing Abstraction: Removed the clocks
  - Interface Abstraction: Transactions instead of pins
    - TLM2.0 for memory & register access
    - Custom TLM for: UART, QSPI, I2C, SDIO, etc..
  - Functionality Abstraction: Implemented only what is visible to software
- CPU Model: ***CST-ISS***
- eFPGA: Implemented as a stub model

OPENHW GROUP
PROVEN PROCESSOR IP

Fixed Virtual Platform to be available for free download: CircuitSutra website / OpenHW website
For customizable or editable Virtual Platform, please contact CircuitSutra.

CST-ISS Features

- Use any ISS in plug & play fashion
- Running simulation in 32 / 64-bit mode
- GDB Integration
- External Debug Support (compliant to RISCV Debug Specification)
- Tracing (compliant to RISCV N-Trace Specification)
- Semi-hosting
- Multiple levels of configuration

** All the features are independent of the underlying ISS's native compatibility with them.

CST-ISS: Using any ISS in plug & play fashion

- **Vast variety of supported RISCV ISA extensions. (just have to pick the ISS as per requirement)**
- **Currently Supported (and tested) open-source Instruction Set Simulators:**
  - Spike (From RISC-V Software)
  - VeeR (From ChipsAlliance)
  - Whisper (From Tenstorrent)
  - MPACT-riscv (From Google)
- **Ability to add support for any other ISS**
- **Develop your own ISS**
  - Just focus on instruction simulation, All the common features come from CST-ISS
- **Dynamic Selection of underlying Instruction Set Simulators**
  - CST-ISS uses Dynamic Loading of the pre-built ISS libraries and hence, can be switched from one ISS to another when launching the simulation => No recompilation required.

Case Study: Multi-ISS Workflows

- Individual ISSs come with certain supported features and hence, can be selected based on the use-case.
- For example:

Spike ISS

Functionally Accurate (reference model), but not targeted for high-performance

VeeR ISS

Fast ISS, tuned for SweRV cores, but limited support for MMU and suitable only for single-core simulations

Whisper ISS

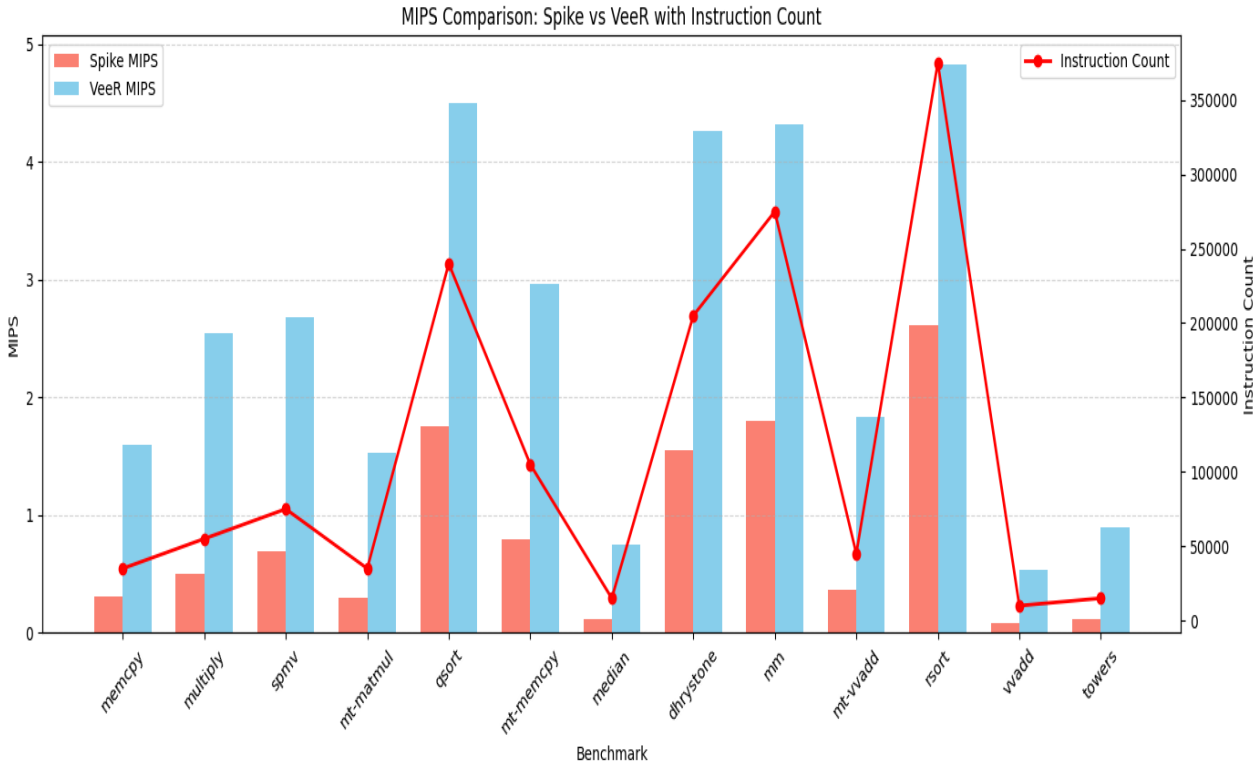
Built on VeeR ISS, increased functionality, decreased performance

