بسم الله الرحمن الرحيم

OUTLINE

1. **Introduction and Importance of IC Design Verification**
2. **Lecture**
   - Functional Verification of DCLS Feature of RISC-V
3. **Practical Demonstration**
   - Simulating the open-source SweRV EH1 core using RISC-V Tool chain + Synopsys VCS

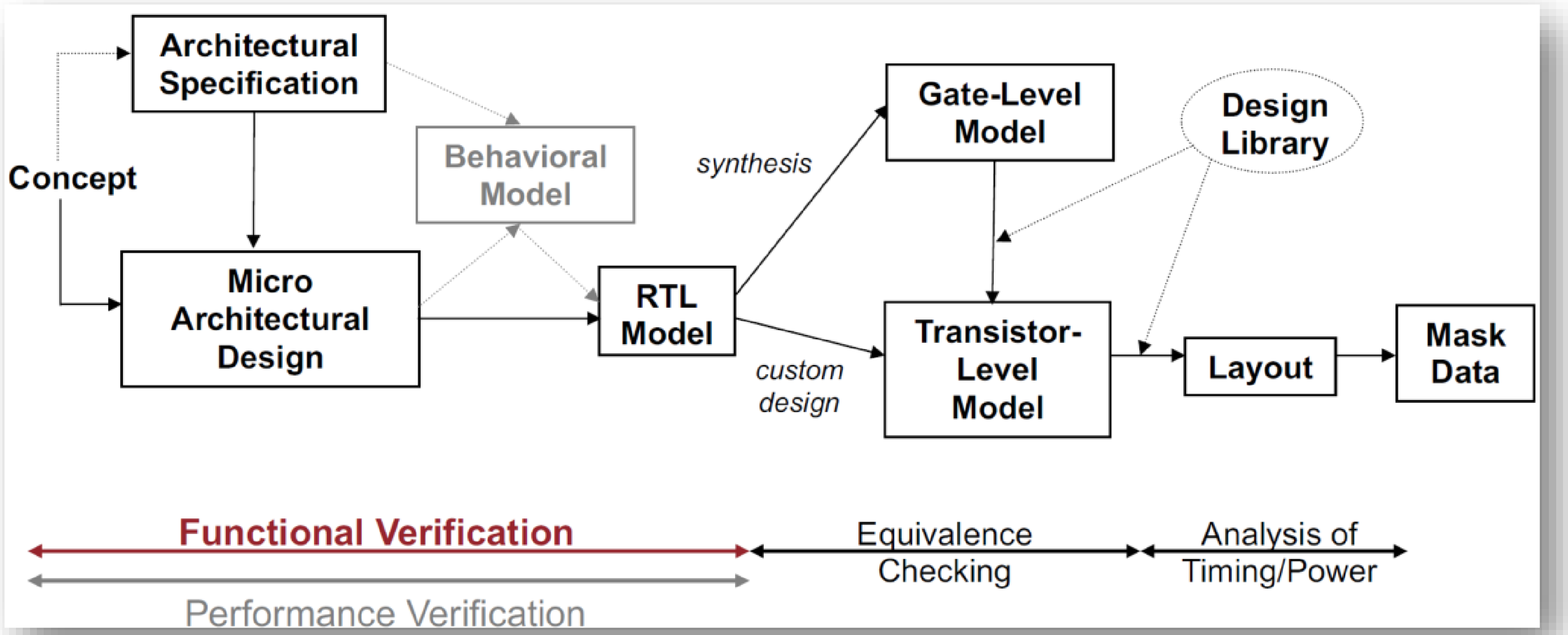
INTRODUCTION AND IMPORTANCE OF IC DESIGN VERIFICATION

WHAT IS DESIGN VERIFICATION?

Design verification is the process used to gain confidence in the correctness of a design w.r.t. the requirements & specifications.

VERIFICATION IN THE IC DESIGN PROCESS

- Functional verification aims to demonstrate that the functional intent of a design is preserved in its implementation.



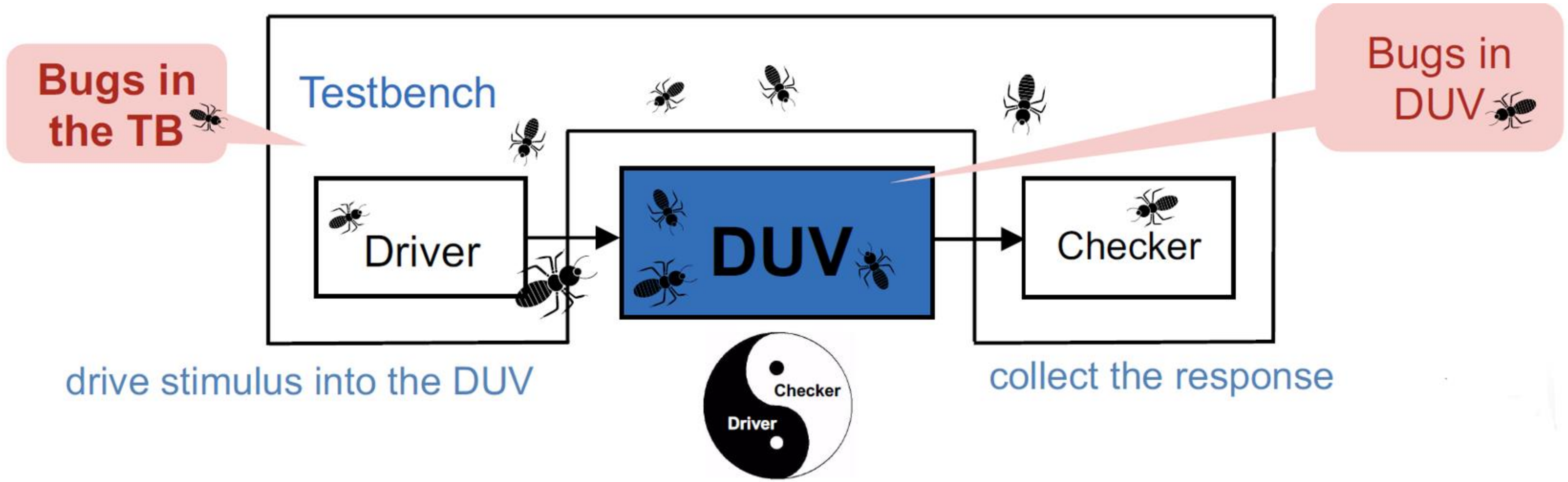
All about Bugs

Types of bugs

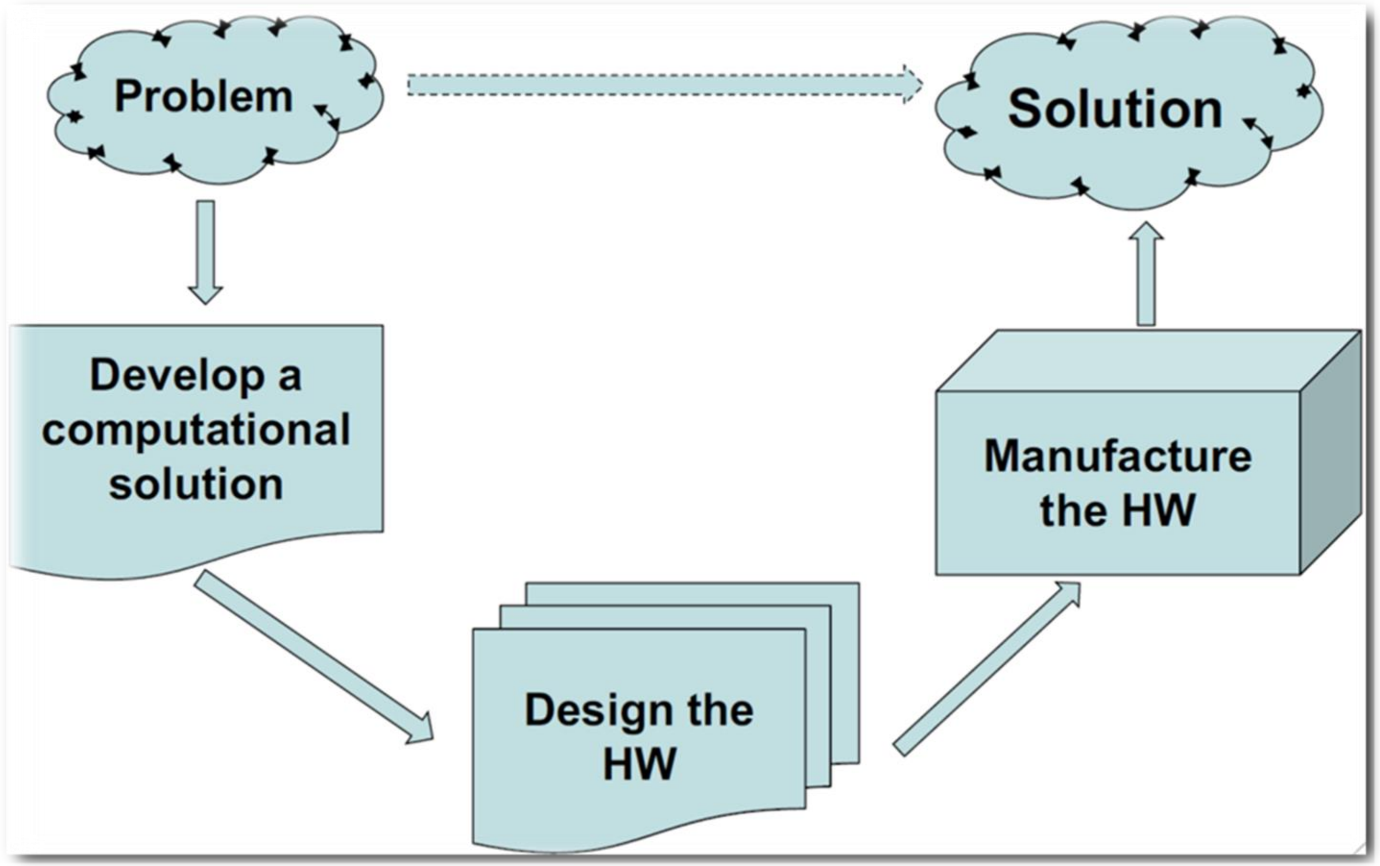
How are bugs introduced?

How can bugs be found?

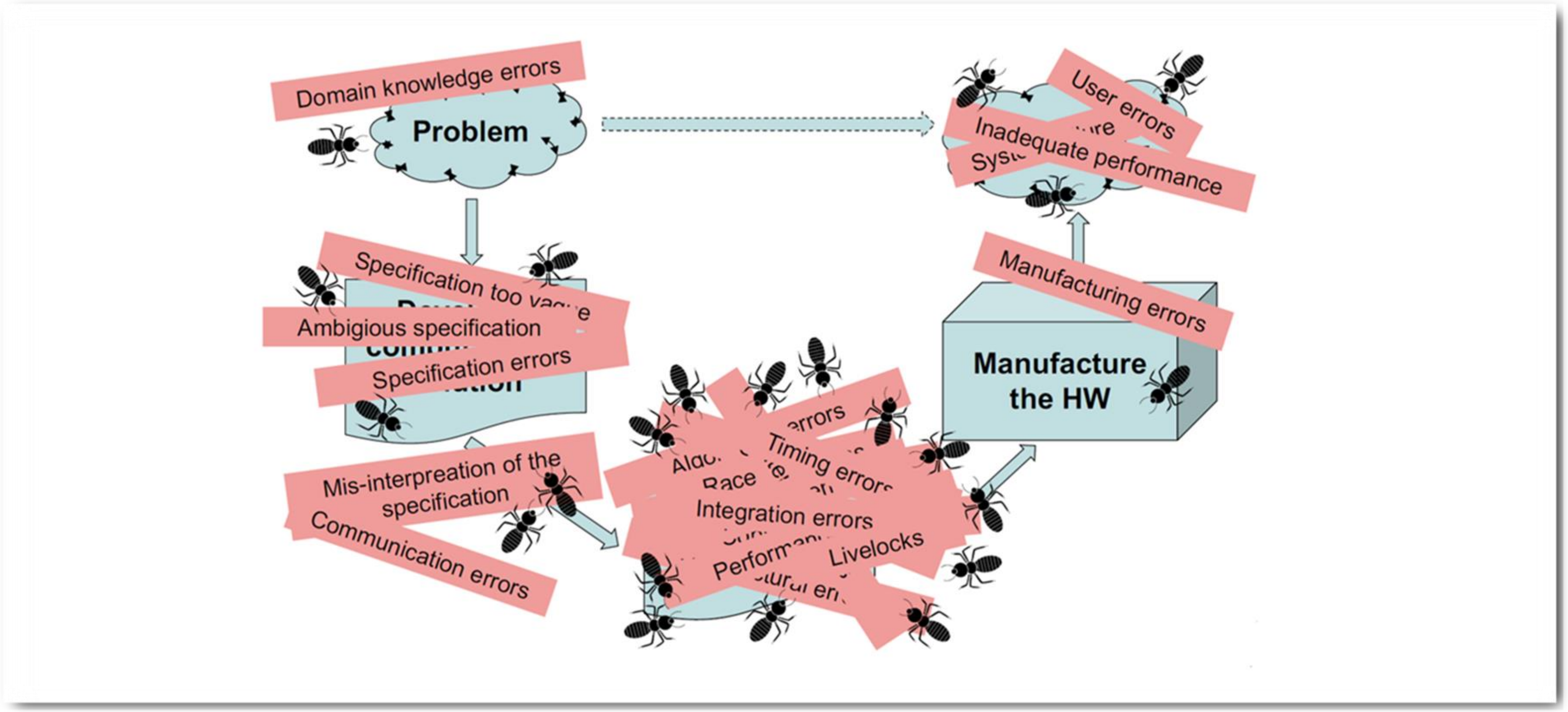
WHY IS VERIFICATION IMPORTANT?



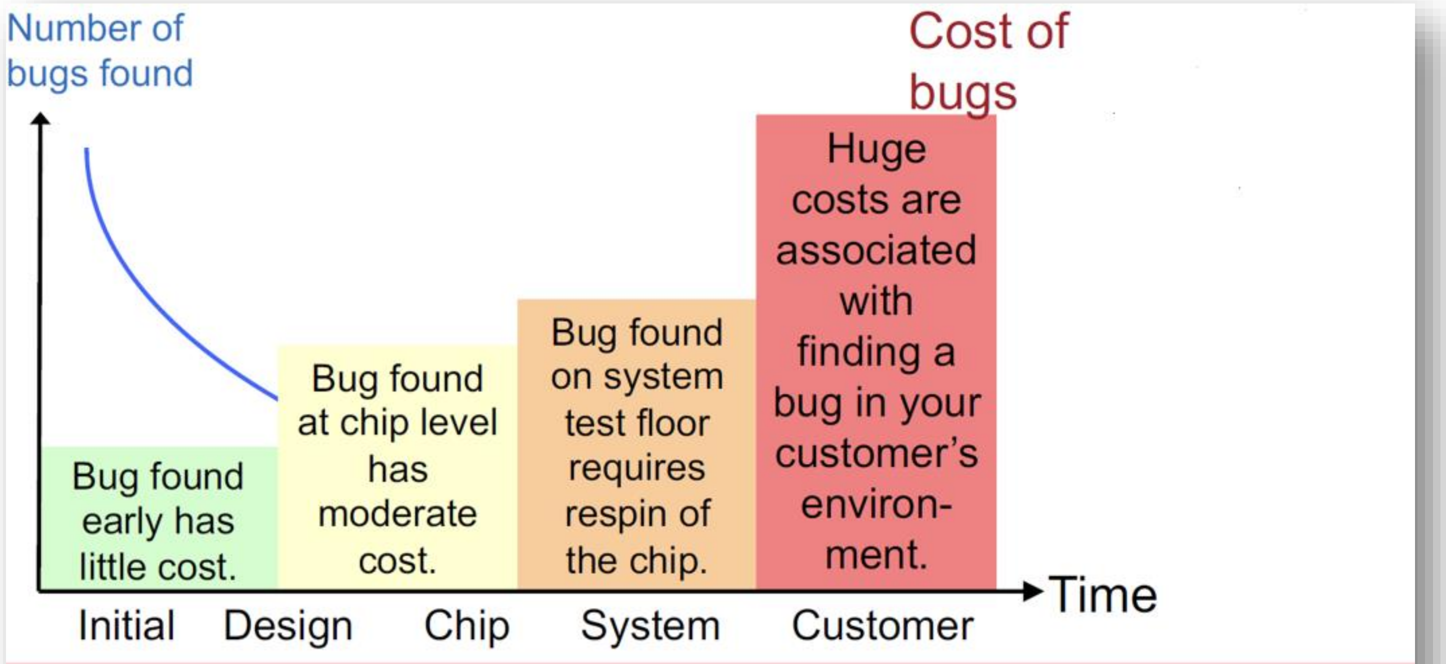
WHY IS VERIFICATION IMPORTANT?



WHY IS VERIFICATION IMPORTANT?



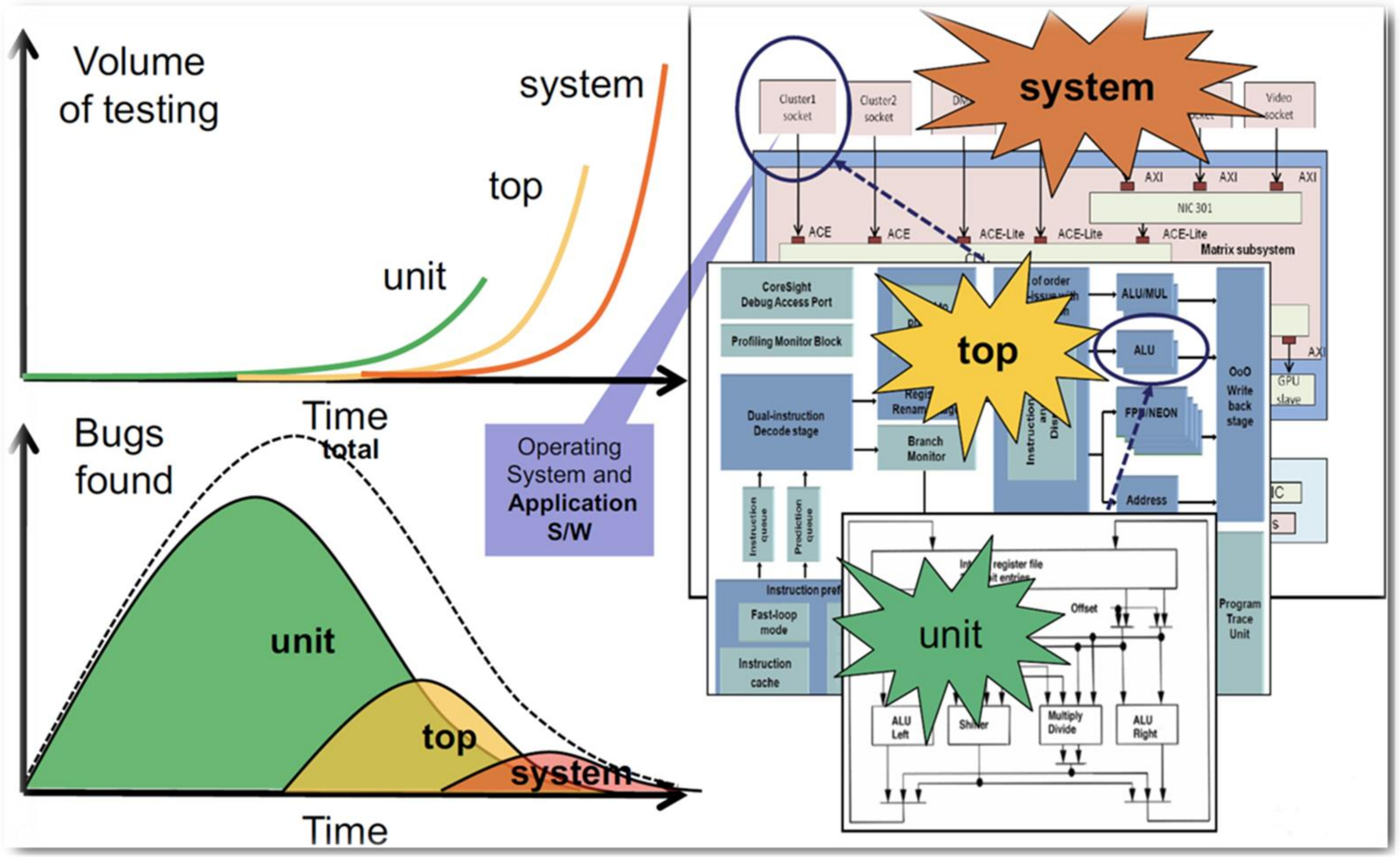
COST OF BUGS OVER TIME


Number of bugs found
Cost of bugs
Bug found early has little cost.
Bug found at chip level has moderate cost.
Bug found on system test floor requires respin of the chip.
Huge costs are associated with finding a bug in your customer's environ-ment.
Time
Initial
Design
Chip
System
Customer