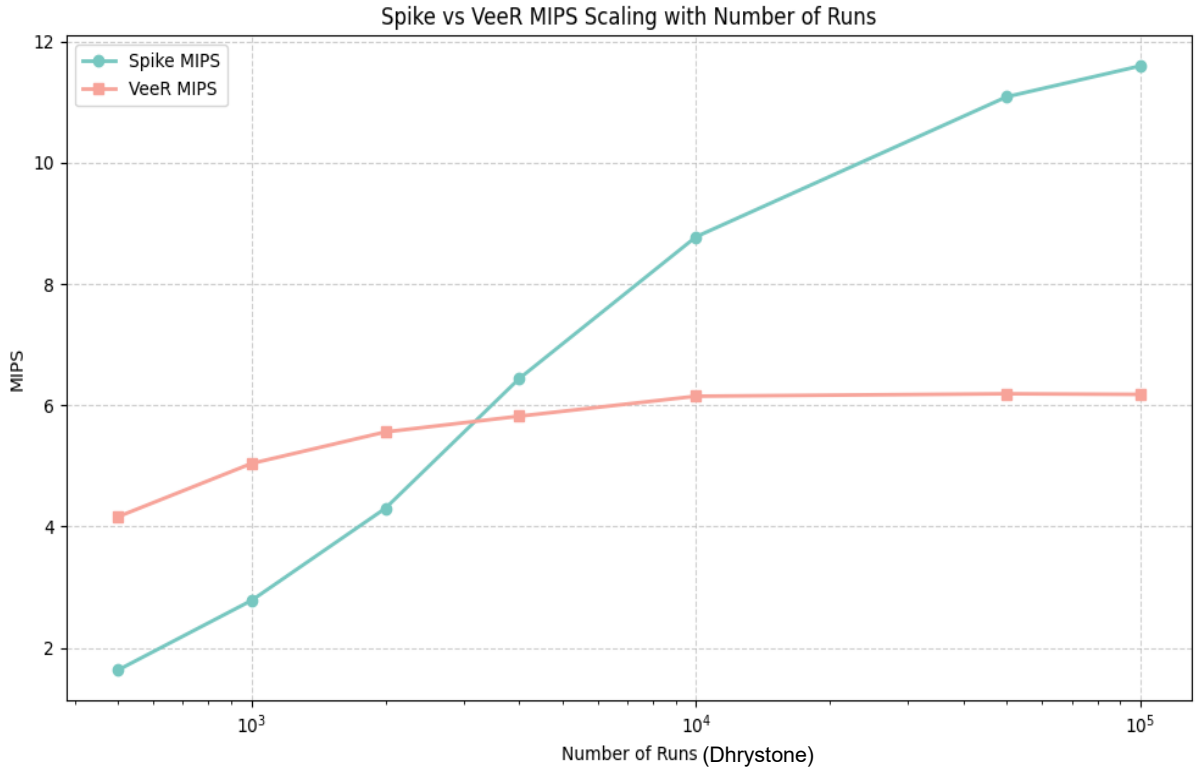
MPACT-riscv

Highly configurable, modular research framework

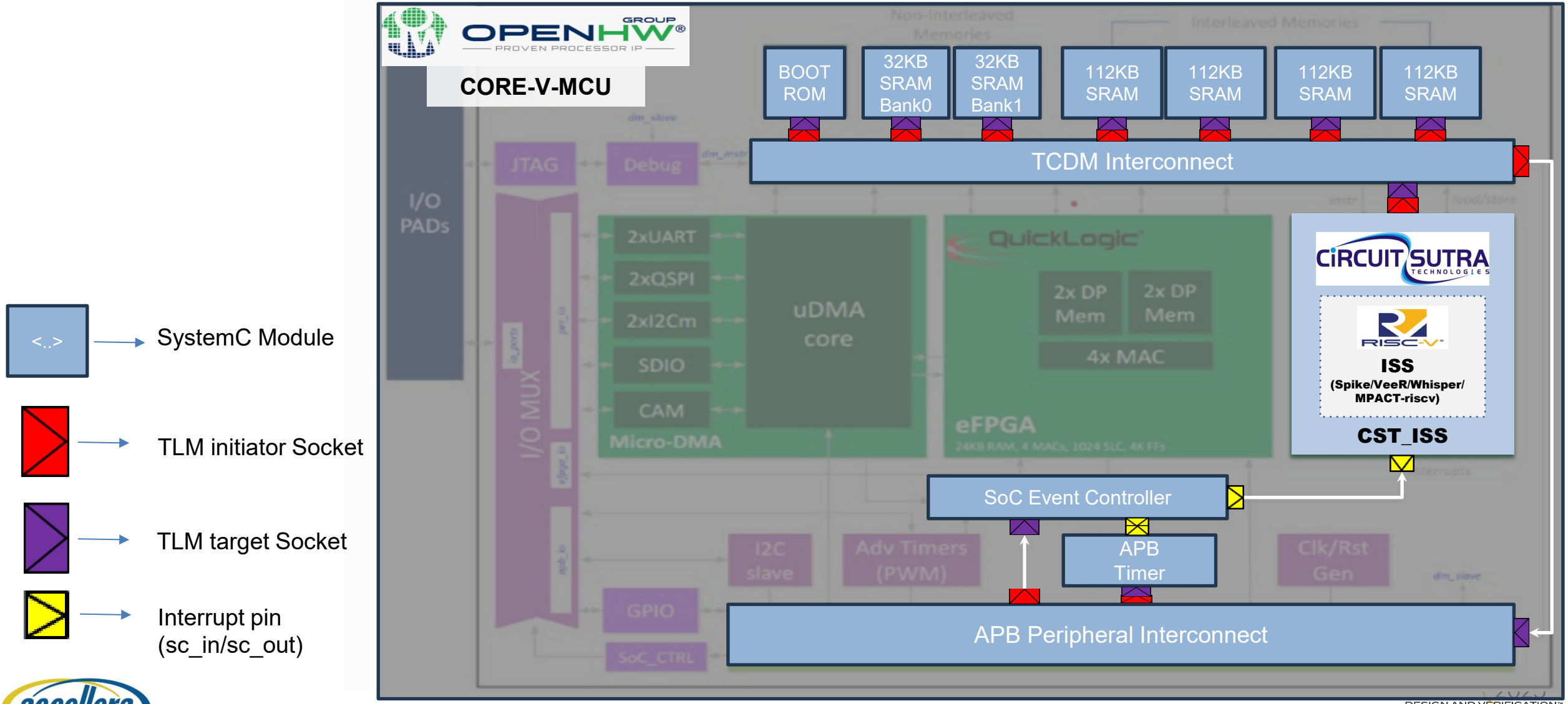
Case Study: Multi-ISS Workflows

- Performance Analysis using RISCV Benchmarks (Spike vs VeeR):