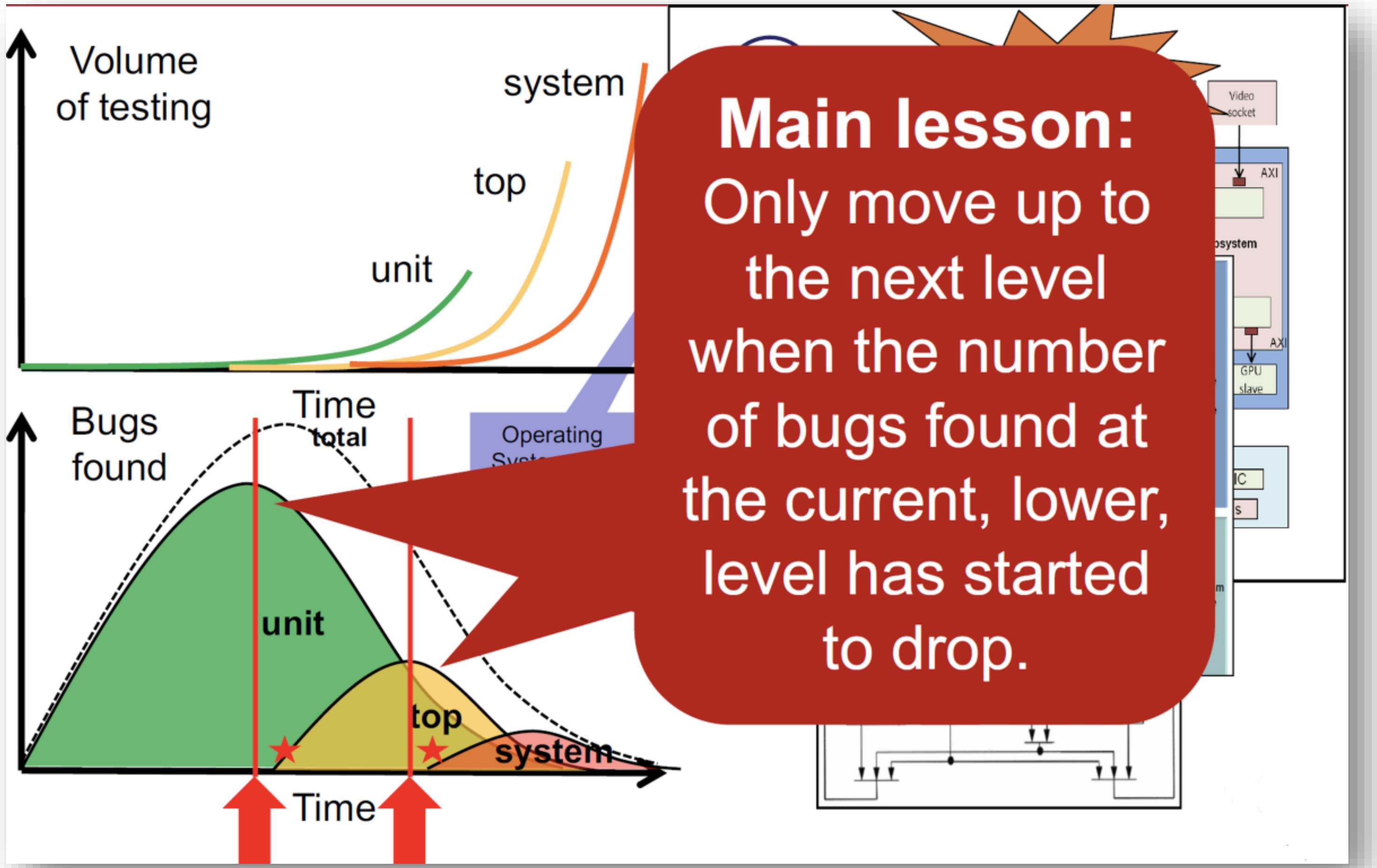
The longer a bug goes undetected, the more expensive it is!

Remember the Intel Pentium FDIV bug!
http://en.wikipedia.org/wiki/Pentium_FDIV_bug

VERIFICATION AT DIFFERENT DESIGN LEVELS



VERIFICATION AT DIFFERENT DESIGN LEVELS



Main lesson: Only move up to the next level when the number of bugs found at the current, lower, level has started to drop.

WHY IS VERIFICATION IMPORTANT?

Verification is the single biggest lever to affect the triple constraints:

- **Quality**
  1. A high-quality track record preserves revenue and reputation.
  2. Ideally a team can establish a “right-first-time” track record.
- **Cost**
  1. Fewer revisions through the fabrication/development process means lower costs.
  2. Re-spinning a chip costs hundreds and thousands of dollars.
- **Timing/Schedule**
  1. Fewer revisions through the fabrication/development process means faster time-to-market.
  2. Re-spinning a chips costs 6-8 weeks at least.

ROLE OF VERIFICATION IN IC DESIGN

- **Engineers need to balance the conflict of interest:**
  - Tight time-to-market constraints vs. increasing design complexity

- **Aim:** “Right-first-time” design, “correct-by-construction”
  - More and more time-consumed to obtain acceptable level of confidence in correctness of design!

- **Design time << Verification time**
  - Upto 70% of design effort can go into verification
  - Remember: Verification does not create value! But it preserves revenue and reputation!
  - In some cases, verification engineers out number designers 2:1

DESIGN VERIFICATION TEAM @ NECOP

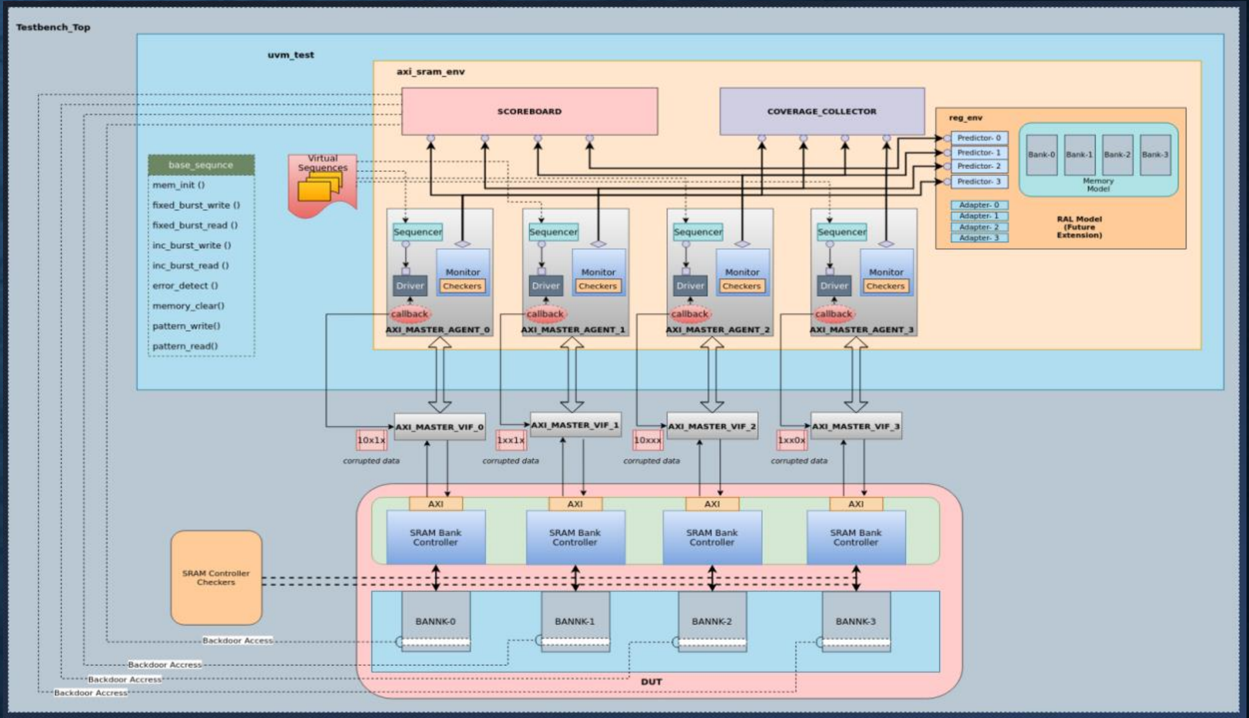
- Established in 2022
- Functional verification of RTL designs provided by the design teams

10xEngineers → Expertise Acquired → IP level testing → Provision of IP verification services

YONGATEK → Expertise Acquired → SoC level verification → Establishment of SoC verification facility

DESIGN VERIFICATION TEAM @ NECOP

- Team's Skillset
  - IP Verification
  - SoC Verification
  - CPU Verification



NECOP DV TEAM WORKING FLOW

Verification Tools:

1. Synopsys

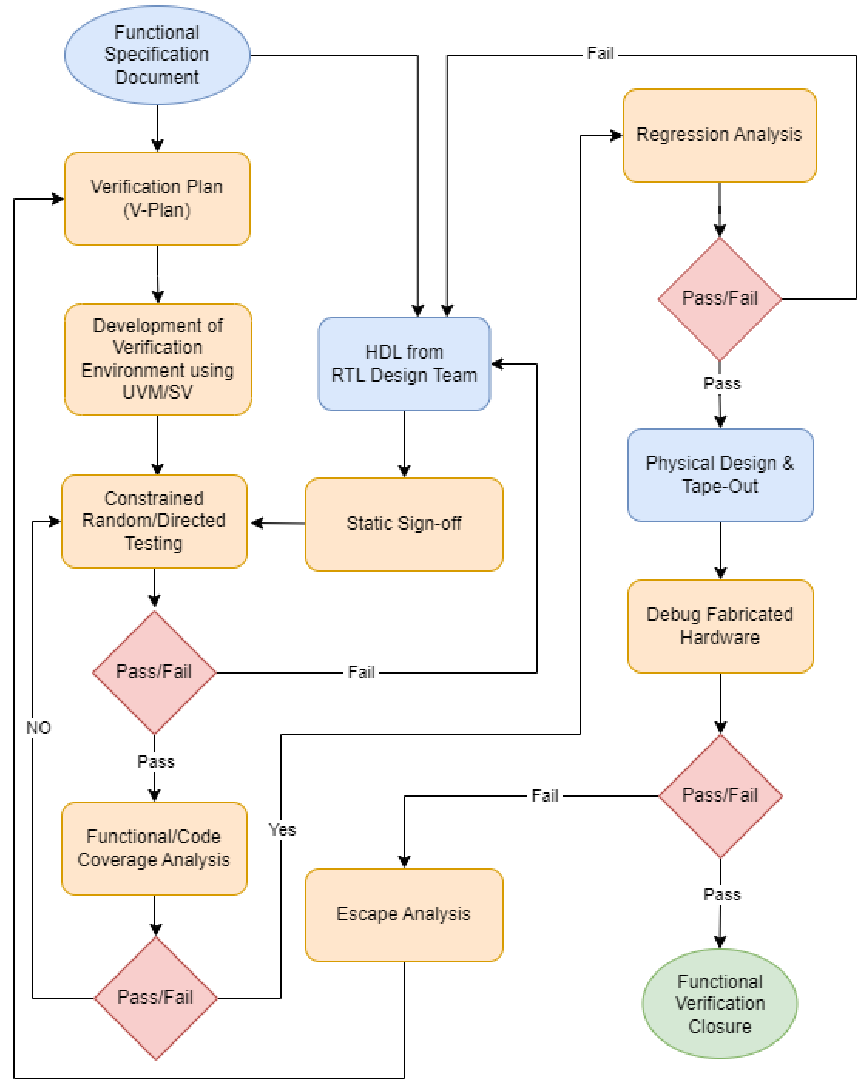
- SpyGlass
- Verdi Debug
- VCS
- VC Execution Manager

2. Cadence

- Jasper Gold
- Xcelium
- V-Manager

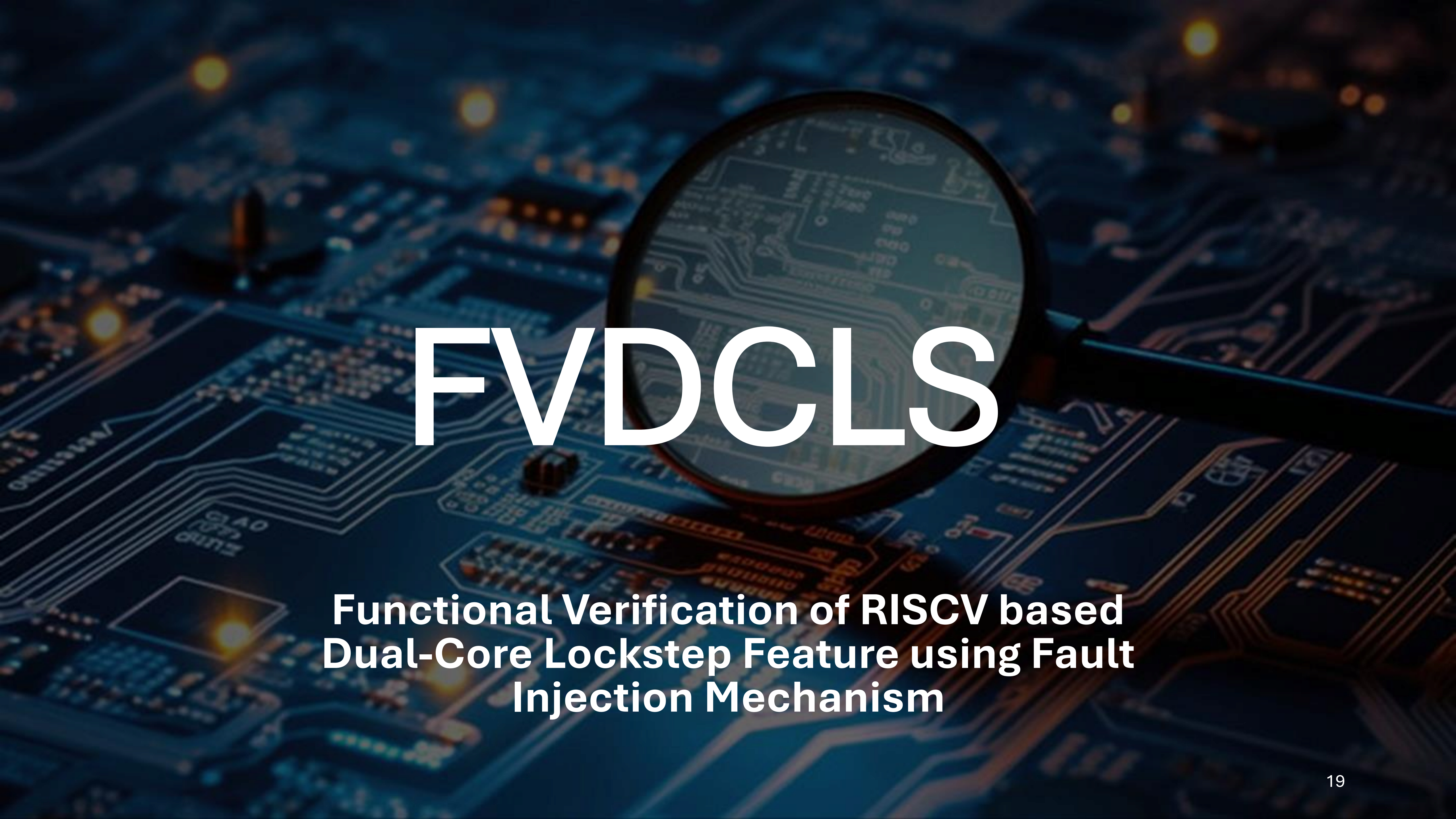
3. Mentor Graphics

- HDL Designer
- QuestaSim


Functional Specification Document
Fail
Regression Analysis
Verification Plan (V-Plan)
Pass/Fail
Development of Verification Environment using UVM/SV
HDL from RTL Design Team
Pass
Physical Design & Tape-Out
Constrained Random/Directed Testing
Static Sign-off
Debug Fabricated Hardware
Pass/Fail
Fail
NO
Pass
Fail
Pass/Fail
Yes
Functional/Code Coverage Analysis
Escape Analysis
Pass
Pass/Fail
Functional Verification Closure

VERIFICATION LANGUAGES

- Programming Languages
  - Verilog/System Verilog
  - UVM Methodology
  - C/C++
  - SystemC
- Scripting Languages
  - Makefile
  - Python
  - Bash
  - Tcl



FVDCLS

Functional Verification of RISCV based Dual-Core Lockstep Feature using Fault Injection Mechanism

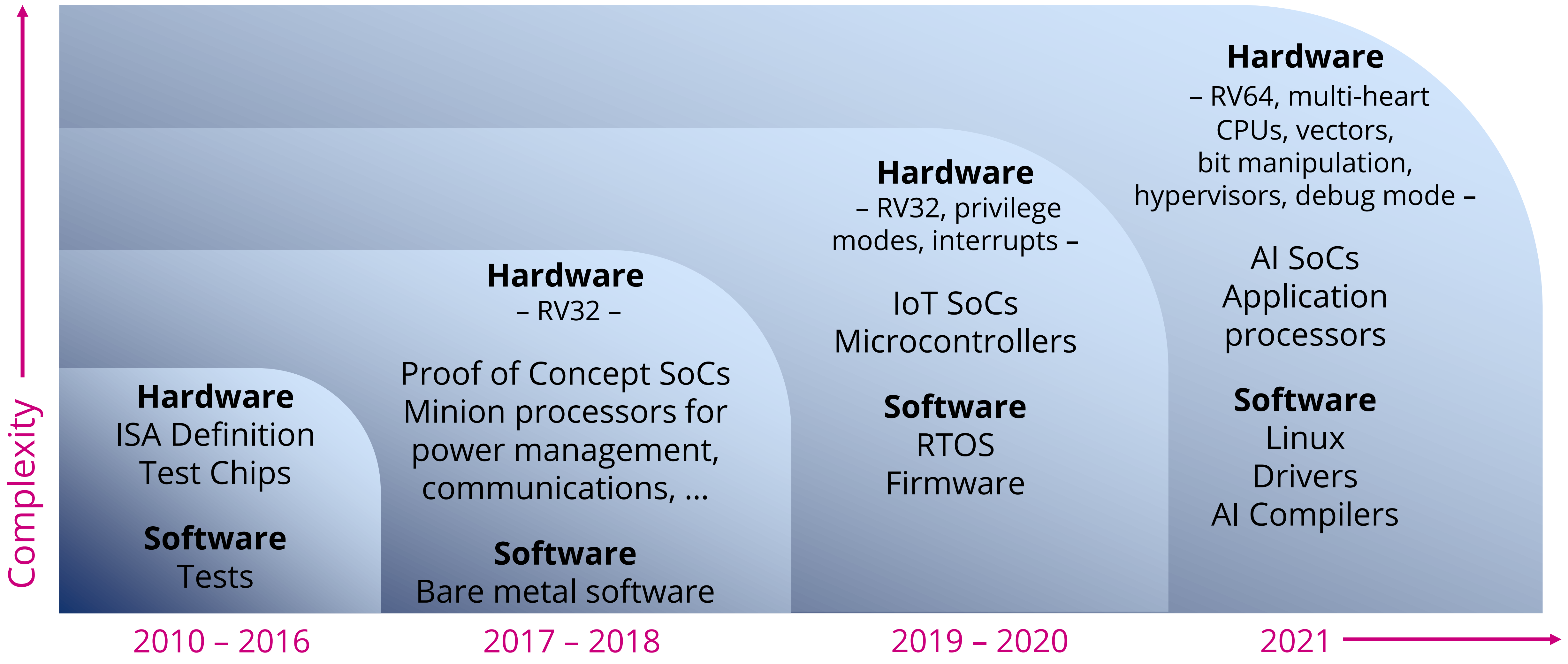
RISC-V is the free and open Instruction Set Architecture…

- … Driven through open collaboration
- … Enabling freedom of design across all domains and industries
- … Cementing the strategic foundation of semiconductors

Disruptive **Technology**

Barriers	**Legacy ISA**	**RISC-V ISA**
Complexity	1500+ base instructions Incremental ISA	47 base instructions Modular ISA
Design freedom	$$$ – Limited	Free – Unlimited
License and Royalty fees	$$$	Free
Design ecosystem	Moderate	Growing rapidly. Numerous extensions, open and proprietary cores
Software ecosystem	Extensive	Growing rapidly

Industry innovation on RISC-V



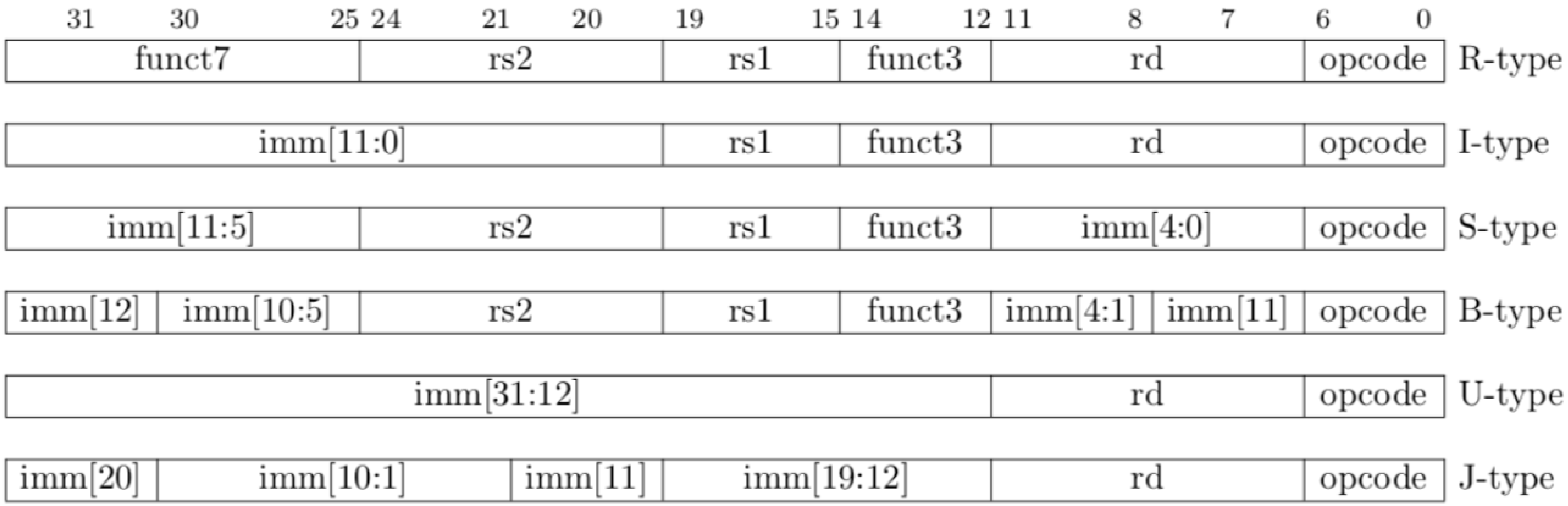
ISA Discussion

The base integer ISA is named “I” (prefixed by RV32 or RV64 depending on integer register width), and contains integer computational instructions, integer loads, integer stores, and control-flow instructions;