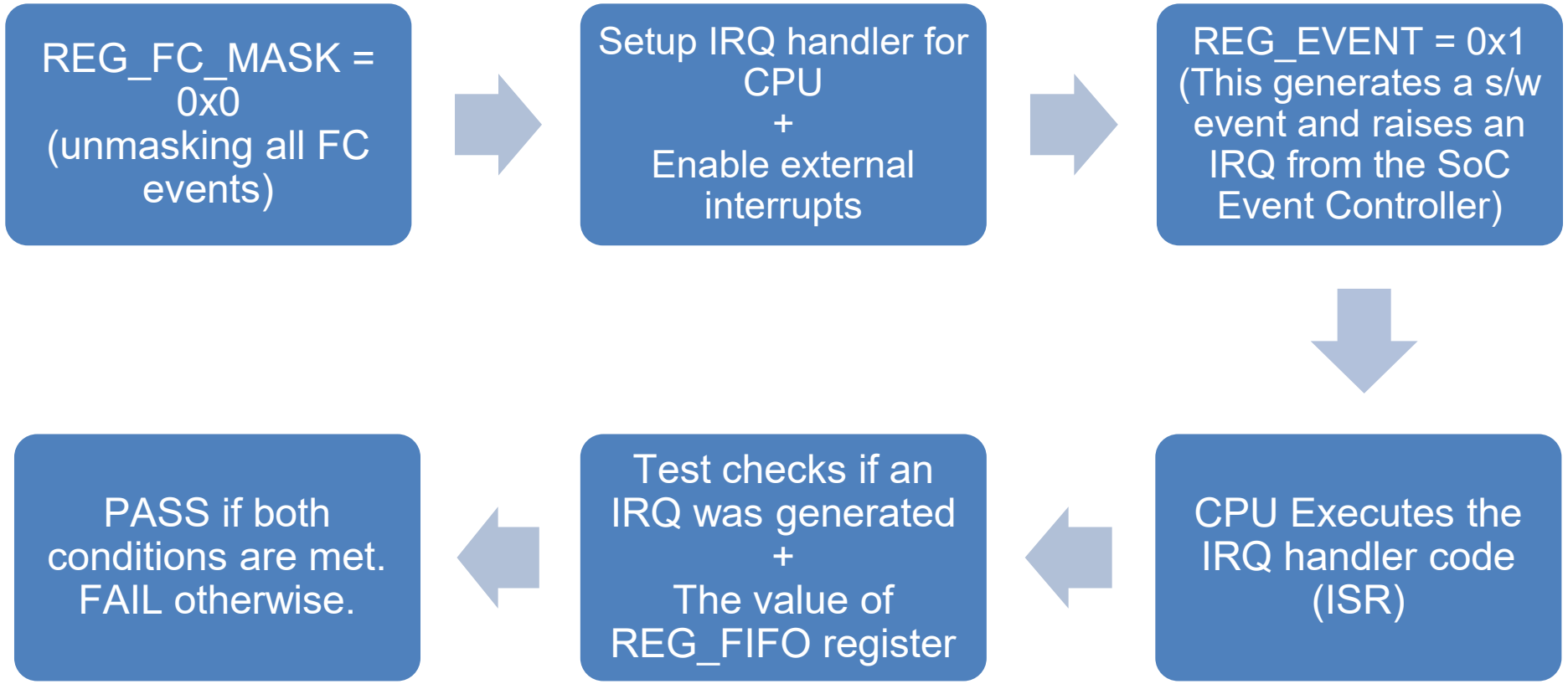


Demo Board Setup



Sample FW Test

- FW test flow:



Demo 1: Sample FW using Multi-ISS

- Running the FW on the demo board with underlying ISS as:
    1. VeeR (ChipsAlliance)
    2. Spike (riscv-isa-sim)
    3. Whisper (Tenstorrent)
    4. MPACT-riscv (Google)
- Running simulation in 32-bit and 64-bit modes

CST-ISS: GDB Integration

- **CST-ISS supports:**
  - Using GDB from RISC-V toolchain: For the SW/FW code (RISC-V Assembly or C)
    (GDB connecting to the model using RSP -> Remote Serial Protocol)
  - Using the GNU GDB: for the HW model code (SystemC/C++)

- **Benefits:**
  - Seamless integration with existing toolchains.
  - Standardized debugging experience for developers.
  - Enables breakpoints, watchpoints, single-step execution, and inspection of registers/memory.
  - HW and SW co-debugging:
    Ability to closely explore the SW/FW flow and its effects on the HW model execution in real-time

Demo 2: Sample FW using GDB

Running the FW with GDB enabled demonstrating HW and SW co-debugging.

CST-ISS <-> GDB

CST-ISS: External Debug Support

CST-ISS is compliant with the official RISCV-Debug specification.

Benefits:

- Ensures compliance with RISC-V ecosystem standards.
- Enables external tools to control and investigate the simulation transparently.
- Supports hardware-like debug flows in a virtual environment.
- Compatible with tools such as OpenOCD, GDB, and commercial probes.