Integer multiplication and division extension is named “M”, and adds instructions to multiply and divide values held in the integer registers;

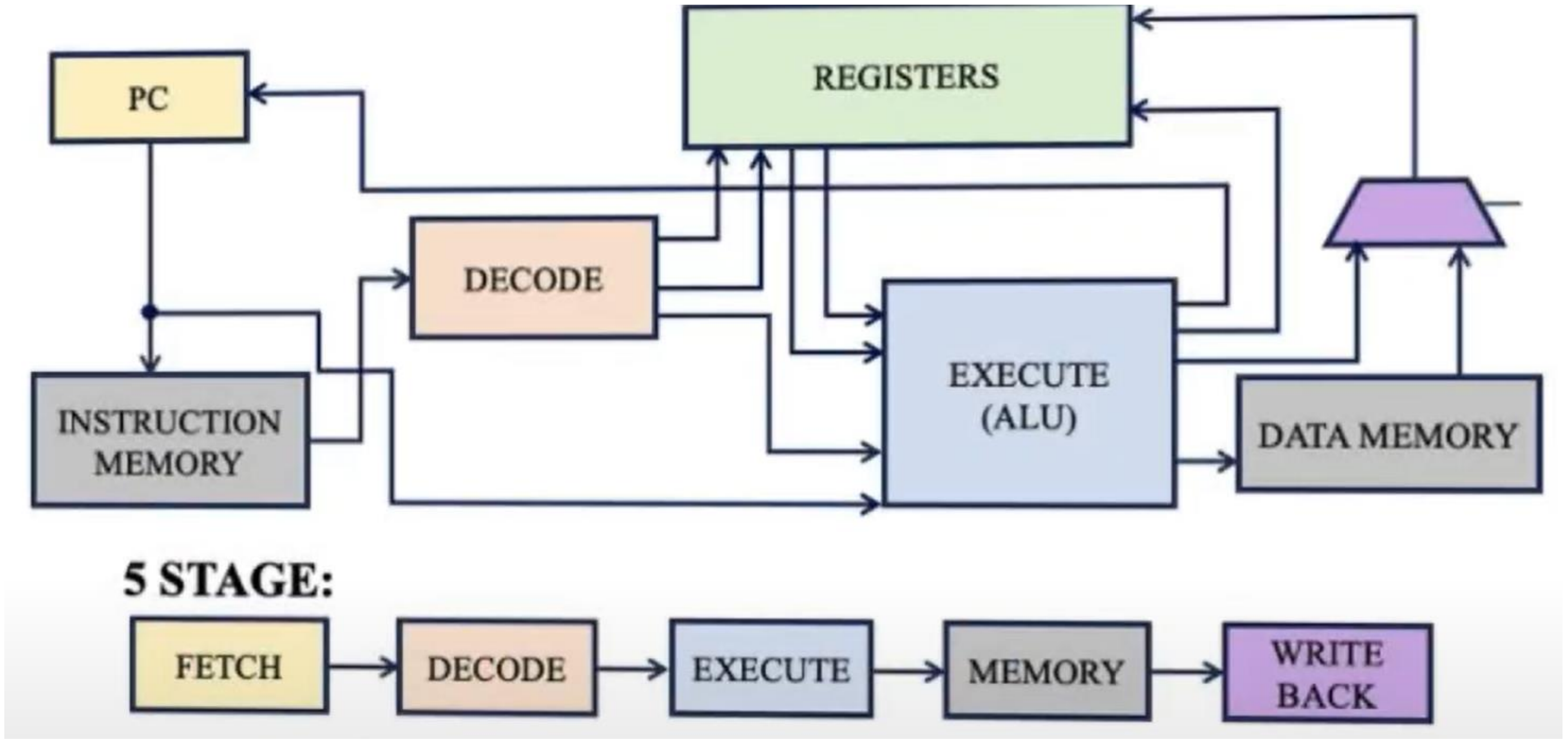
Standard single-precision floating-point extension is denoted by “F”, adds floating-point registers, single-precision computational instructions, and single-precision loads and stores;

ISA Discussion



31	30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7		rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]				rs1	funct3	rd		opcode	I-type
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

Classical 5-Stage Pipelining in RISC-V

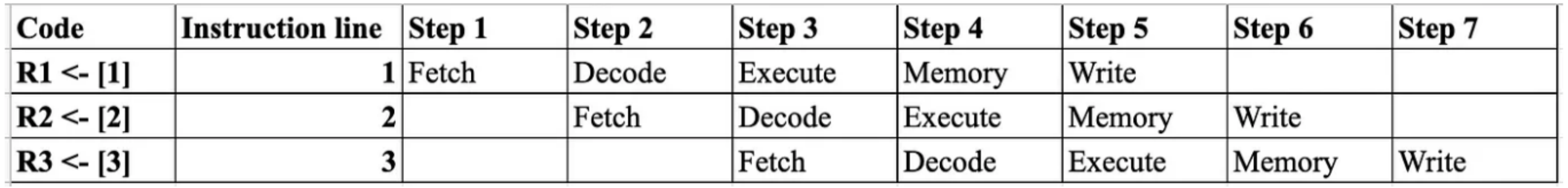


RISC-V Load/Store Architecture

```
R1 <- [1]
R2 <- [2]
R3 <- [3]
```

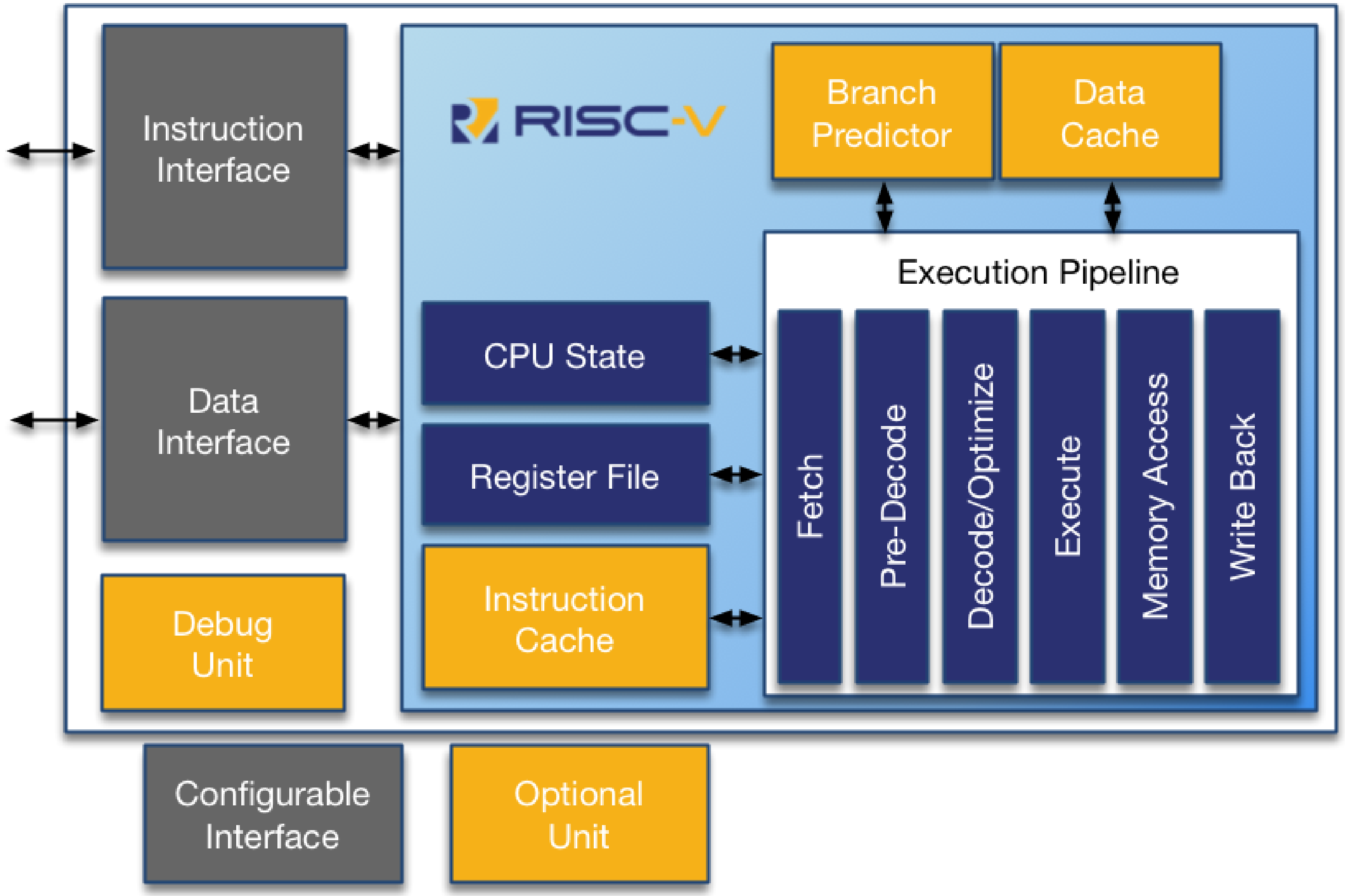
In the code above, we are performing three load types. In line one, we are storing the address 1 to R1, line 2, we are storing address of 2 to R2 and finally in line 3, we are storing the address 3 to R3.

The RISC Pipeline will look something like this:



Code	Instruction line	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
R1 <- [1]	1	Fetch	Decode	Execute	Memory	Write		
R2 <- [2]	2		Fetch	Decode	Execute	Memory	Write	
R3 <- [3]	3			Fetch	Decode	Execute	Memory	Write

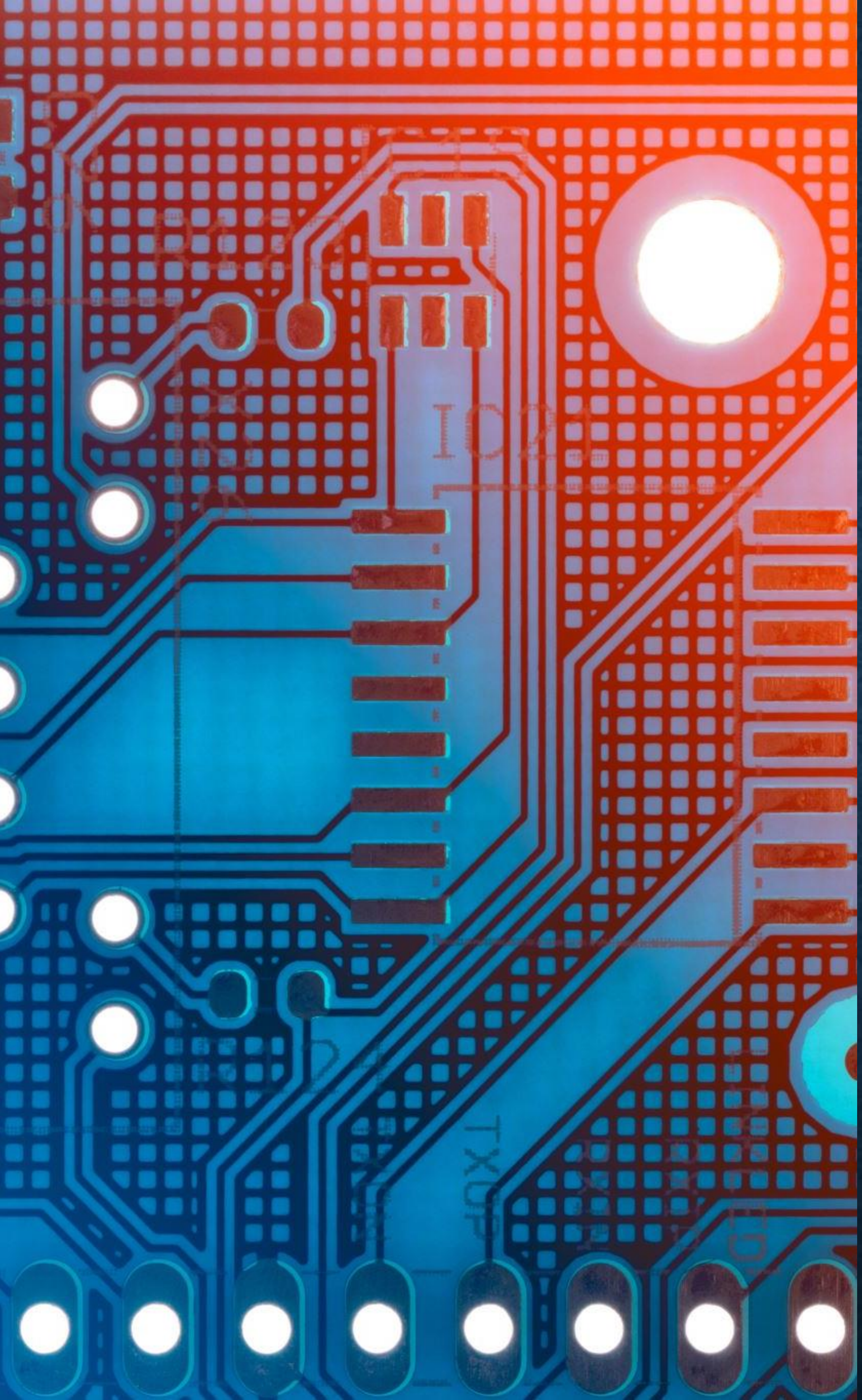
RISC-V based General Processor



MOTIVATION OF DCLS CORE

- Dual-core lockstep cores are used to
    i. Enhance fault tolerance,
    ii. Improve reliability, and
    iii. Meet the stringent safety requirements of critical applications.

- They provide a robust and proven approach to building high-reliability computing systems that can withstand hardware faults and environmental challenges.



SOURCES OF FAILURES

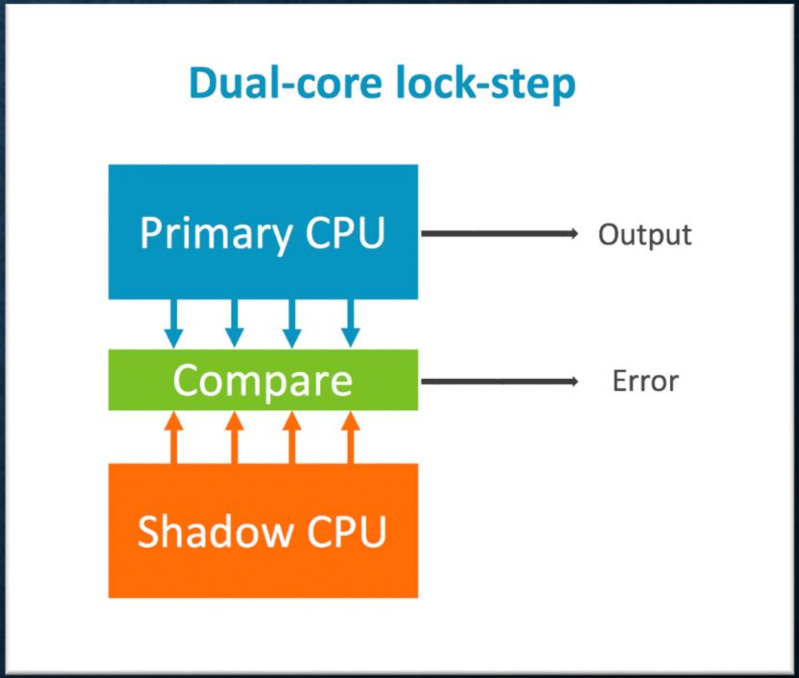
- **Radiational Issues in ICs (SEUs)**
  - Memory: There is an accidental trigger that changes the memory state in the system. Common scenarios include a hit by a radiation particle, interference from RF transmitter
- **Single Event Latchup (SEL)**
  - Short-circuits between the power signal and the ground

WHAT IS DUAL-CORE LOCKSTEP (DCLS)?

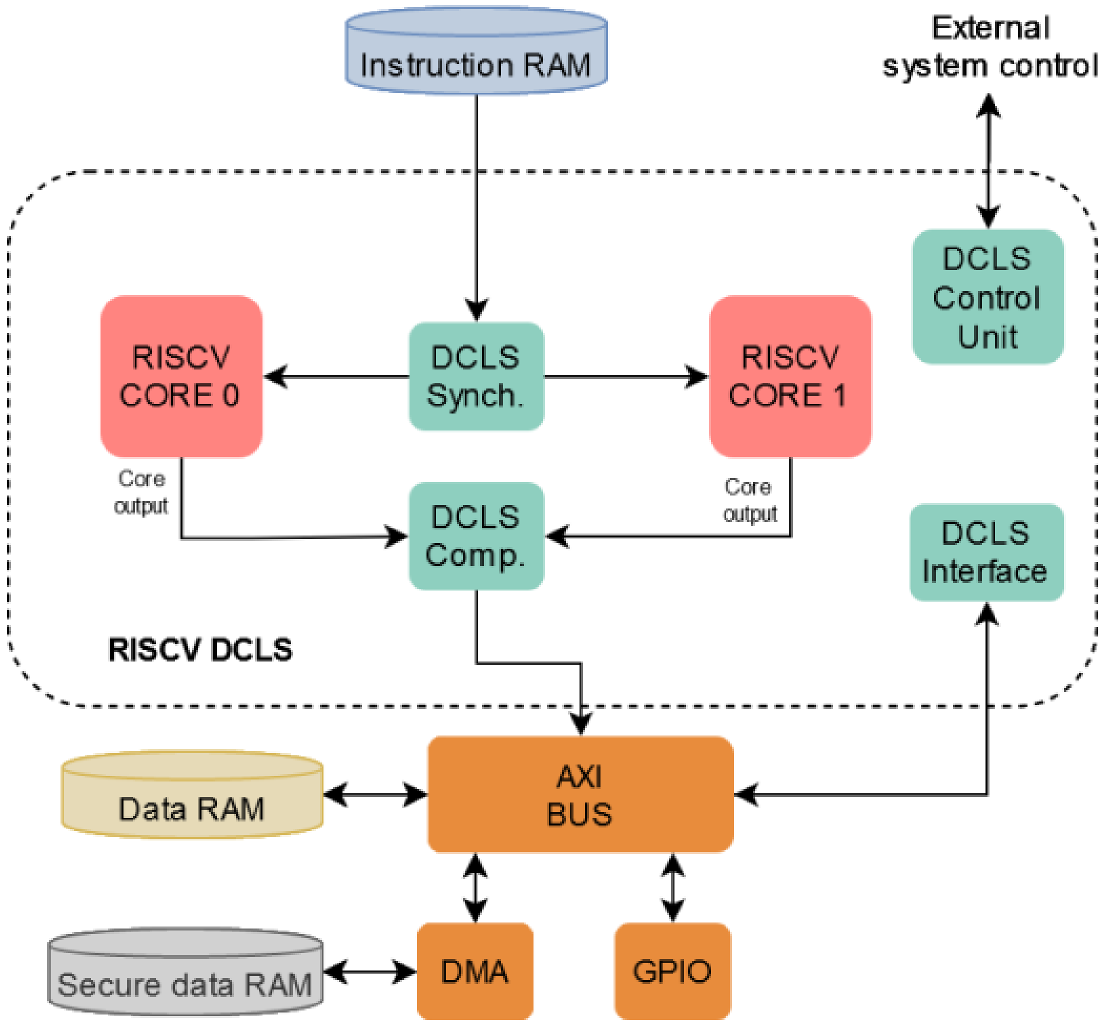
- Dual-core lockstep (DCLS) is a redundancy technique for high-reliability computing used in safety-critical systems like aerospace, automotive, and industrial control systems.

- Both core's internal states & outputs are compared at each clock cycle.

- Any divergence or mismatch between core's states is indicated as an error in the system.



GENERIC DCLS BLOCK DIAGRAM



CHECKPOINT AND ROLLBACK METHODOLOGY

- The checkpoint is an operation that saves a consistent state of the processor in the memory
- The rollback recovers the system from an error by restoring that previous state
- When the Checker detects a mismatch in the CPU’s data output the interrupt is launched to perform a rollback.