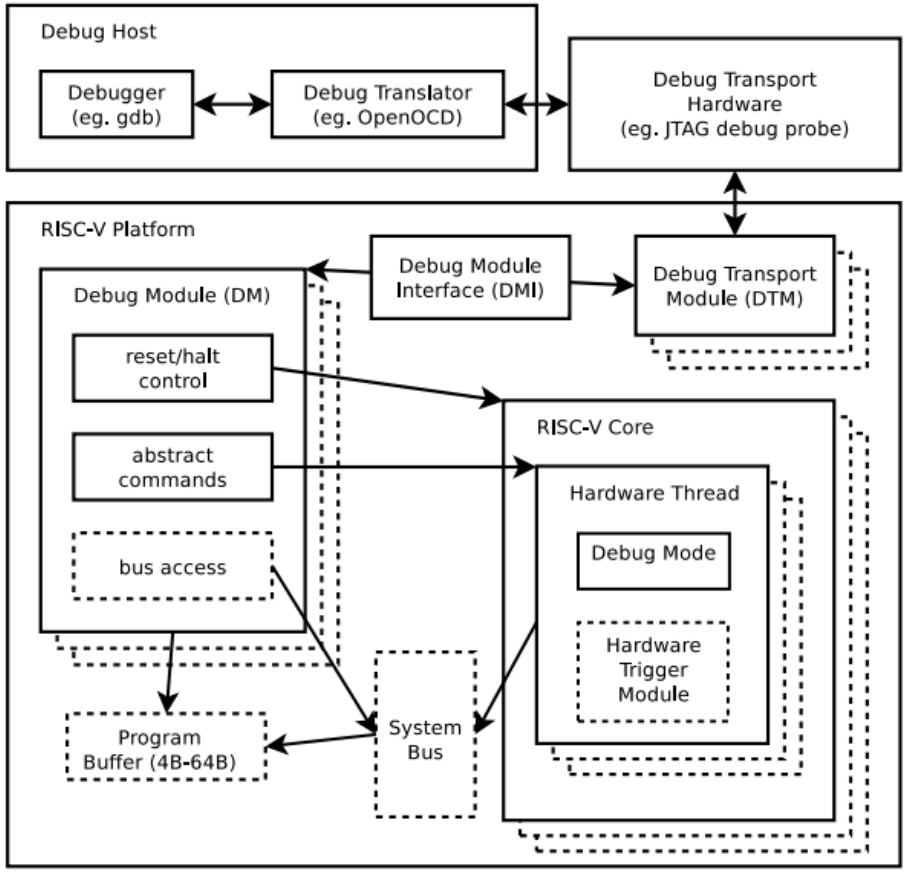


Figure 2.1: RISC-V Debug System Overview

Demo 3: Sample FW using External Debug

Running the FW with External Debug enabled and debugging with GDB via OpenOCD.

CST-ISS <-> OpenOCD <-> GDB

CST-ISS: Trace Support

What is Tracing?

- Language used by processor to communicate what's happening when a program is executed.
- It allows the users to determine:
  - The exact set of instructions executed by the processor
  - The exact sequence of execution
  - Use of memory/cache
  - Occurrences of interrupts/exceptions, etc.

  **This gives good opportunities for debugging and profiling

Applications:

- Analyzing and debugging low level code like firmware, drivers, RTOS.
- Used in performance profiling, bottleneck analysis.
- Helps in understanding behavior, handling of exceptions & interrupt.

Examples:

ARM: ETM (Embedded Trace Macrocell)
Intel: Intel PT (Processor Trace)
PowerPC: Nexus Trace (IEEE-ISTO 5001)

RISC-V: E-trace (Efficient Trace), N-trace (Nexus based Trace)

CST-ISS: Trace Support

CST-ISS implements RISC-V N-trace (Nexus Based Trace) and is compliant with the official N-trace specification.

- RISC-V N-trace:
  - Based on the well-established IEEE-5001 Nexus Standard.
  - Tailored to support the trace of RISC-V ISA cores, harts and SoC/MCU designs.
  - Easier to implement from H/W perspective.
  - Offers good trace compression.

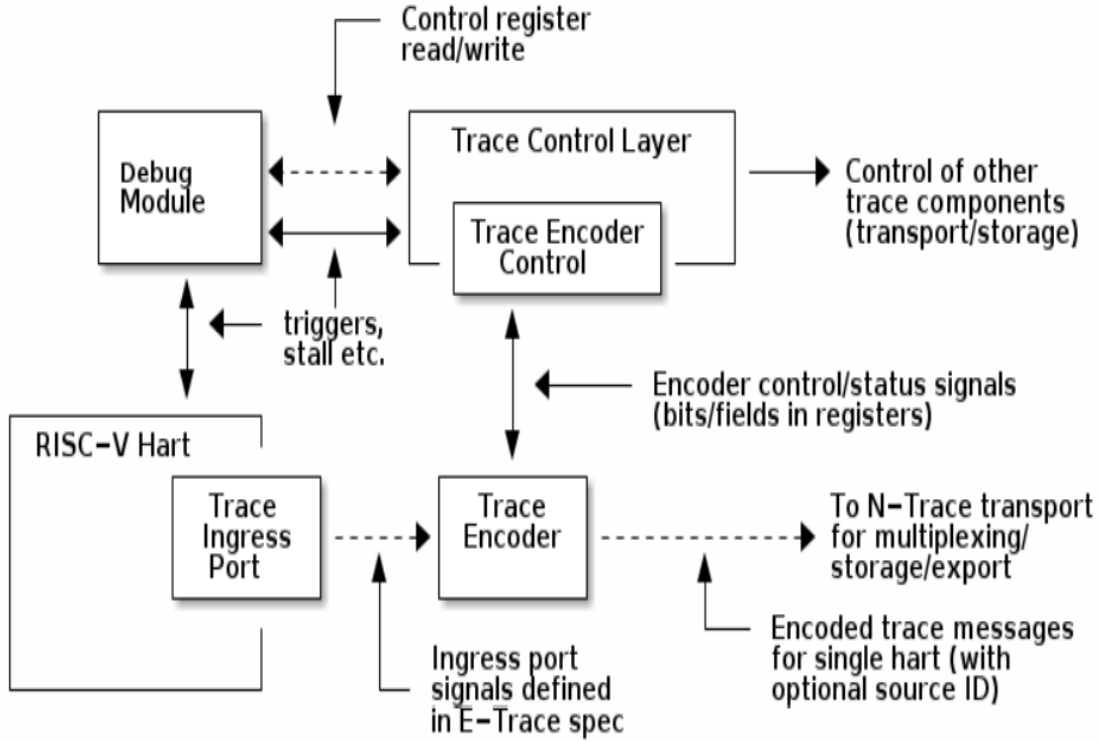


Figure 1. Trace Encoder Interfaces

CST-ISS: Trace Support

Encoding Format: [TCODE] [SRC] [optional SYNC] [optional B-TYPE] [I-CNT] [optional ADDR] [optional HIST] [optional TSTAMP]

Field	Bits	Description
TCODE	6	**Transfer Code** (6 bits) — Identifies message type (e.g., sync, error, trace).
SRC	Cfg	**Source ID** — Distinguishes harts (cores) in multi-core trace setups.
SYNC	4	**Synchronization Code** (optional) — Indicates context reset (debug exit, etc.).
B-TYPE	2	**Branch Type** (optional) — Describes control-flow type (branch, exception).
I-CNT	Var	**Instruction Count** — Number of instructions retired since last message.
ADDR	Var	**F-ADDR / U-ADDR** (optional) — Full or XOR-compressed PC address.
HIST	Var	**Branch History Mask** (optional, HTM only) — Bits encode taken/not-taken paths.
TSTAMP	Var	**Timestamp** (optional) — Absolute or relative time; resets on SYNC messages.

Messages are byte-packed via [MSEO (2 bits) + MDO (6 bits)]

CST-ISS: Trace Support

Example:

PC	Instructions
0x100	addi x1, x0, 1
0x102	beq x1, x2, 0x200
0x106	addi x3, x0, 3
0x10A	addi x4, x0, x3

Byte	Field	Value (Hex)	Description
1	**MSEO + MDO**	0xC0	Start of message (MSEO = 11, MDO = 000000)
2	TCODE	0x00	Program trace sync
3	SRC	0x00	Source core ID
4	SYNC	0x03	Sync: Exit debug mode
5	I-CNT	0x04	4 instructions retired
6	F-ADDR	0x80	Varint part 1 (0x100 >> 1 = 0x80)
7	F-ADDR	0x00	End of varint (last byte)
8	HIST	0x00	one branch = not taken
9	TSTAMP	0x01	Optional timestamp
10	**MSEO + MDO**	0xC0	End of message (MSEO = 11, MDO = 000000)

CST-ISS: Trace Support

- **CST-ISS generates required N-trace compliant trace packets which can be decoded using:**
  - Reference C implementation decoder (from the official RISC-V GitHub) https://github.com/riscv-non-isa/tg-nexus-trace/tree/main/refcode/c
    - Requires the trace data in a **binary file** and runs in **post-simulation** phase: Difficult to integrate with **real-HW like** flows
    - Generates **PC sequence** as output to validate correctness of the trace
    - **Limited support** for trace packets: Doesn't support interrupts or Debug Mode entry/exit sequences
  - CircuitSutra's inhouse N-Trace decoding Utility (**cst_n_trace_decoder**)
    - Connects with the model using **TCP port** during simulation
    - Collects the data from the **Trace sink** (configurable memory in CST-ISS)
    - Supports packet sequences for **Interrupts** and **Debug Mode entry/exit**
    - Can generate output containing:
      - The entire instruction trace (in sequence of execution)
      - N-Trace packets generated

Demo 4: Sample FW using N-Trace

Running the FW with N-Trace enabled and decoding using cst_n_trace_decoder

[CST-ISS + N-Trace] <-> cst_n_trace_decoder

CST-ISS: Semi-hosting (printf) support

- **What is semi-hosting?**
  - A lightweight method to provide host (e.g., OS, kernel, etc.) services to the target program (e.g., FW running on a simulator).
    These services include
    - Console I/O
    - File I/O
    - Exit codes
    - …
  - ISSs like Spike, support this functionality natively by providing interfaces like HTIF (Host-Target interface)
  - Not supported by all ISSs out-of-the-box (E.g., VeeR ISS)