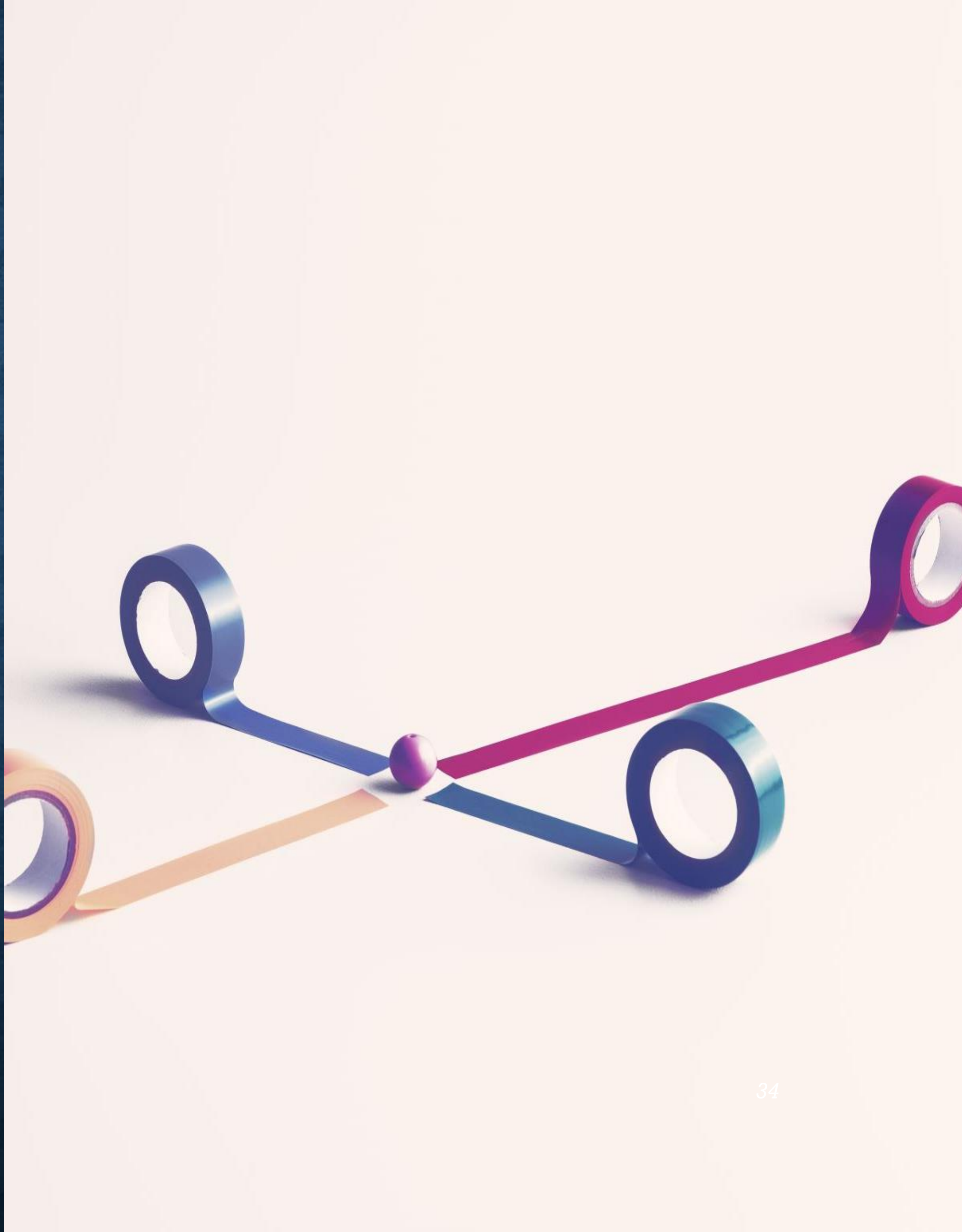
COMMON MODE FAILURES

- DCLS cannot detect potential failures that can occur at the same point in both cores since the failures do not cause any difference between their outputs.

- These failures are referred to as common mode failures, which cause false match in the DCLS system.

TEMPORAL DIVERSITY

- A common approach to this is delaying the redundant core for few cycles by inserting shift registers into the inputs.
- With a temporal diversity of even a few cycles, it is less likely that an erroneous trigger occurs at the same point of two cores.
- Note that this approach requires resynchronization of outputs from two cores before comparisons.

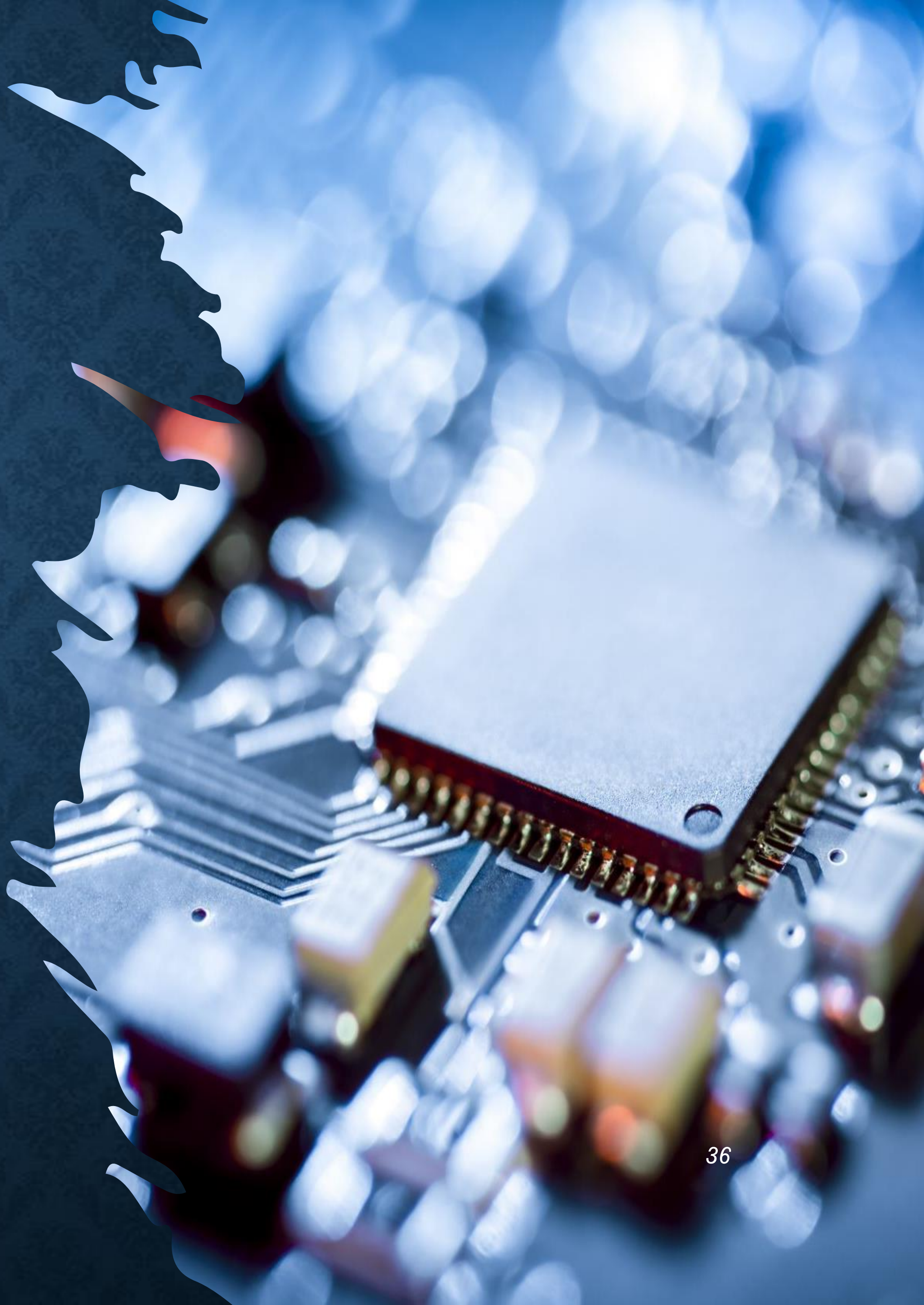


REFERENCE DESIGN

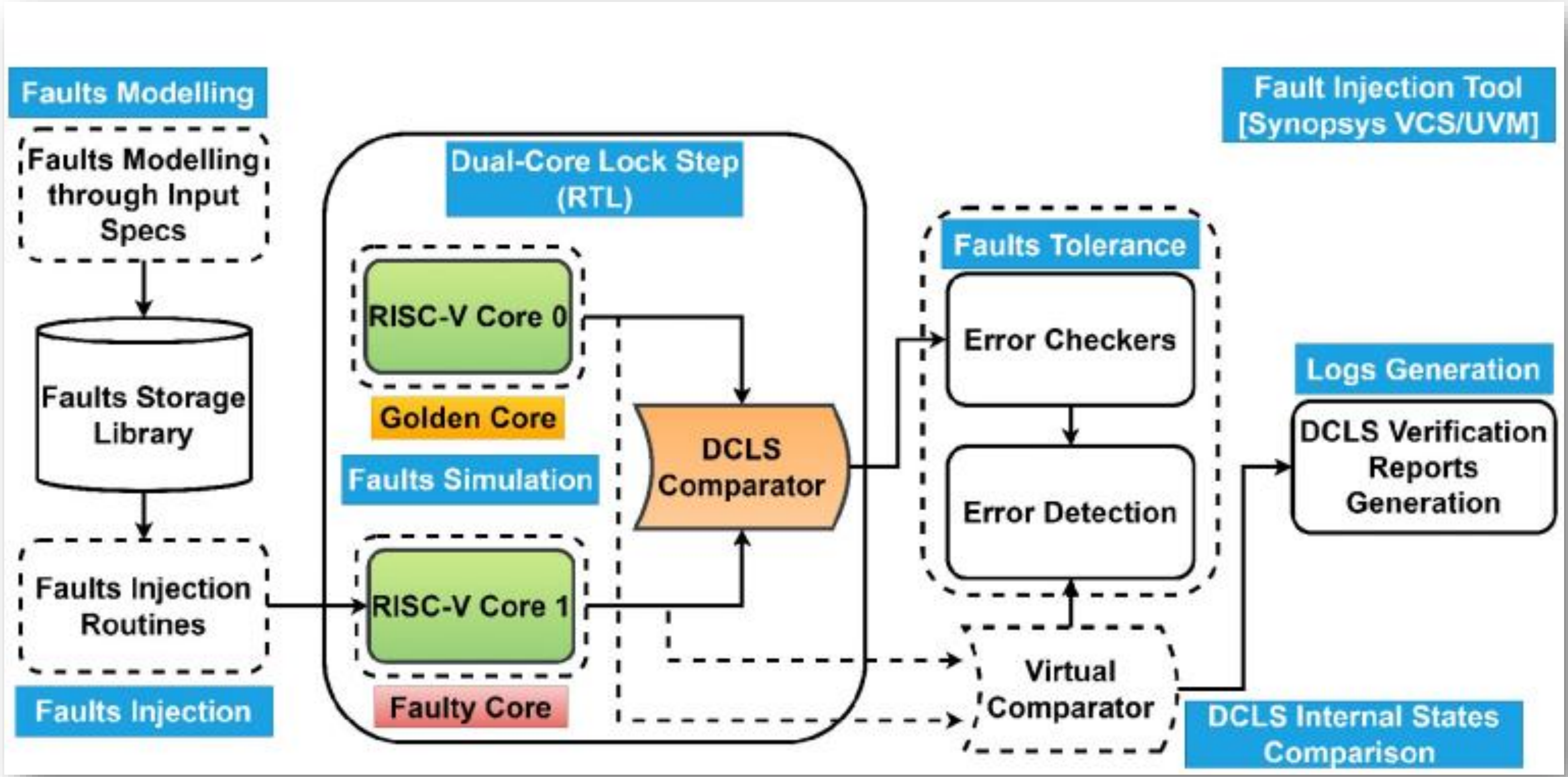
- An Open-Source SweRV-Core with Integrated DCLS Feature
- RV32-IMF architecture where “I” stands for Integer, “M” Multiplication & “F” for floating point
- GitHub - chipsalliance/Cores-VeeR-EH1: VeeR EH1 core

DIFFERENT FAULT INJECTION (FI) TECHNIQUES

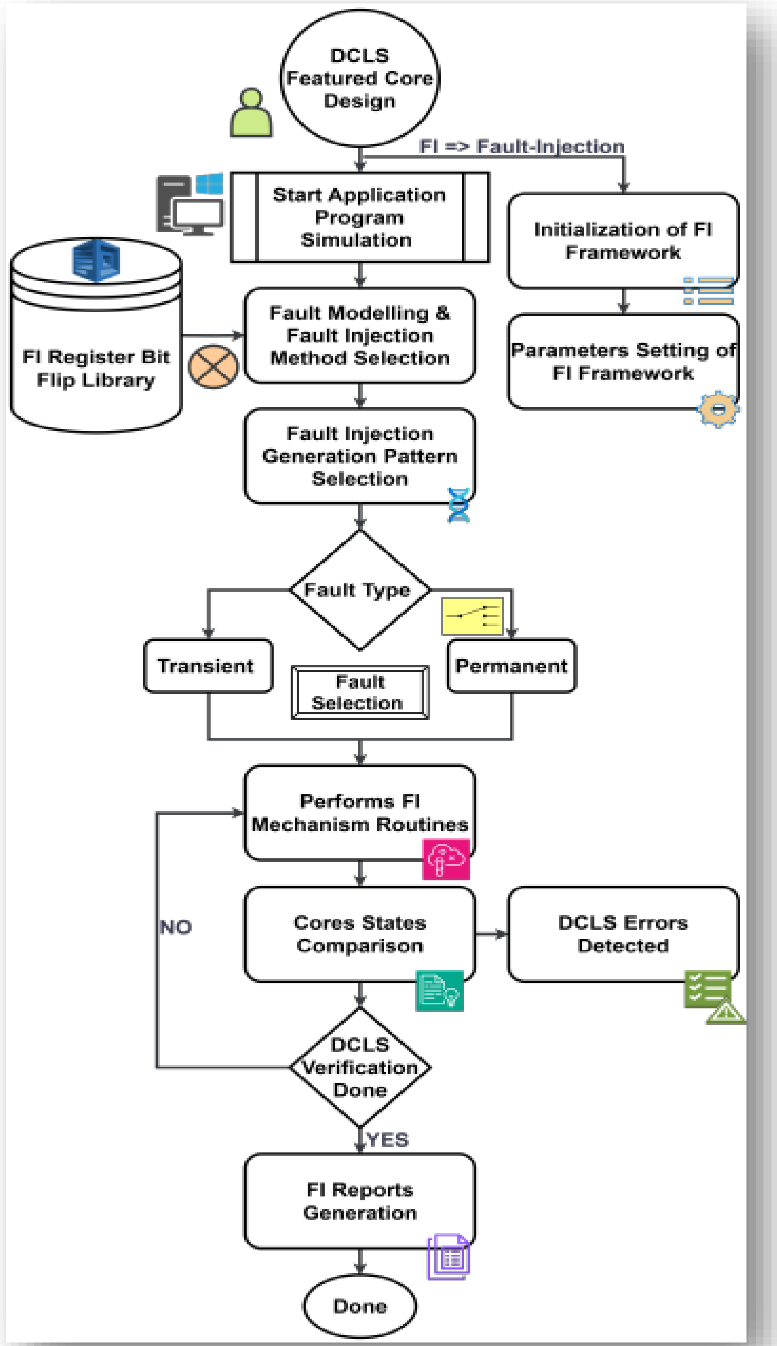
- Hardware and software-based techniques
- **This work focus on software-based FI techniques**
- Software-based techniques are categorized based on the **time of fault-injection**, i.e, **compile-time**, and **run-time**
- Code Modification/Insertion
- RTL of design under verification (DUV) is altered during run-time.



PROPOSED FI FRAMEWORK



FLOW DIAGRAM


DCLS Featured Core Design
FI => Fault-Injection
Start Application Program Simulation
Initialization of FI Framework
FI Register Bit Flip Library
Fault Modelling & Fault Injection Method Selection
Parameters Setting of FI Framework
Fault Injection Generation Pattern Selection
Fault Type
Transient
Fault Selection
Permanent
Performs FI Mechanism Routines
NO
Cores States Comparison
DCLS Errors Detected
DCLS Verification Done
YES
FI Reports Generation
Done

Faults Modeling

The process of describing and characterizing the types, locations, and behaviors of faults that might arise in SoCs is known as fault modelling.

In the proposed FI framework, faults are modelled using **UVM-macros**.

The UVM-macros allows **backdoor access to DUV internal registers.**

H**undred different DCLS internal state signals** are accessed in the fault model, for customizing or flipping their existing stored data.

The UVM macros used for faults modelling are:

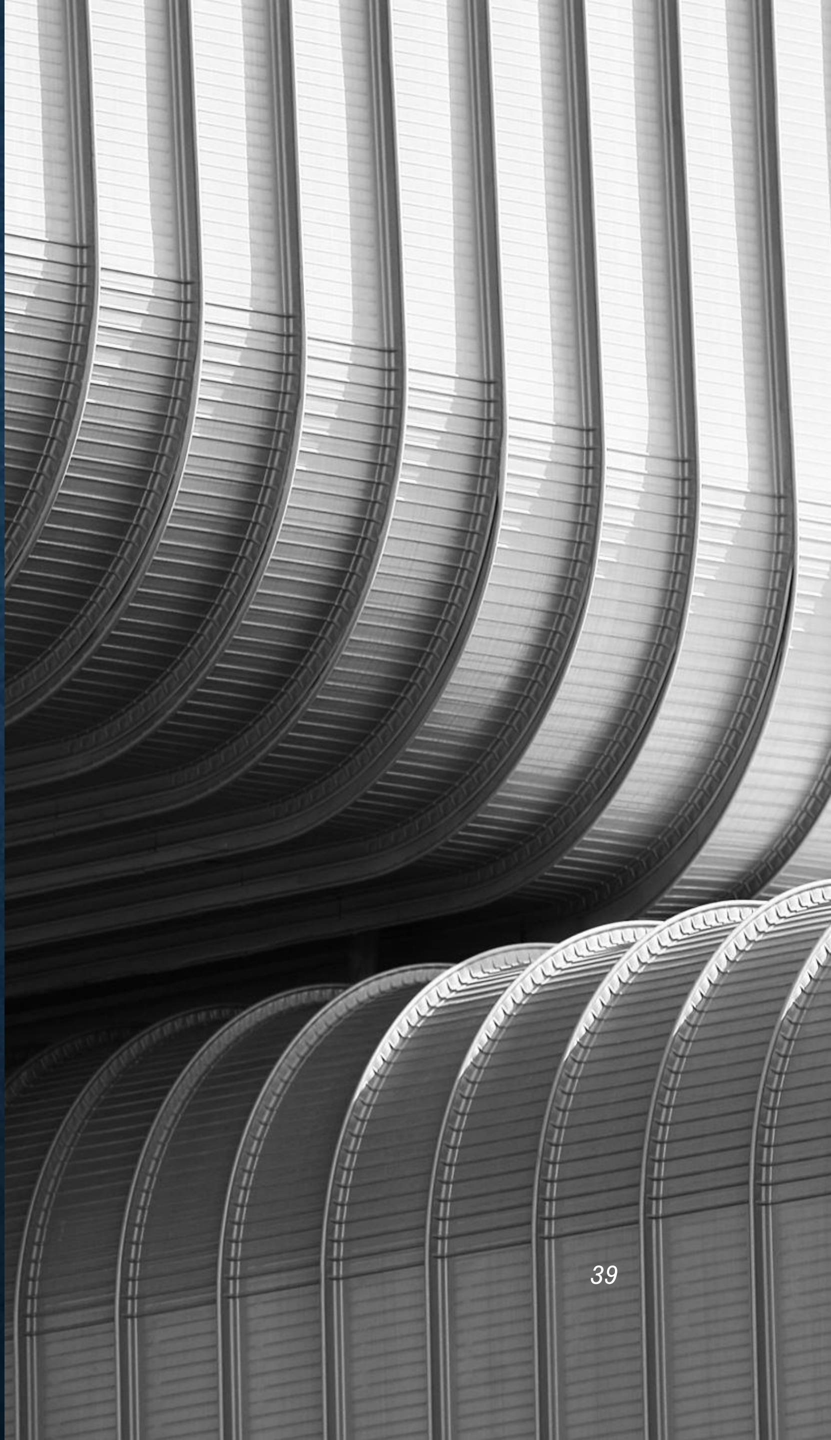
(i) **uvm-hdl-deposit**,

(ii) **uvm-hdl-force**,

(iii) **uvm-hdl-force-time**,

(iv) **uvm-hdl-release** and

(v) **uvm-hdl-read**.



USED APPLICATION CASES

- The applications used to test DCLS functionality is an open-source Test-Suite known as “**Google RISC-V DV**”.

- GitHub - chipsalliance/riscv-dv: Random instruction generator for RISC-V processor verification.

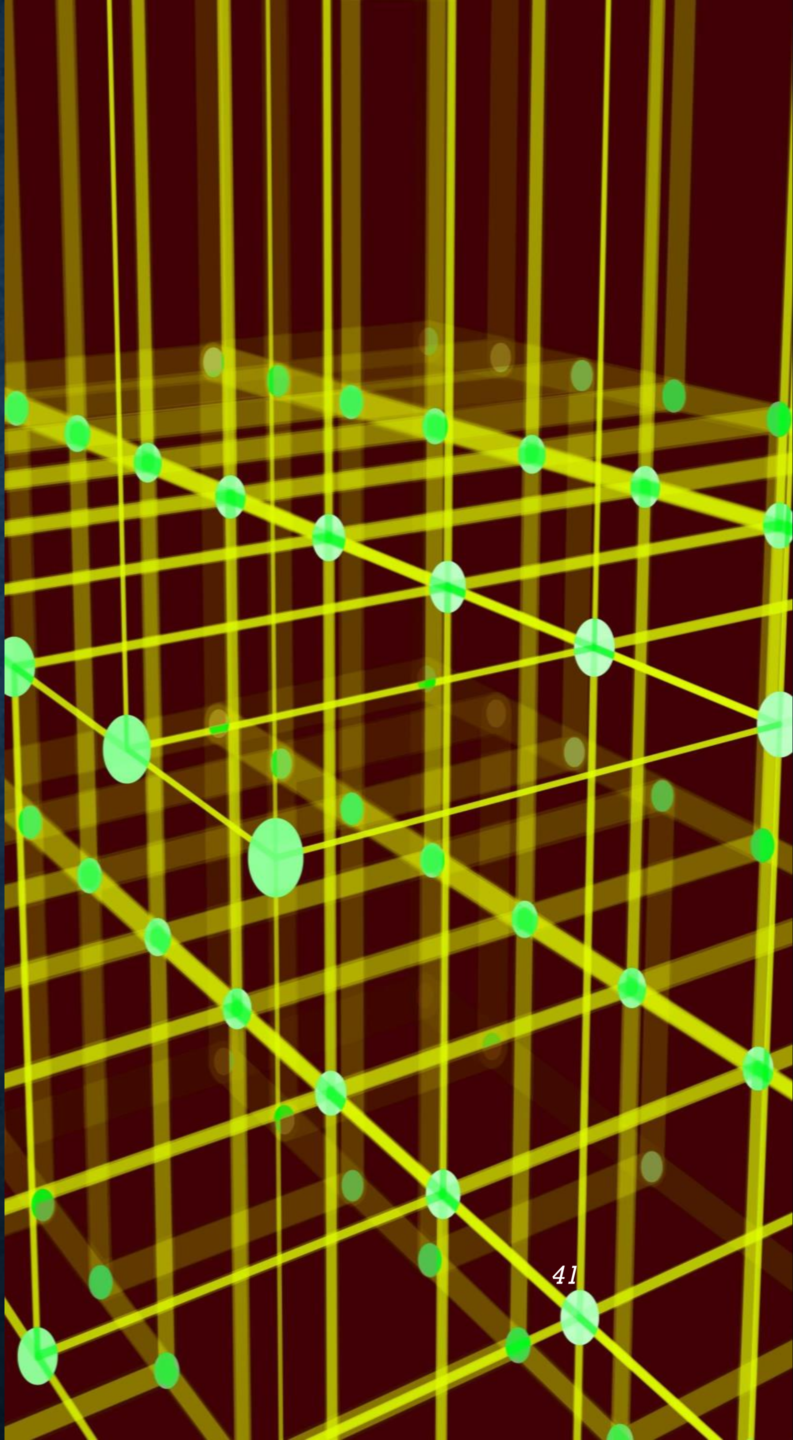
Faults Simulation

Application cases in the form of Google RISC-V DV test-suite are applied to the DUV.

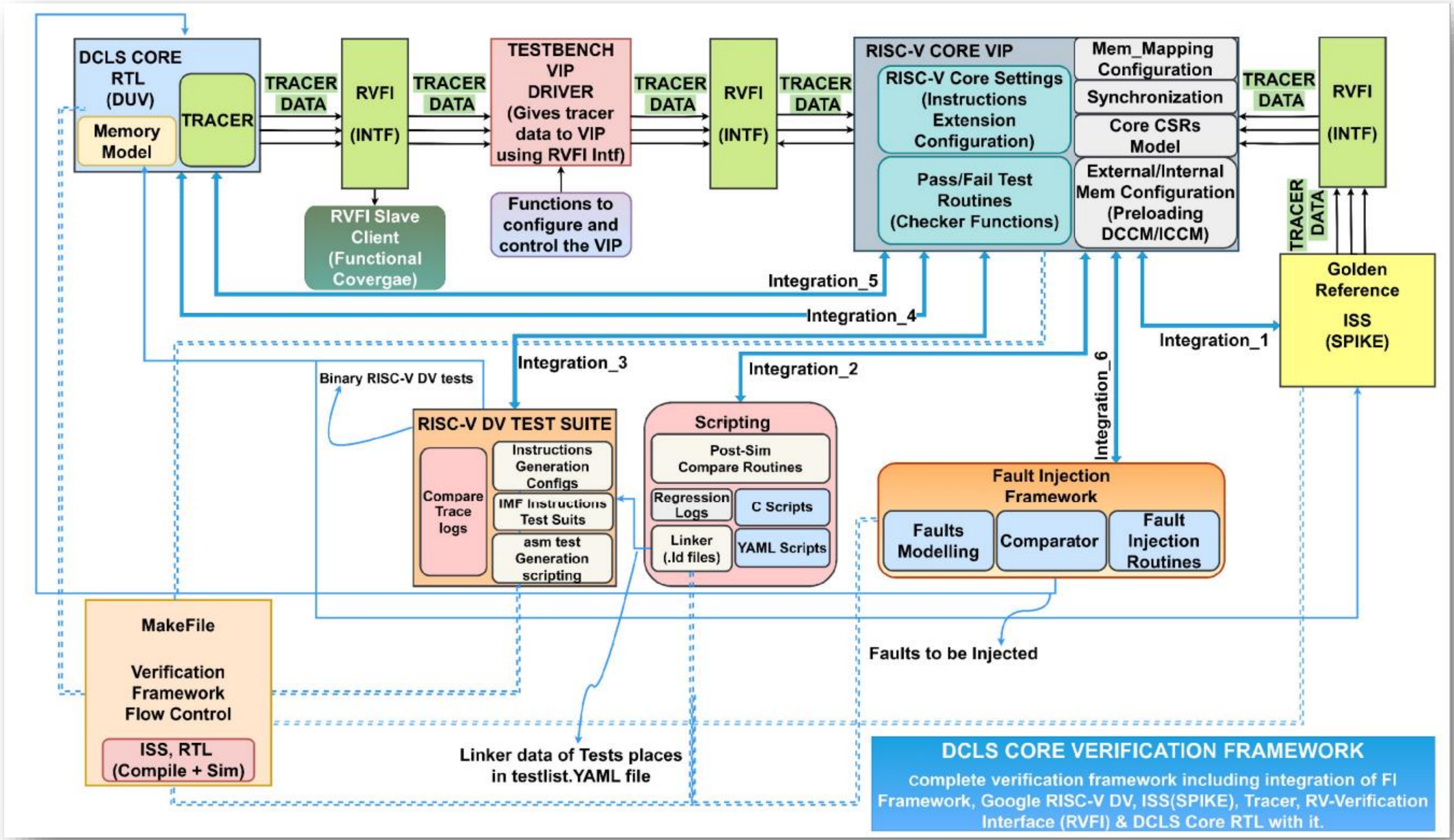
Faults are induced during each application test case simulation.

Simultaneously, the FI routines are also applied to the DUV.

The resulting fault's latency, propagation and severity levels are then analyzed using simulation waveforms data.



UVM-BASED TESTBENCH

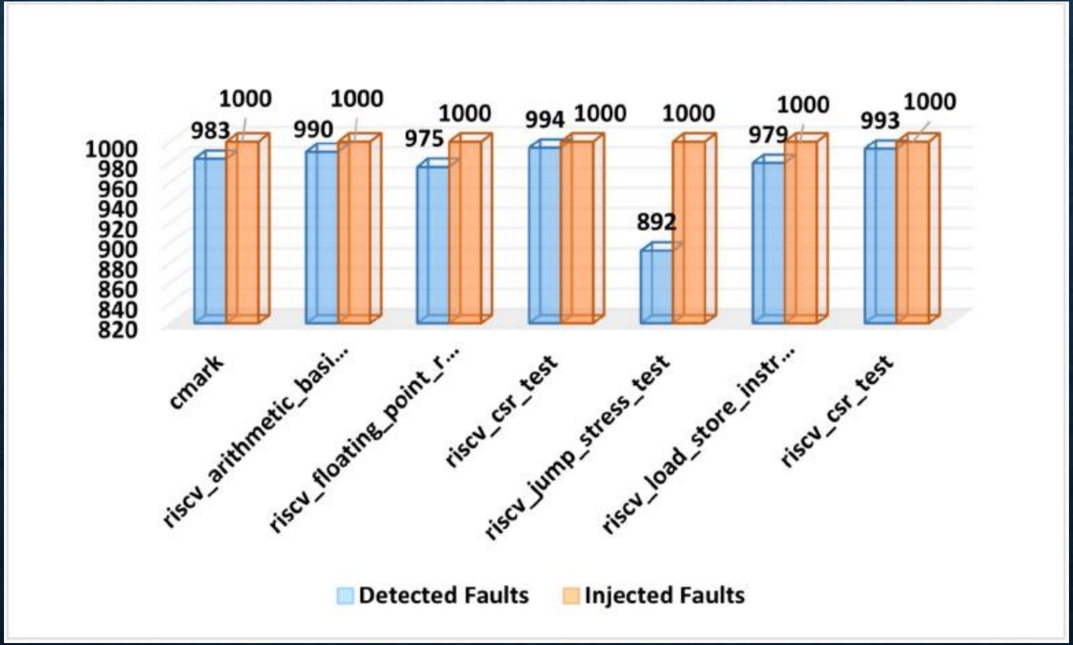

DCLS CORE RTL (DUV)
Memory Model
TRACER
TRACER DATA
RVFI (INTF)
TRACER DATA
TESTBENCH VIP DRIVER (Gives tracer data to VIP using RVFI Intf)
TRACER DATA
RVFI (INTF)
TRACER DATA
RISC-V CORE VIP
RISC-V Core Settings (Instructions Extension Configuration)
Pass/Fail Test Routines (Checker Functions)
Mem_Mapping Configuration
Synchronization
Core CSRs Model
External/Internal Mem Configuration (Preloading DCCM/ICCM)
TRACER DATA
RVFI (INTF)
TRACER DATA
RVFI Slave Client (Functional Covergae)
Functions to configure and control the VIP
Integration_5
Integration_4
Integration_3
Integration_2
Integration_6
Integration_1
Golden Reference ISS (SPIKE)
Binary RISC-V DV tests
RISC-V DV TEST SUITE
Compare Trace logs
Instructions Generation Configs
IMF Instructions Test Suits
asm test Generation scripting
Scripting
Post-Sim Compare Routines
Regression Logs
C Scripts
Linker (.ld files)
YAML Scripts
Fault Injection Framework
Faults Modelling
Comparator
Fault Injection Routines
Faults to be Injected
MakeFile
Verification Framework Flow Control
ISS, RTL (Compile + Sim)
Linker data of Tests places in testlist.YAML file

DCLS CORE VERIFICATION FRAMEWORK

complete verification framework including integration of FI Framework, Google RISC-V DV, ISS(SPIKE), Tracer, RV-Verification Interface (RVFI) & DCLS Core RTL with it.

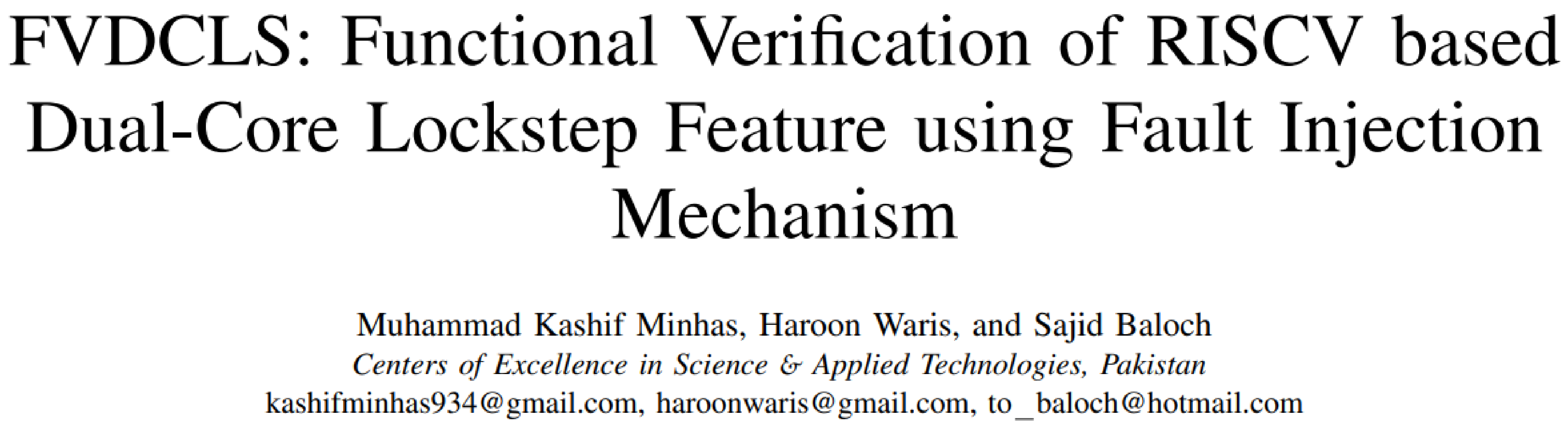
RESULTS

- Approximately 20,000 errors are injected by FI campaigns.
- The DCLS feature successfully detects **98.7%** of errors from the overall fault injection routines.
- The application cases and their number of detected versus injected faults are shown below



FUTURE DIRECTIONS

- Add new Fault-Injection scenarios
- Use advanced AI features to model the faults
- Test the enhanced fault-injection framework with different cores architectures



FVDCLS: Functional Verification of RISCV based Dual-Core Lockstep Feature using Fault Injection Mechanism

Muhammad Kashif Minhas, Haroon Waris, and Sajid Baloch
Centers of Excellence in Science & Applied Technologies, Pakistan
kashifminhas934@gmail.com, haroonwaris@gmail.com, to_baloch@hotmail.com

FVDCLS RESEARCH PAPER

Research paper accepted in VLSI-SoC Conference Morocco, 2024

RISC-V CORE VERIFICATION FLOW/DEMO

INTRODUCTION TO RISC-V ISA

- RISC-V (pronounced "risk-five”): An open-source implementation of a reduced instruction set computing (RISC) based instruction set architecture (ISA)

- Permitting any person or group to construct compatible computers

- Originated in 2010 by researchers at UC Berkeley

- RISC-V ISA includes: A small base integer ISA, usable by itself as a base for customized accelerators or for educational purposes, and
  - ✓ Optional standard extensions, to support general-purpose software development
  - ✓ Optional customer extensions

INTRODUCTION TO RISC-V ISA

- *ISA support is given by RV + word-width + extensions supported permitting any person or group to construct compatible computers*
  - ✓ *RV32I means 32-bit RISC-V with support for the I(Integer) instruction set*

- *A mandatory Base integer ISA*

  I: Integer instructions

- *Standard Extensions*

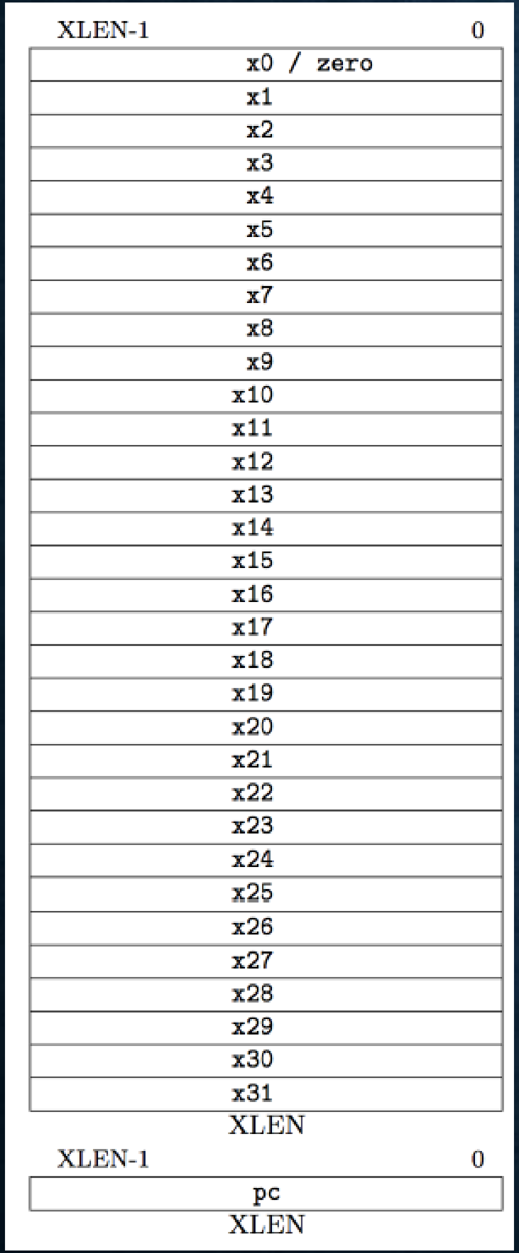
  M: Integer Multiplication and Division

  A: Atomic Instructions

  F: Single-Precision Floating-Point

  D: Double-Precision Floating-Point

  C: Compressed Instructions (16 bit)



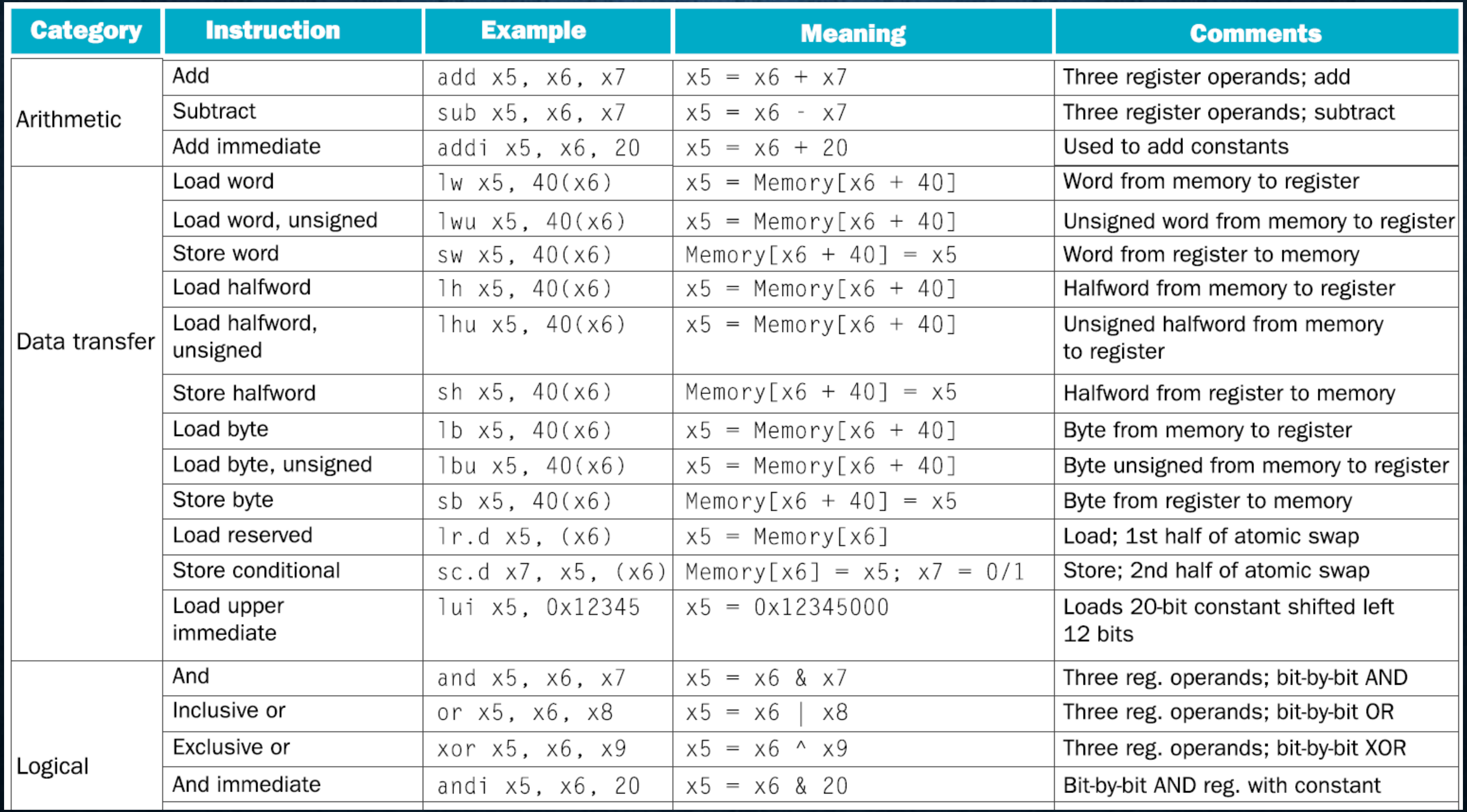
RV32/64 PROCESSOR REGISTER SET

- *32 32/64-bit integer registers (x0-x31)*
    - *x0 always contains a 0*

- *32 floating-point (FP) registers (f0-f31)*
    - *Each can contain a single- or double-precision FP value (32-bit or 64-bit IEEE FP)*