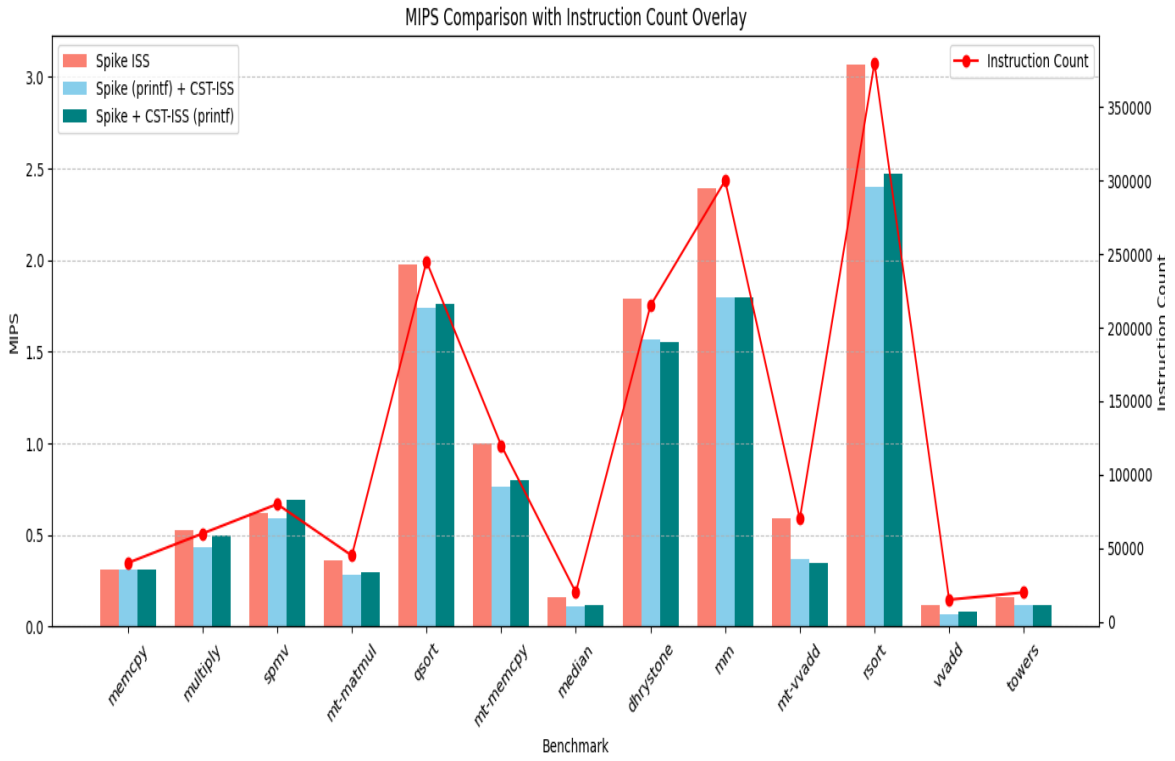
CST-ISS: Semi-hosting (printf) support

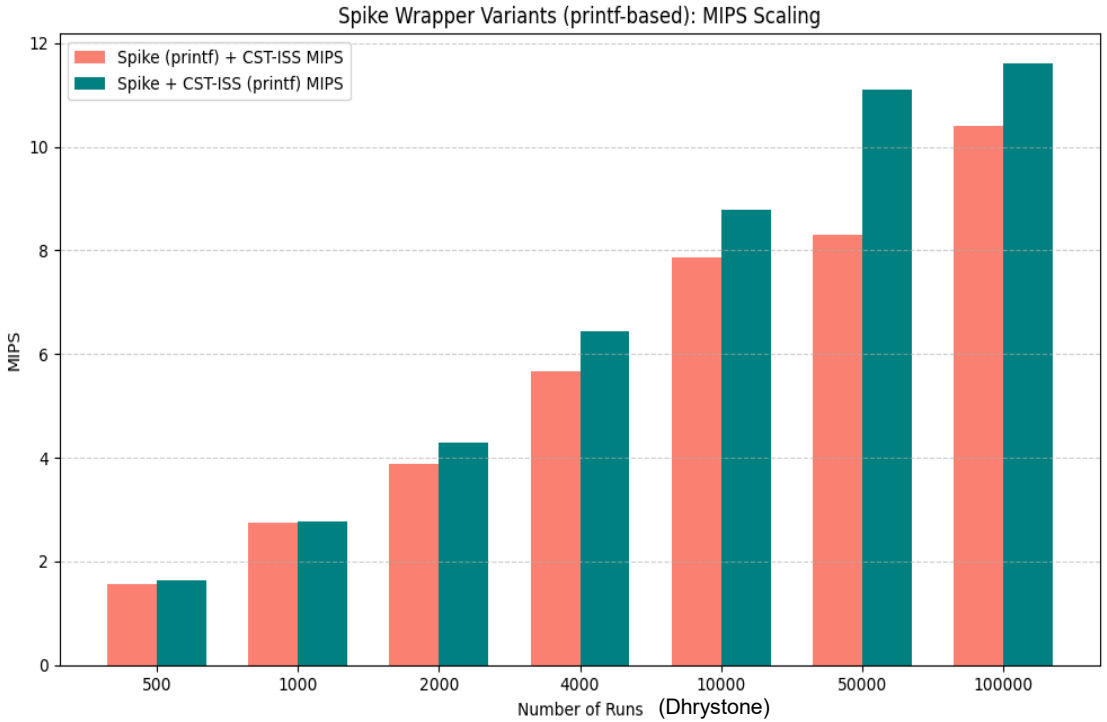
- **CST-ISS provides a non-intrusive method of semi-hosting which can be utilized by the underlying ISSs**