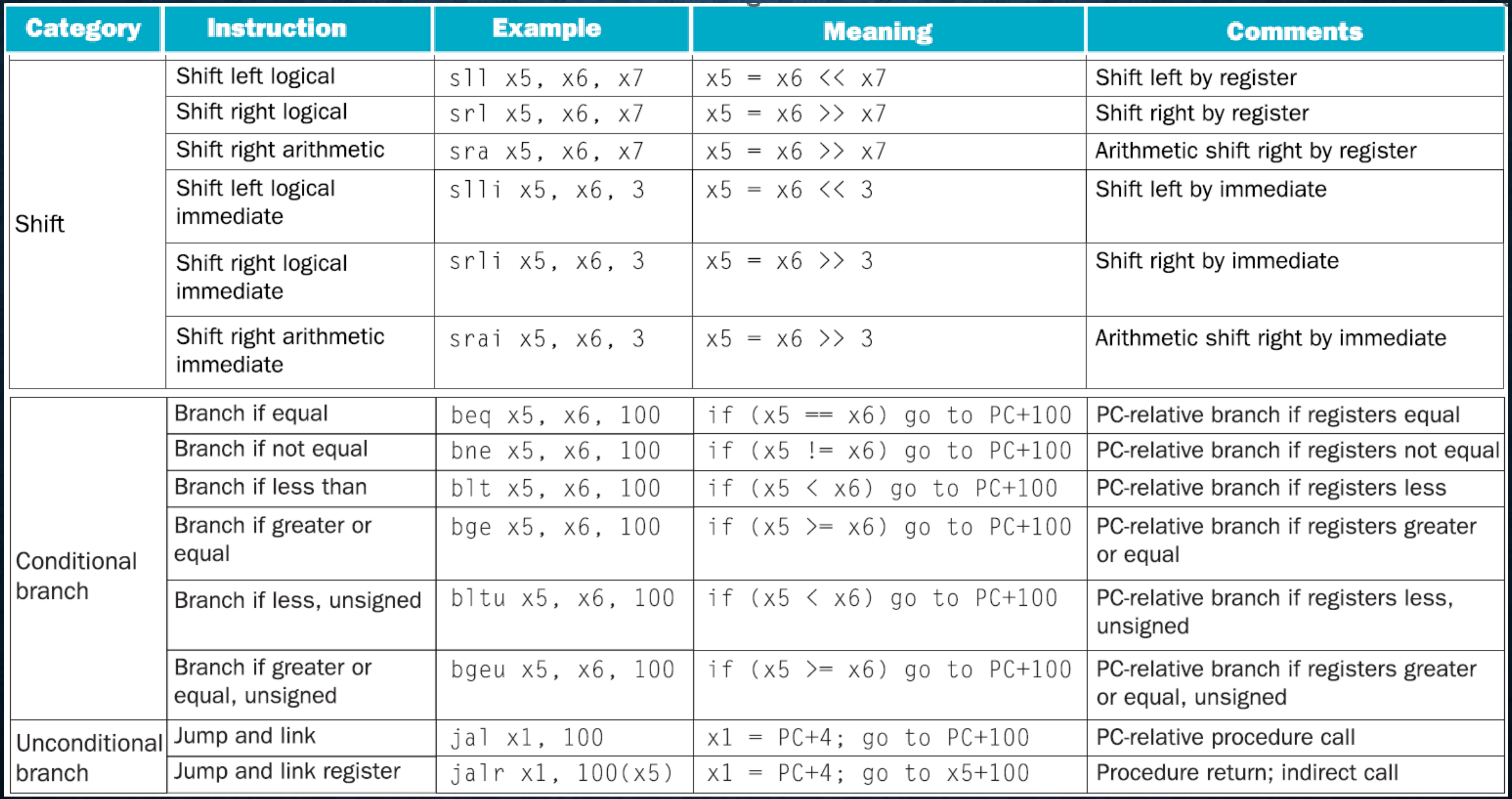
- *Program counter (pc) which holds the address of the current instruction*

DIFFERENT RISC-V INSTRUCTIONS



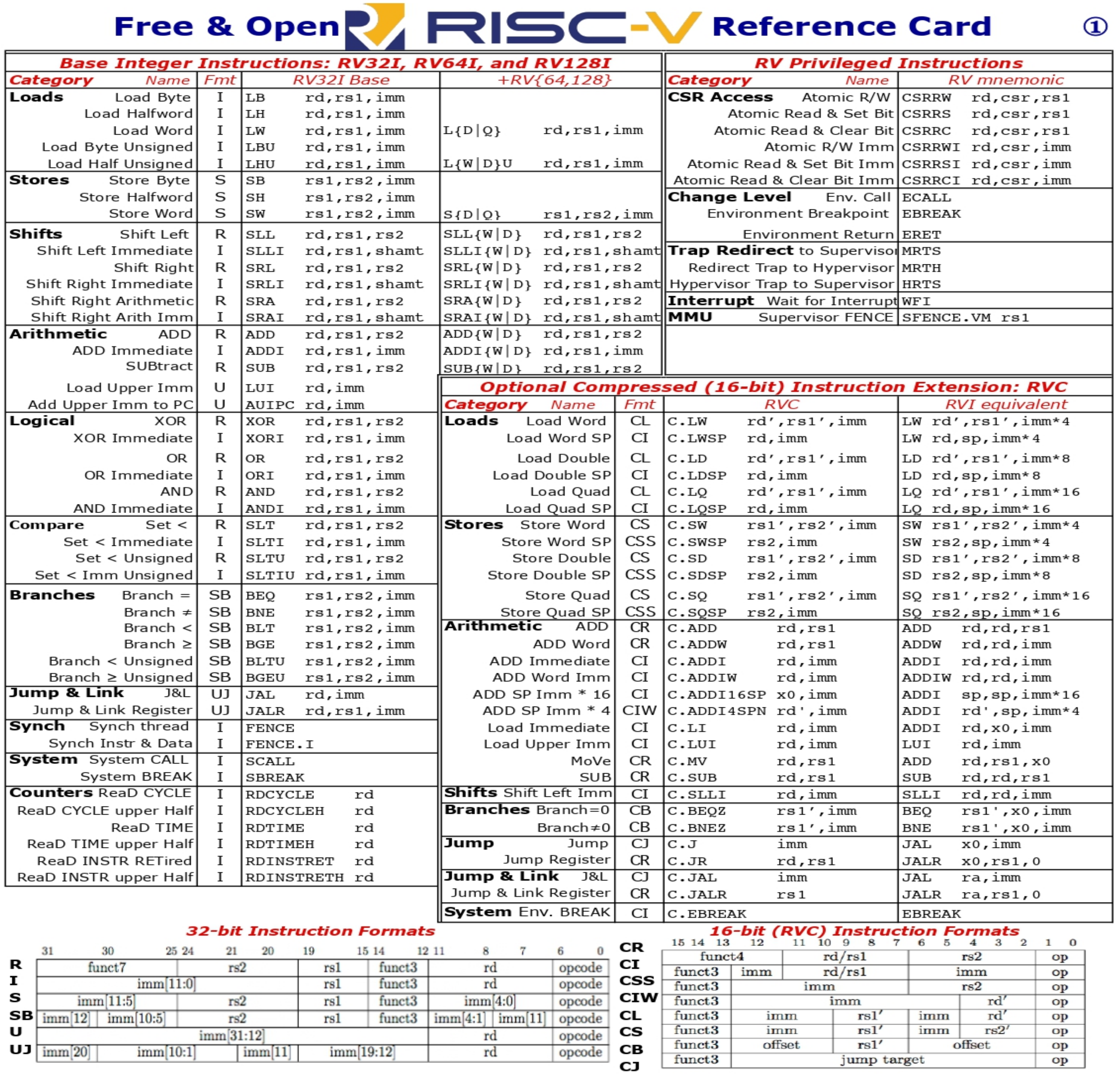
Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	`add x5, x6, x7`	`x5 = x6 + x7`	Three register operands; add
	Subtract	`sub x5, x6, x7`	`x5 = x6 - x7`	Three register operands; subtract
	Add immediate	`addi x5, x6, 20`	`x5 = x6 + 20`	Used to add constants
Data transfer	Load word	`lw x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Word from memory to register
	Load word, unsigned	`lwu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned word from memory to register
	Store word	`sw x5, 40(x6)`	`Memory[x6 + 40] = x5`	Word from register to memory
	Load halfword	`lh x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Halfword from memory to register
	Load halfword, unsigned	`lhu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned halfword from memory to register
	Store halfword	`sh x5, 40(x6)`	`Memory[x6 + 40] = x5`	Halfword from register to memory
	Load byte	`lb x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte from memory to register
	Load byte, unsigned	`lbu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte unsigned from memory to register
	Store byte	`sb x5, 40(x6)`	`Memory[x6 + 40] = x5`	Byte from register to memory
	Load reserved	`lr.d x5, (x6)`	`x5 = Memory[x6]`	Load; 1st half of atomic swap
	Store conditional	`sc.d x7, x5, (x6)`	`Memory[x6] = x5; x7 = 0/1`	Store; 2nd half of atomic swap
	Load upper immediate	`lui x5, 0x12345`	`x5 = 0x12345000`	Loads 20-bit constant shifted left 12 bits
Logical	And	`and x5, x6, x7`	`x5 = x6 & x7`	Three reg. operands; bit-by-bit AND
	Inclusive or	`or x5, x6, x8`	`x5 = x6 | x8`	Three reg. operands; bit-by-bit OR
	Exclusive or	`xor x5, x6, x9`	`x5 = x6 ^ x9`	Three reg. operands; bit-by-bit XOR
	And immediate	`andi x5, x6, 20`	`x5 = x6 & 20`	Bit-by-bit AND reg. with constant

DIFFERENT RISC-V INSTRUCTIONS



Category	Instruction	Example	Meaning	Comments
Shift	Shift left logical	`sll x5, x6, x7`	`x5 = x6 << x7`	Shift left by register
	Shift right logical	`srl x5, x6, x7`	`x5 = x6 >> x7`	Shift right by register
	Shift right arithmetic	`sra x5, x6, x7`	`x5 = x6 >> x7`	Arithmetic shift right by register
	Shift left logical immediate	`slli x5, x6, 3`	`x5 = x6 << 3`	Shift left by immediate
	Shift right logical immediate	`srli x5, x6, 3`	`x5 = x6 >> 3`	Shift right by immediate
	Shift right arithmetic immediate	`srai x5, x6, 3`	`x5 = x6 >> 3`	Arithmetic shift right by immediate
Conditional branch	Branch if equal	`beq x5, x6, 100`	`if (x5 == x6) go to PC+100`	PC-relative branch if registers equal
	Branch if not equal	`bne x5, x6, 100`	`if (x5 != x6) go to PC+100`	PC-relative branch if registers not equal
	Branch if less than	`blt x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less
	Branch if greater or equal	`bge x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal
	Branch if less, unsigned	`bltu x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less, unsigned
	Branch if greater or equal, unsigned	`bgeu x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal, unsigned
Unconditional branch	Jump and link	`jal x1, 100`	`x1 = PC+4; go to PC+100`	PC-relative procedure call
	Jump and link register	`jalr x1, 100(x5)`	`x1 = PC+4; go to x5+100`	Procedure return; indirect call

RISC-V GREEN CARD



Free & Open RISC-V Reference Card ①

Base Integer Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base	+RV{64,128}
Loads	Load Byte	I	LB rd,rs1,imm	
	Load Halfword	I	LH rd,rs1,imm	
	Load Word	I	LW rd,rs1,imm	L{D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm	
	Load Half Unsigned	I	LHU rd,rs1,imm	L{W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm	
	Store Halfword	S	SH rs1,rs2,imm	
	Store Word	S	SW rs1,rs2,imm	S{D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL rd,rs1,rs2	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI rd,rs1,shamt	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL rd,rs1,rs2	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI rd,rs1,shamt	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI rd,rs1,shamt	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUB{W|D} rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Logical	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Compare	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Branches	Branch =	SB	BEQ rs1,rs2,imm	
	Branch ≠	SB	BNE rs1,rs2,imm	
	Branch <	SB	BLT rs1,rs2,imm	
	Branch ≥	SB	BGE rs1,rs2,imm	
	Branch < Unsigned	SB	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm	
Jump & Link	J&L	UJ	JAL rd,imm	
	Jump & Link Register	UJ	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
System	System CALL	I	SCALL	
	System BREAK	I	SBREAK	
Counters	ReaD CYCLE	I	RDCYCLE rd	
	ReaD CYCLE upper Half	I	RDCYCLEH rd	
	ReaD TIME	I	RDTIME rd	
	ReaD TIME upper Half	I	RDTIMEH rd	
	ReaD INSTR RETired	I	RDINSTRET rd	
	ReaD INSTR upper Half	I	RDINSTRETH rd	

RV Privileged Instructions

Category	Name	RV mnemonic
CSR Access	Atomic R/W	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	CSRRC rd,csr,rs1
	Atomic R/W Imm	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	CSRRCI rd,csr,imm
Change Level	Env. Call	ECALL
	Environment Breakpoint	EBREAK
	Environment Return	ERET
Trap Redirect	to Supervisor	MRTS
	Redirect Trap to Hypervisor	MRTH
	Hypervisor Trap to Supervisor	HRTS
Interrupt	Wait for Interrupt	WFI
MMU	Supervisor FENCE	SFENCE.VM rs1

Optional Compressed (16-bit) Instruction Extension: RVC

Category	Name	Fmt	RVC	RVI equivalent
Loads	Load Word	CL	C.LW rd′,rs1′,imm	LW rd′,rs1′,imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD rd′,rs1′,imm	LD rd′,rs1′,imm*8
	Load Double SP	CI	C.LDSP rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ rd′,rs1′,imm	LQ rd′,rs1′,imm*16
	Load Quad SP	CI	C.LQSP rd,imm	LQ rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1′,rs2′,imm	SW rs1′,rs2′,imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Store Double	CS	C.SD rs1′,rs2′,imm	SD rs1′,rs2′,imm*8
	Store Double SP	CSS	C.SDSP rs2,imm	SD rs2,sp,imm*8
	Store Quad	CS	C.SQ rs1′,rs2′,imm	SQ rs1′,rs2′,imm*16
	Store Quad SP	CSS	C.SQSP rs2,imm	SQ rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Word	CR	C.ADDW rd,rs1	ADDW rd,rd,imm
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm	ADDIW rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm	ADDI rd',sp,imm*4
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1′,imm	BEQ rs1',x0,imm
	Branch≠0	CB	C.BNEZ rs1′,imm	BNE rs1',x0,imm
Jump	Jump	CJ	C.J imm	JAL x0,imm
	Jump Register	CR	C.JR rd,rs1	JALR x0,rs1,0
Jump & Link	J&L	CJ	C.JAL imm	JAL ra,imm
	Jump & Link Register	CR	C.JALR rs1	JALR ra,rs1,0
System	Env. BREAK	CI	C.EBREAK	EBREAK

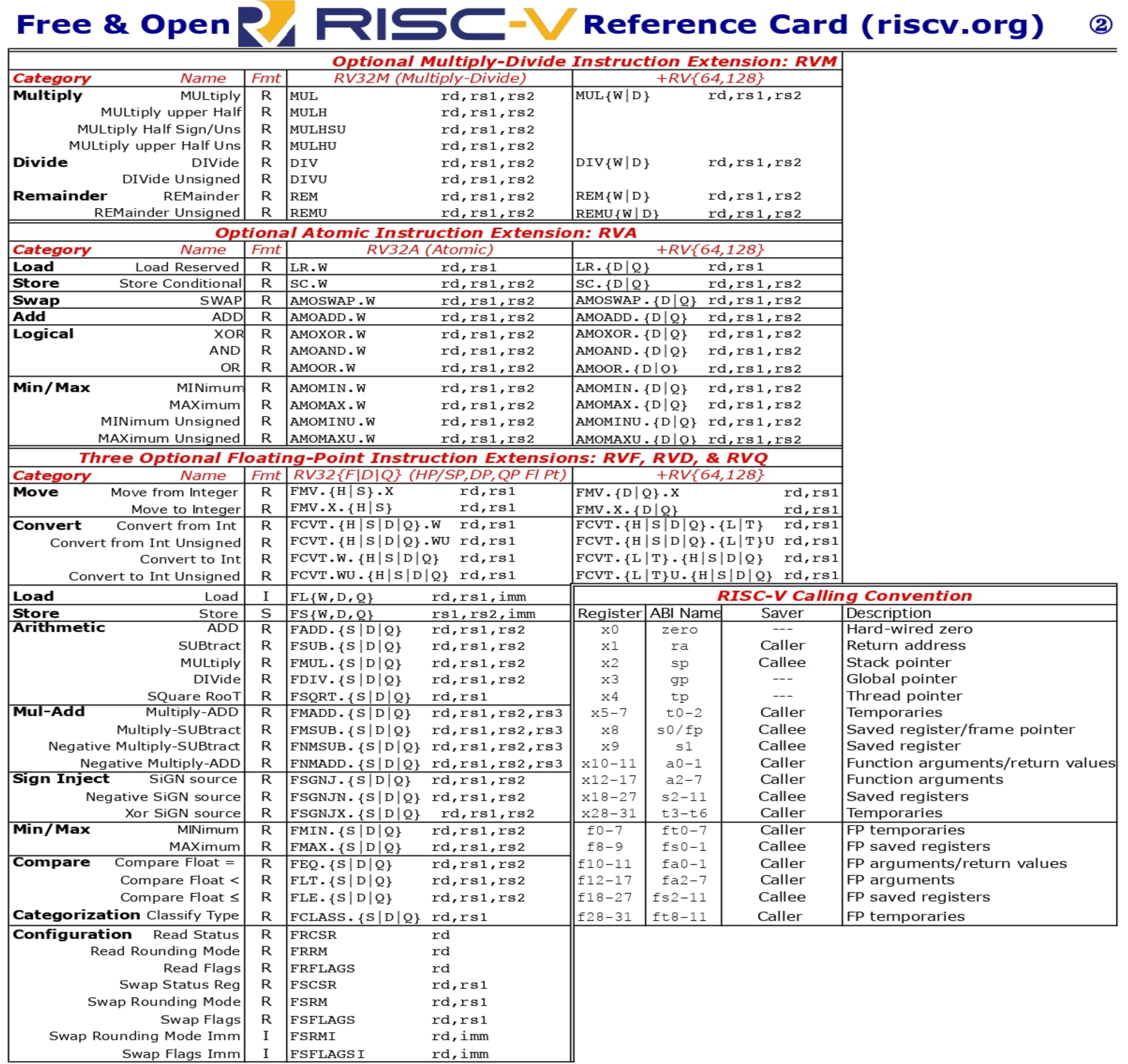
32-bit Instruction Formats

	31	30 … 25	24 … 21	20	19 … 15	14 … 12	11 … 8	7	6 … 0
R	funct7		rs2		rs1	funct3	rd		opcode
I	imm[11:0]				rs1	funct3	rd		opcode
S	imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode
SB	imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode
U	imm[31:12]						rd		opcode
UJ	imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode

16-bit (RVC) Instruction Formats

	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	funct4		rd/rs1		rs2		op
CI	funct3	imm	rd/rs1		imm		op
CSS	funct3	imm			rs2		op
CIW	funct3	imm				rd′	op
CL	funct3	imm		rs1′	imm	rd′	op
CS	funct3	imm		rs1′	imm	rs2′	op
CB	funct3	offset		rs1′	offset		op
CJ	funct3	jump target					op

RISC-V GREEN CARD



Free & Open RISC-V Reference Card (riscv.org) ②

Optional Multiply-Divide Instruction Extension: RVM

Category	Name	Fmt	RV32M (Multiply-Divide)		+RV{64,128}	
Multiply	MULtiply	R	MUL	rd,rs1,rs2	MUL{W|D}	rd,rs1,rs2
	MULtiply upper Half	R	MULH	rd,rs1,rs2		
	MULtiply Half Sign/Uns	R	MULHSU	rd,rs1,rs2		
	MULtiply upper Half Uns	R	MULHU	rd,rs1,rs2		
Divide	DIVide	R	DIV	rd,rs1,rs2	DIV{W|D}	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2		
Remainder	REMainder	R	REM	rd,rs1,rs2	REM{W|D}	rd,rs1,rs2
	REMainder Unsigned	R	REMU	rd,rs1,rs2	REMU{W|D}	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV32A (Atomic)		+RV{64,128}	
Load	Load Reserved	R	LR.W	rd,rs1	LR.{D|Q}	rd,rs1
Store	Store Conditional	R	SC.W	rd,rs1,rs2	SC.{D|Q}	rd,rs1,rs2
Swap	SWAP	R	AMOSWAP.W	rd,rs1,rs2	AMOSWAP.{D|Q}	rd,rs1,rs2
Add	ADD	R	AMOADD.W	rd,rs1,rs2	AMOADD.{D|Q}	rd,rs1,rs2
Logical	XOR	R	AMOXOR.W	rd,rs1,rs2	AMOXOR.{D|Q}	rd,rs1,rs2
	AND	R	AMOAND.W	rd,rs1,rs2	AMOAND.{D|Q}	rd,rs1,rs2
	OR	R	AMOOR.W	rd,rs1,rs2	AMOOR.{D|Q}	rd,rs1,rs2
Min/Max	MINimum	R	AMOMIN.W	rd,rs1,rs2	AMOMIN.{D|Q}	rd,rs1,rs2
	MAXimum	R	AMOMAX.W	rd,rs1,rs2	AMOMAX.{D|Q}	rd,rs1,rs2
	MINimum Unsigned	R	AMOMINU.W	rd,rs1,rs2	AMOMINU.{D|Q}	rd,rs1,rs2
	MAXimum Unsigned	R	AMOMAXU.W	rd,rs1,rs2	AMOMAXU.{D|Q}	rd,rs1,rs2

Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVQ

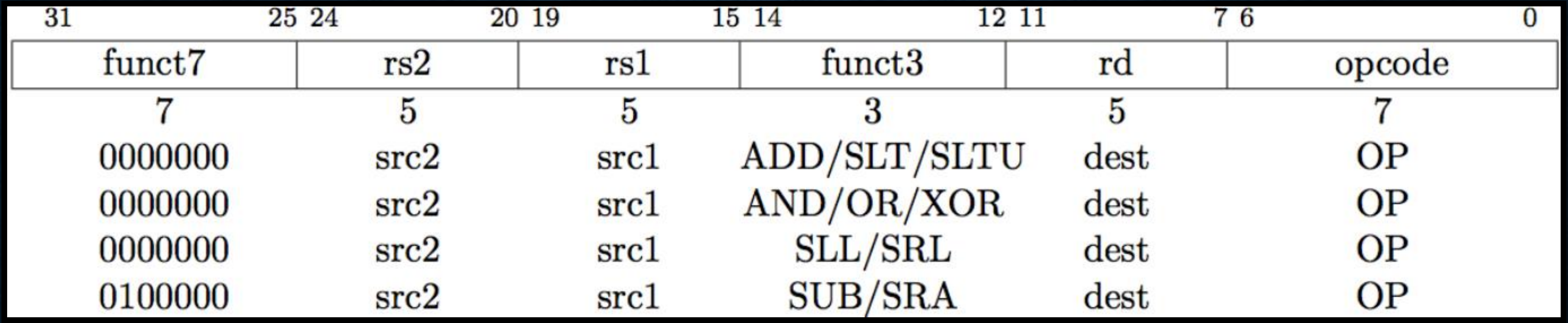
Category	Name	Fmt	RV32{F|D|Q} (HP/SP,DP,QP Fl Pt)		+RV{64,128}	
Move	Move from Integer	R	FMV.{H|S}.X	rd,rs1	FMV.{D|Q}.X	rd,rs1
	Move to Integer	R	FMV.X.{H|S}	rd,rs1	FMV.X.{D|Q}	rd,rs1
Convert	Convert from Int	R	FCVT.{H|S|D|Q}.W	rd,rs1	FCVT.{H|S|D|Q}.{L|T}	rd,rs1
	Convert from Int Unsigned	R	FCVT.{H|S|D|Q}.WU	rd,rs1	FCVT.{H|S|D|Q}.{L|T}U	rd,rs1
	Convert to Int	R	FCVT.W.{H|S|D|Q}	rd,rs1	FCVT.{L|T}.{H|S|D|Q}	rd,rs1
	Convert to Int Unsigned	R	FCVT.WU.{H|S|D|Q}	rd,rs1	FCVT.{L|T}U.{H|S|D|Q}	rd,rs1
Load	Load	I	FL{W,D,Q}	rd,rs1,imm		
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm		
Arithmetic	ADD	R	FADD.{S|D|Q}	rd,rs1,rs2		
	SUBtract	R	FSUB.{S|D|Q}	rd,rs1,rs2		
	MULtiply	R	FMUL.{S|D|Q}	rd,rs1,rs2		
	DIVide	R	FDIV.{S|D|Q}	rd,rs1,rs2		
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1		
Mul-Add	Multiply-ADD	R	FMADD.{S|D|Q}	rd,rs1,rs2,rs3		
	Multiply-SUBtract	R	FMSUB.{S|D|Q}	rd,rs1,rs2,rs3		
	Negative Multiply-SUBtract	R	FNMSUB.{S|D|Q}	rd,rs1,rs2,rs3		
	Negative Multiply-ADD	R	FNMADD.{S|D|Q}	rd,rs1,rs2,rs3		
Sign Inject	SiGN source	R	FSGNJ.{S|D|Q}	rd,rs1,rs2		
	Negative SiGN source	R	FSGNJN.{S|D|Q}	rd,rs1,rs2		
	Xor SiGN source	R	FSGNJX.{S|D|Q}	rd,rs1,rs2		
Min/Max	MINimum	R	FMIN.{S|D|Q}	rd,rs1,rs2		
	MAXimum	R	FMAX.{S|D|Q}	rd,rs1,rs2		
Compare	Compare Float =	R	FEQ.{S|D|Q}	rd,rs1,rs2		
	Compare Float <	R	FLT.{S|D|Q}	rd,rs1,rs2		
	Compare Float ≤	R	FLE.{S|D|Q}	rd,rs1,rs2		
Categorization	Classify Type	R	FCLASS.{S|D|Q}	rd,rs1		
Configuration	Read Status	R	FRCSR	rd		
	Read Rounding Mode	R	FRRM	rd		
	Read Flags	R	FRFLAGS	rd		
	Swap Status Reg	R	FSCSR	rd,rs1		
	Swap Rounding Mode	R	FSRM	rd,rs1		
	Swap Flags	R	FSFLAGS	rd,rs1		
	Swap Rounding Mode Imm	I	FSRMI	rd,imm		
	Swap Flags Imm	I	FSFLAGSI	rd,imm		

RISC-V Calling Convention

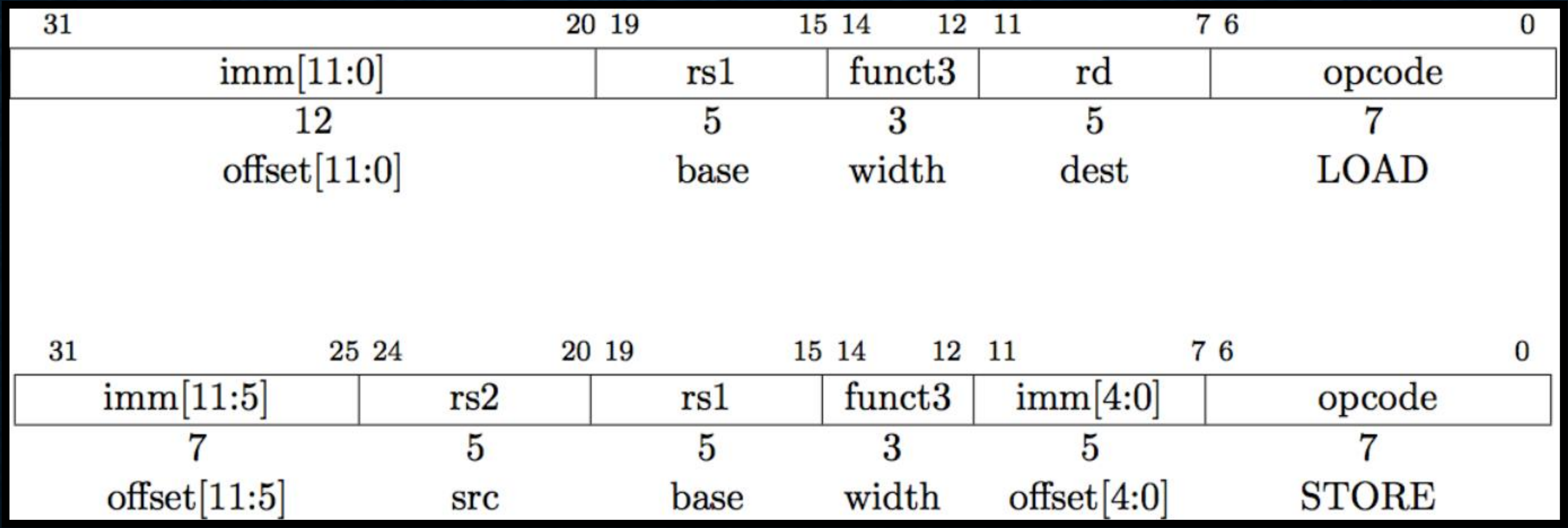
Register	ABI Name	Saver	Description
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5–7	t0–2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10–11	a0–1	Caller	Function arguments/return values
x12–17	a2–7	Caller	Function arguments
x18–27	s2–11	Callee	Saved registers
x28–31	t3–t6	Caller	Temporaries
f0–7	ft0–7	Caller	FP temporaries
f8–9	fs0–1	Callee	FP saved registers
f10–11	fa0–1	Caller	FP arguments/return values
f12–17	fa2–7	Caller	FP arguments
f18–27	fs2–11	Callee	FP saved registers
f28–31	ft8–11	Caller	FP temporaries

RISC-V INSTRUCTION FORMATS

- Specification from RISC-V website
    - ✓ https://riscv.org/specifications/



31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

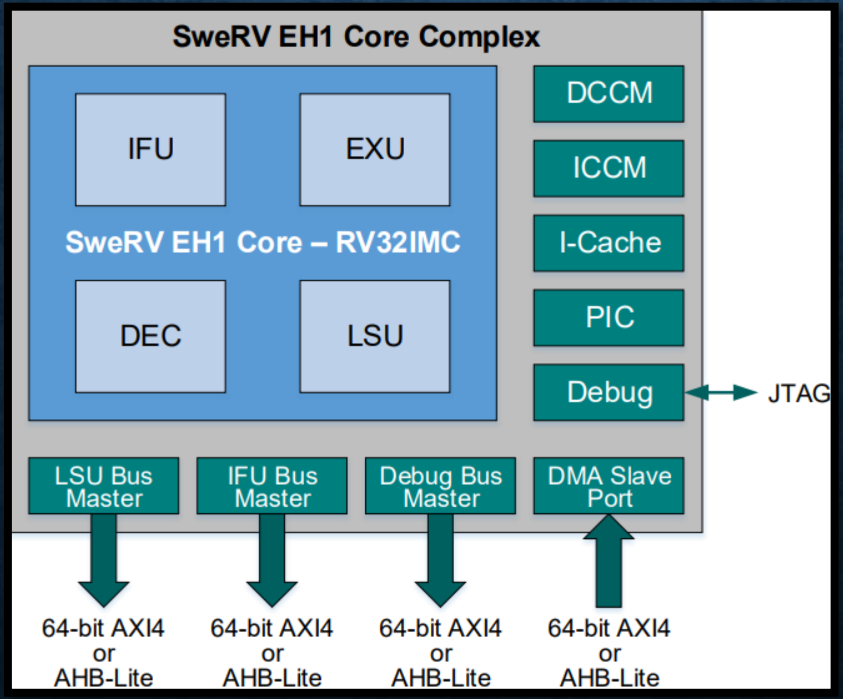


31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	width	offset[4:0]	STORE

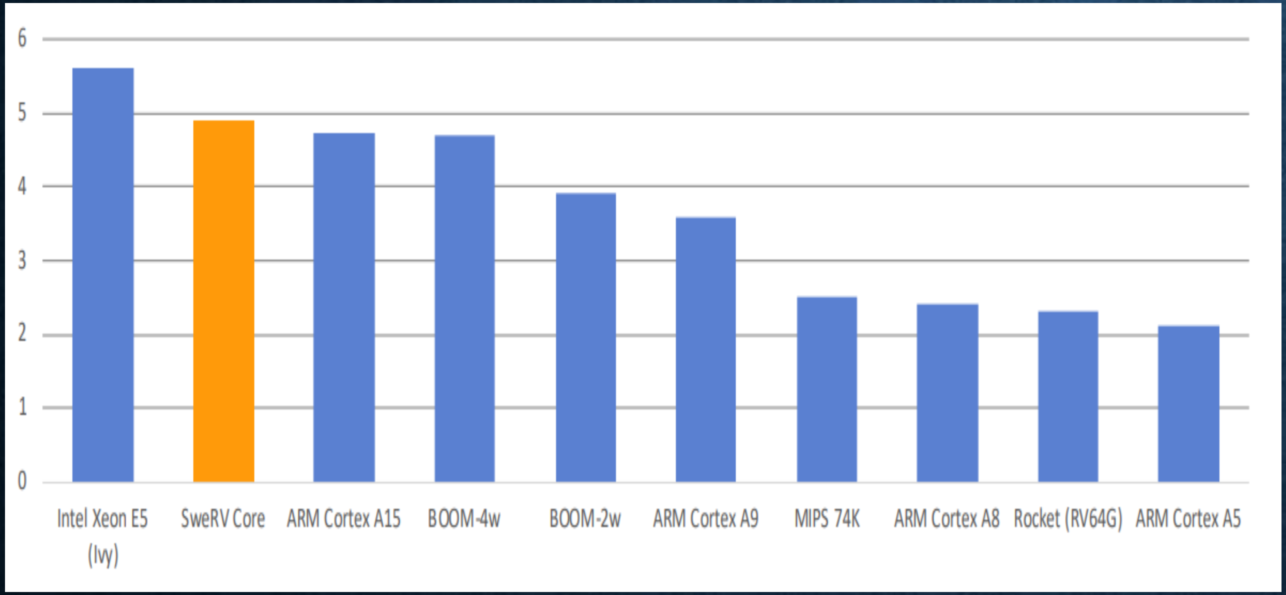
DESIGN UNDER VERIFICATION

- https://github.com/chipsalliance/Cores-SweRV
- The Western Digital SWERV Core EH1 is a 32-bit, dual-issue, 9-stage pipeline core.
- Dual-issue: each clock cycle the processor can move two instructions from one stage of the pipeline to the next stage.



RISC-V PROCESSOR BENCHMARKING

- *Benchmarks determine processor performance by running programs that exercise the hardware.*
- *This enables comparison of different processors.*

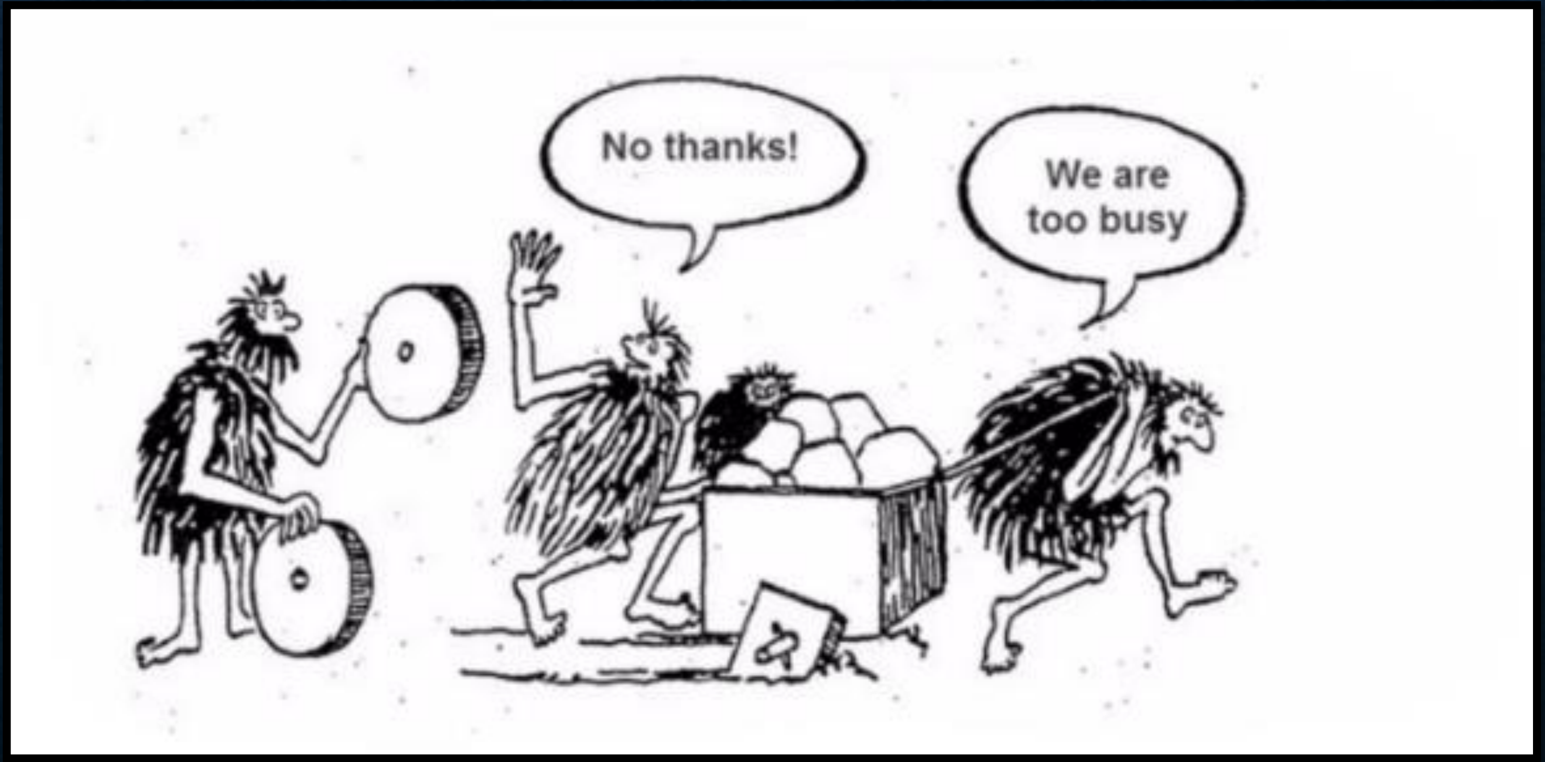


Benchmark	Program Type	Description
Dhrystone	Synthetic	Made to evaluate general purpose performance
CoreMark	Synthetic	Similar to Dhrystone, but improved with more tests

- ✓ 4.9 CoreMark/MHz (The CoreMark Score is the number of iterations completed per second)
- ✓ 2.3 DMIPs/MHz (It's a measure of how many operations the CPU can perform in a single clock cycle)

MOTIVATION

- Accelerate the verification of RISC-V cores by incorporating open-source verification solutions instead of re-inventing the wheel.

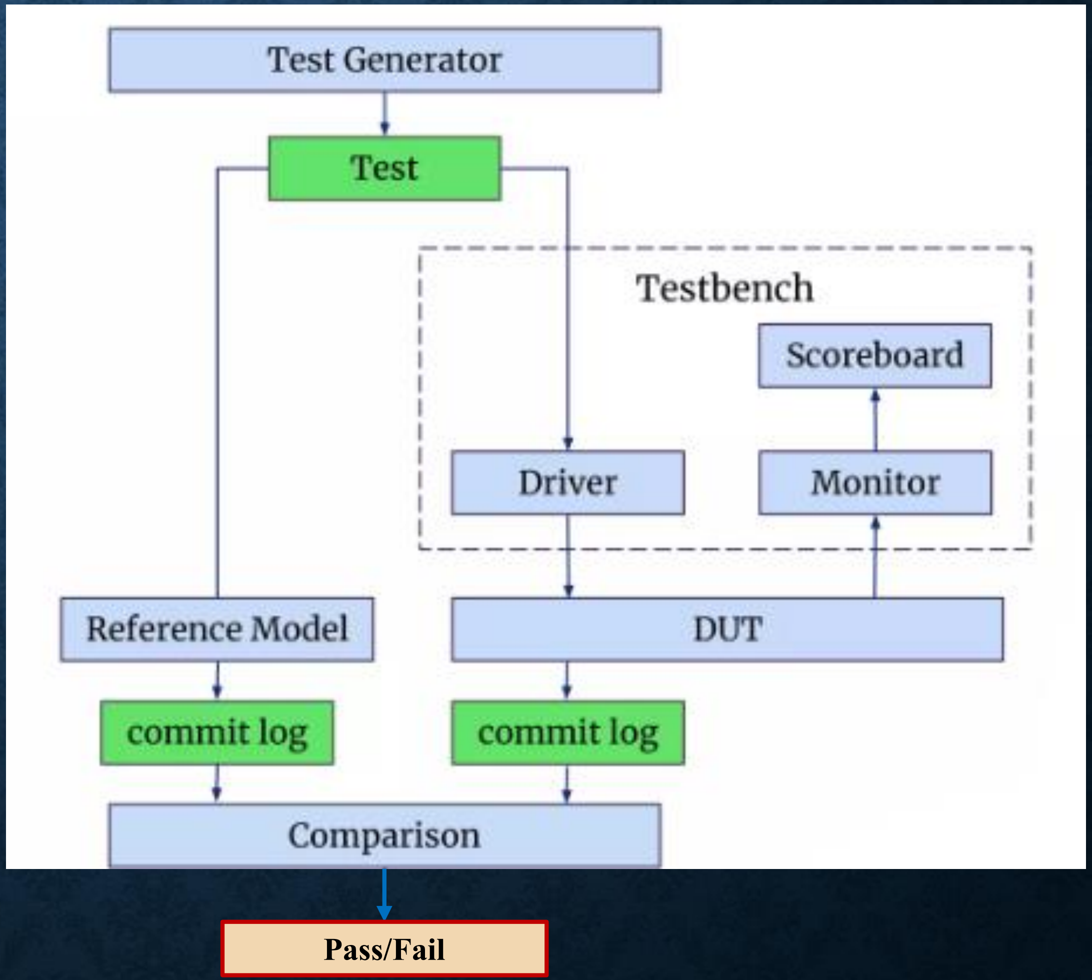


BUILDING BLOCKS OF CPU VERIFICATION

A CPU level design verification environment
Includes;

- DUT RTL
- Testbench
  - Instantiates RTL
  - Driver, Monitor and Scoreboard
- Tests Generator
- Golden Reference Model

Building all these blocks from scratch takes
a lot of time and resources



OPEN-SOURCE RISC-V ECOSYSTEM

- **Google RISC-V DV**
  - An open-source constraint random instruction Generator for RISC-V processor verification
  - Contains Open-source RISC-V Test-Suite
- **RISC-V Toolchain**
- **Spike ISS**

  Open-source RISC-V ISA simulator which implements a functional model of RISC-V Core
- **SWERV-EH1**

  The Western Digital SWERV Core EH-1 is a 32-bit, dual-issue, 9-stage pipeline core