- **Benefits:**
  - A unified method to support the host-target interactions
  - No changes required in the ISS code -> Plug and Play
  - “printf“ statements in the FW/SW will be handled by CST-ISS (not the underlying ISS)
  - Easy-to-extend interface to support more semi-hosting features.
  - “lightweight” mechanism results in improvement in the overall performance.

CST-ISS: Semi-hosting (printf) support

- Performance Analysis using RISCV Benchmarks
  - “Spike (standalone ISS)” vs “Spike (with native printf) + CST-ISS” vs “Spike + CST-ISS (with wrapper’s printf)”



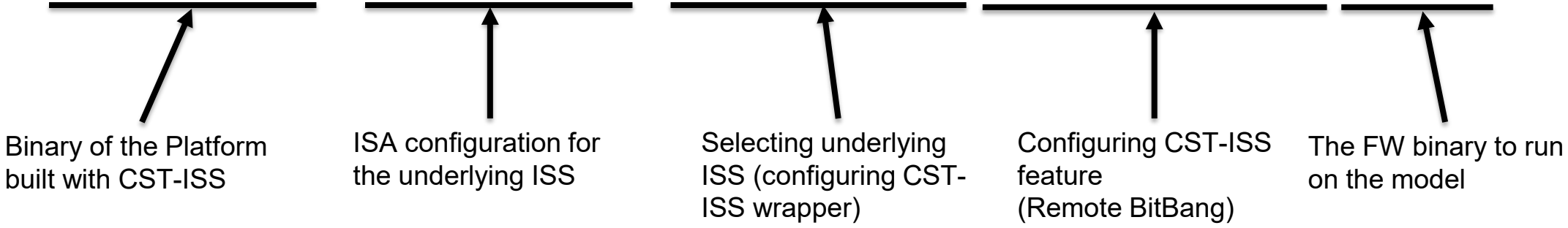


CST-ISS: Multi-level Configuration

- **We have 2 separate levels of configuration options available:**
  1. CST-ISS related configurations (to configure the SystemC world & Features)
  2. ISS related configurations (to configure the underlying ISS)

Example command line:

<Platform/Model> --isa=rv64imafdc --iss_lib=libveer.so --cst_rbb_port=1234 test.riscv



Demo 5: FreeRTOS on Core-V MCU

- Booting FreeRTOS on the Core-V MCU VP board
- Using GDB (RSP) to enable FW debugging

Success & Benefits

- Till date, CST-ISS has been successfully deployed in:

Development and Testing of the CORE-V MCU Virtual Prototype	Setting up development and validation workflows for bare-metal FW tests	Setting up workflows involving RISCV Standard extensions (E.g., External Debug, N-trace, etc.)
Consuming and validating RISCV mainstream tools (Compilers, GDB, etc.)	Validation of RTOS compatibility for RISCV (FreeRTOS on CORE-V MCU)	Evaluation and comparison of various available ISSs across different functional and performance aspects.

CST-ISS: Future RoadMap for RISCV

- Add and validate support for more ISSs. E.g., TinyEMU, QEMU, etc.
- Add support for more RISCV standard tools (E.g., E-trace, etc.)
- Increase adoption in the industry: have more VP SoCs that utilize the framework
- Boot more RTOSs like ThreadX (from Microsoft), Zephyr, etc. on SoCs using CST-ISS (E.g., Core-V MCU VP, and more…)

SystemC Modelling Services

We set up high performing world class SoC modelling teams to work as the extension of the customer's team in INDIA & NORTH AMERICA.

Supporting the industry since 2005, we are the only company with 100% focus on SoC Modelling

Awarded by Accellera Systems Initiative

Outstanding Contribution Award for Driving Accellera Standards in the Indian Ecosystem (DVCon India 2016)


Questions ? ? Answers ?
DVCON India
Visit Our Booth #35
info@circuitsutra.com,
Thank You !
Forbes INDIA
DGEMS
2024
200
SELECT
Companies with
Global Business
Potential
CIRCUIT SUTRA
TECHNOLOGIES
Modelling Silicon Dreams