SweRV
+
RISCV-DV
Instruction Generator
+
Spike ISS

SPECIFICATIONS AND SOFTWARE FROM RISCV.ORG AND GITHUB.COM/RISCV

- Open-Source RISC-V processor verification framework
  - ✓ https://github.com/Lampro-Mellon/LM-RISCV-DV

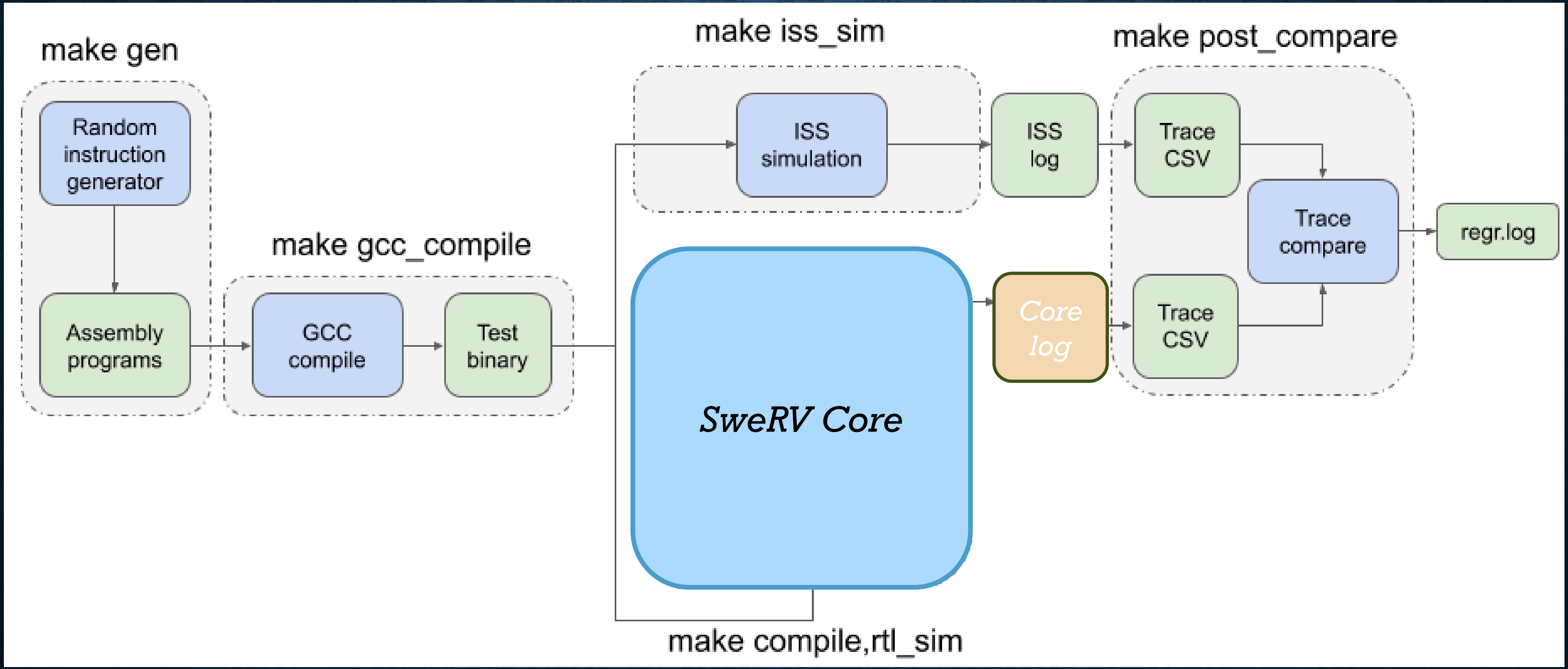
❑ RISC-V software includes

- GNU Compiler Collection (GCC) toolchain (with GDB, the debugger)
  - ✓ https://github.com/riscv/riscv-tools
    - A simulator ("Spike")
  - ✓ https://github.com/riscv/riscv-isa-sim

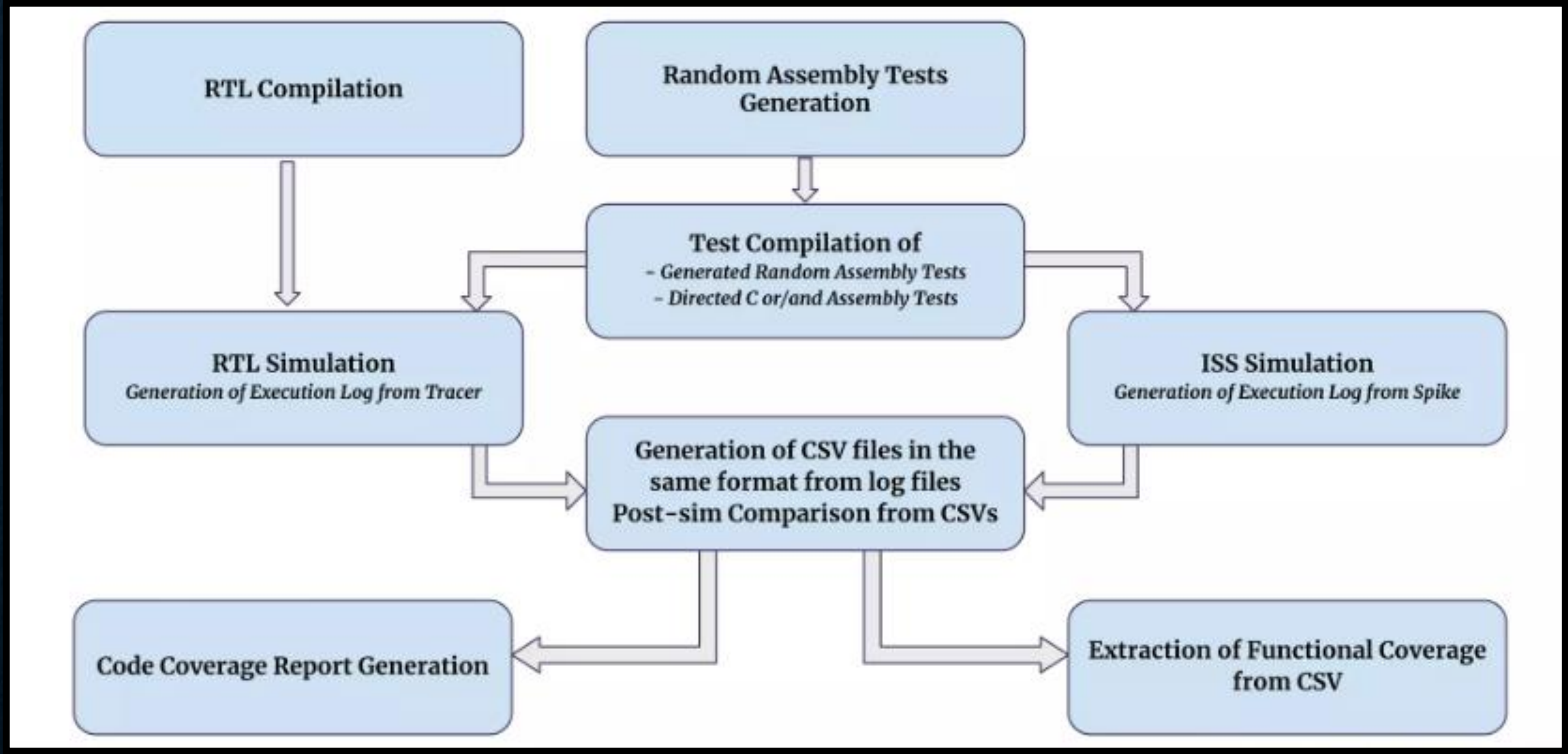
- A list from
  - ✓ https://github.com/riscvarchive/riscv-cores-list

FLOW DIAGRAM

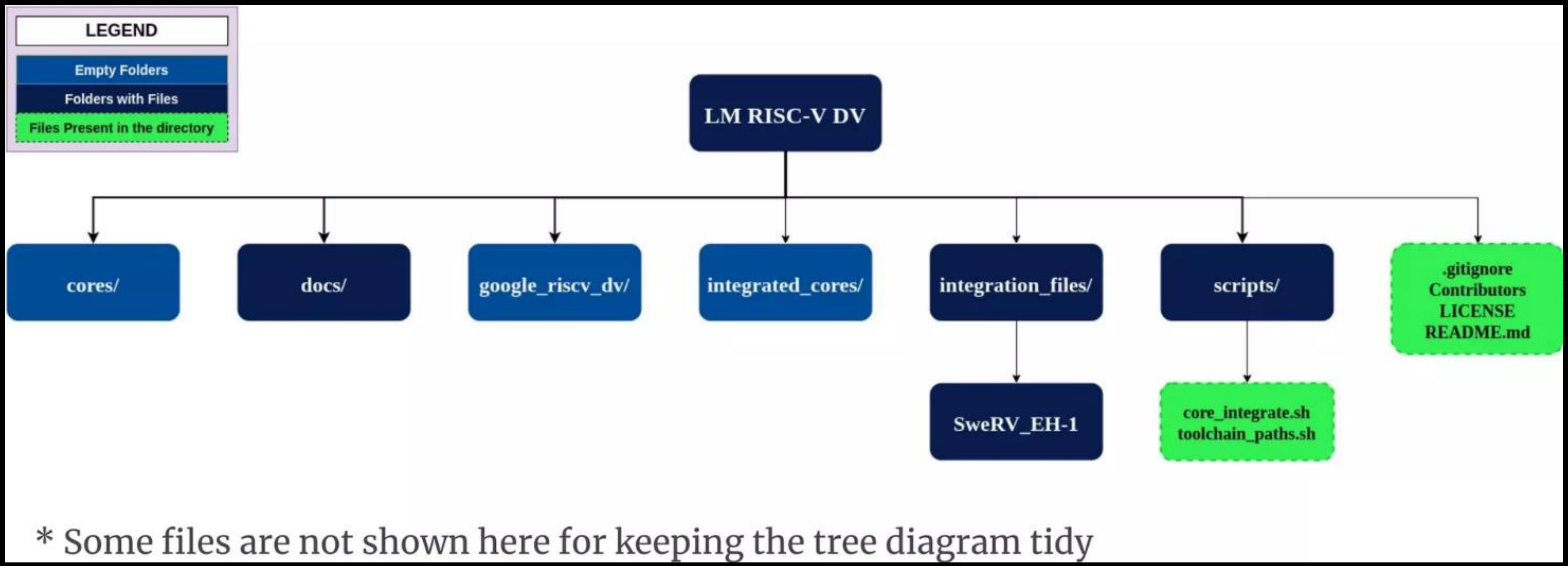
FLOW DIAGRAM



FLOW DIAGRAM WITH COVERAGE

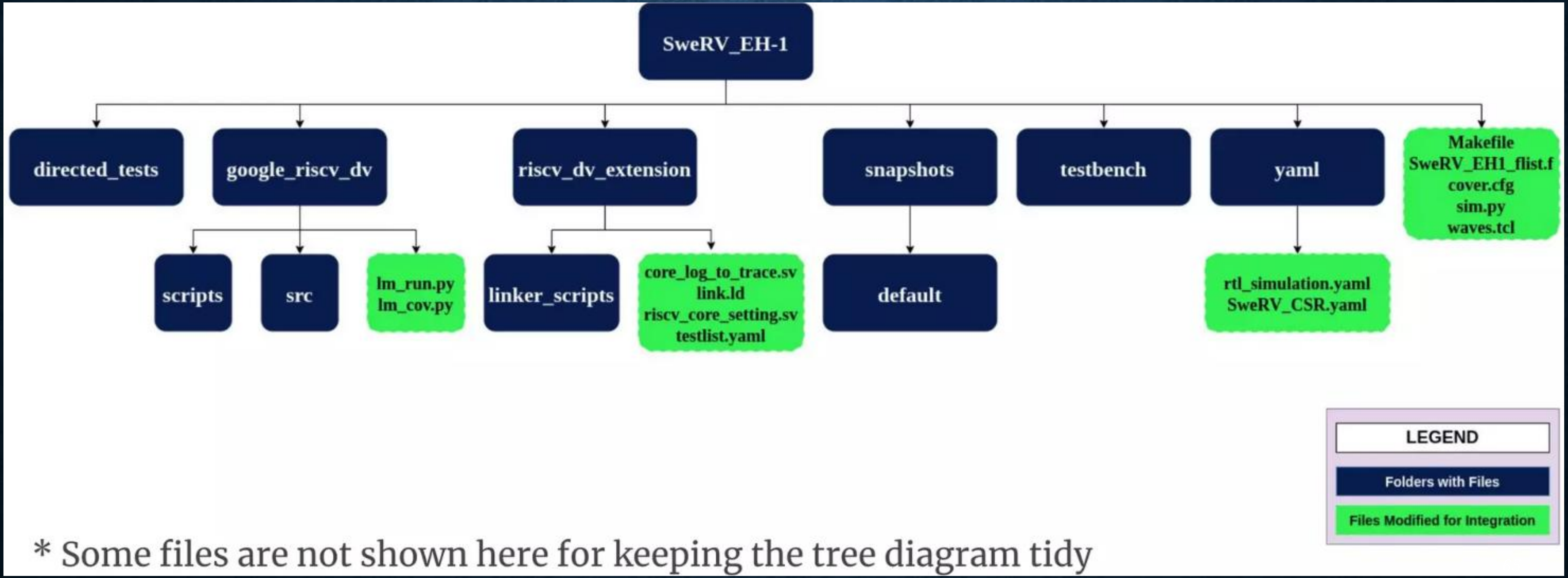

RTL Compilation
Random Assembly Tests Generation
Test Compilation of
- Generated Random Assembly Tests
- Directed C or/and Assembly Tests
RTL Simulation
Generation of Execution Log from Tracer
ISS Simulation
Generation of Execution Log from Spike
Generation of CSV files in the same format from log files
Post-sim Comparison from CSVs
Code Coverage Report Generation
Extraction of Functional Coverage from CSV

RISC-V DV REPOSITORY

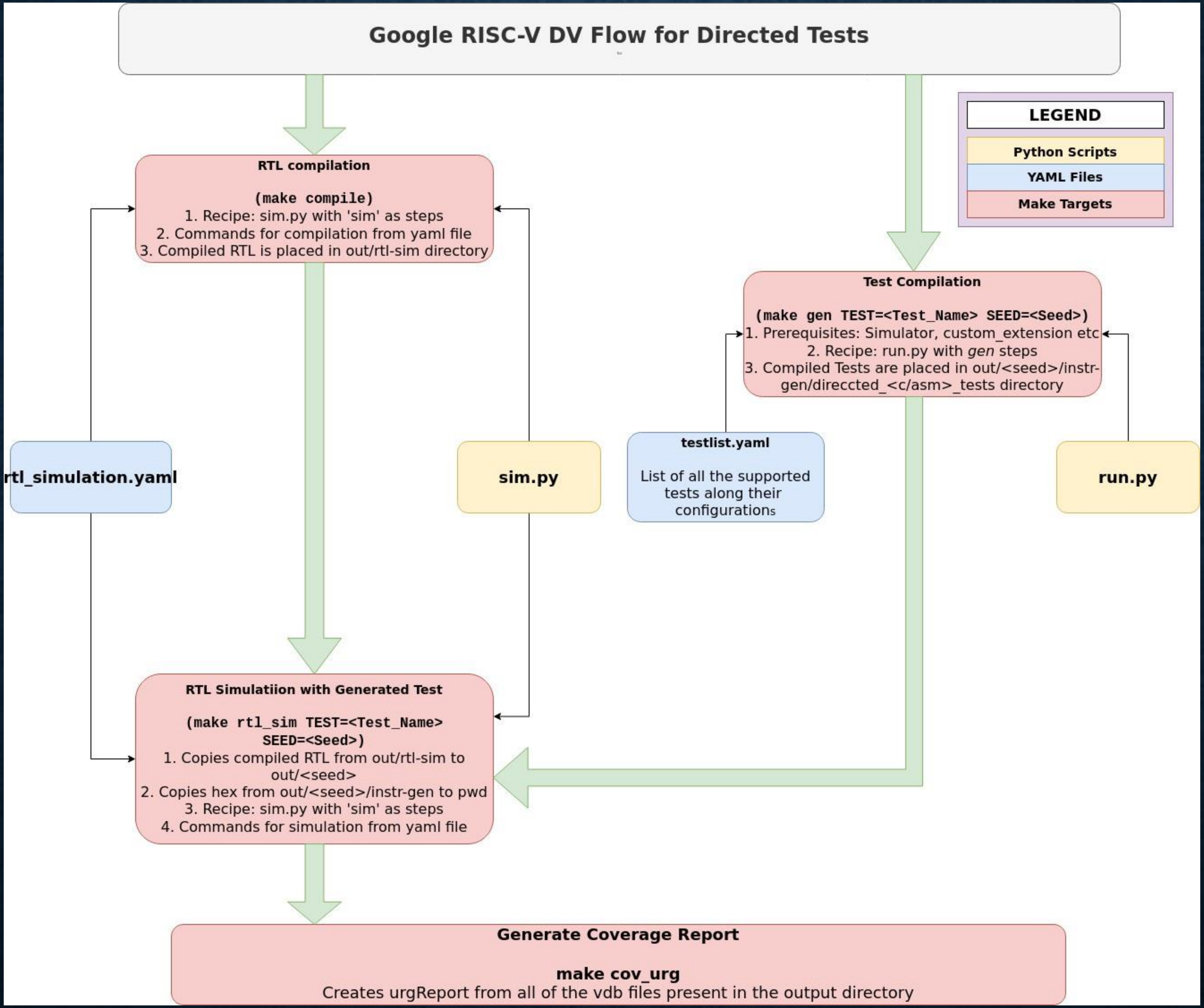


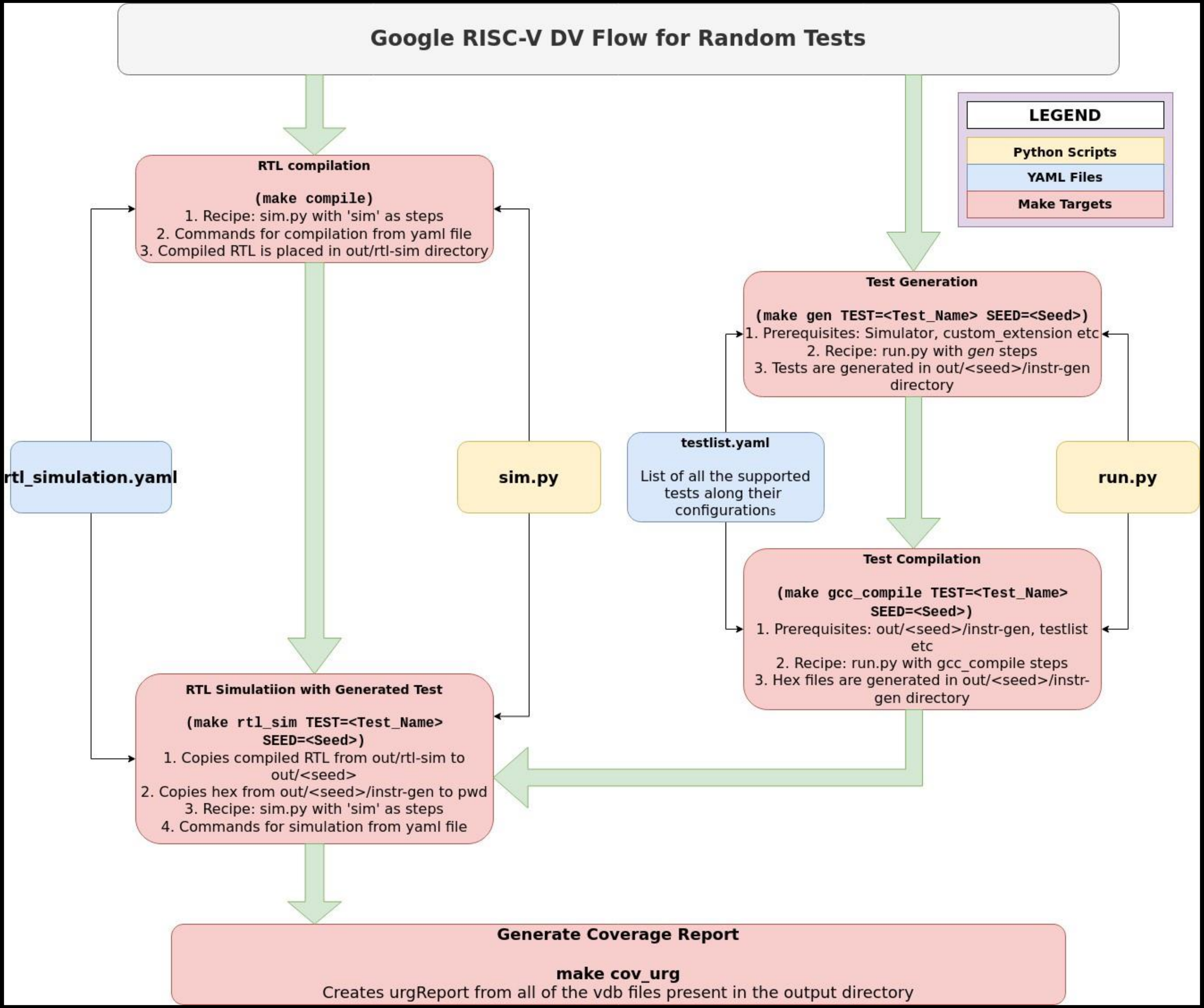
* Some files are not shown here for keeping the tree diagram tidy

FILES FOR INTEGRATION



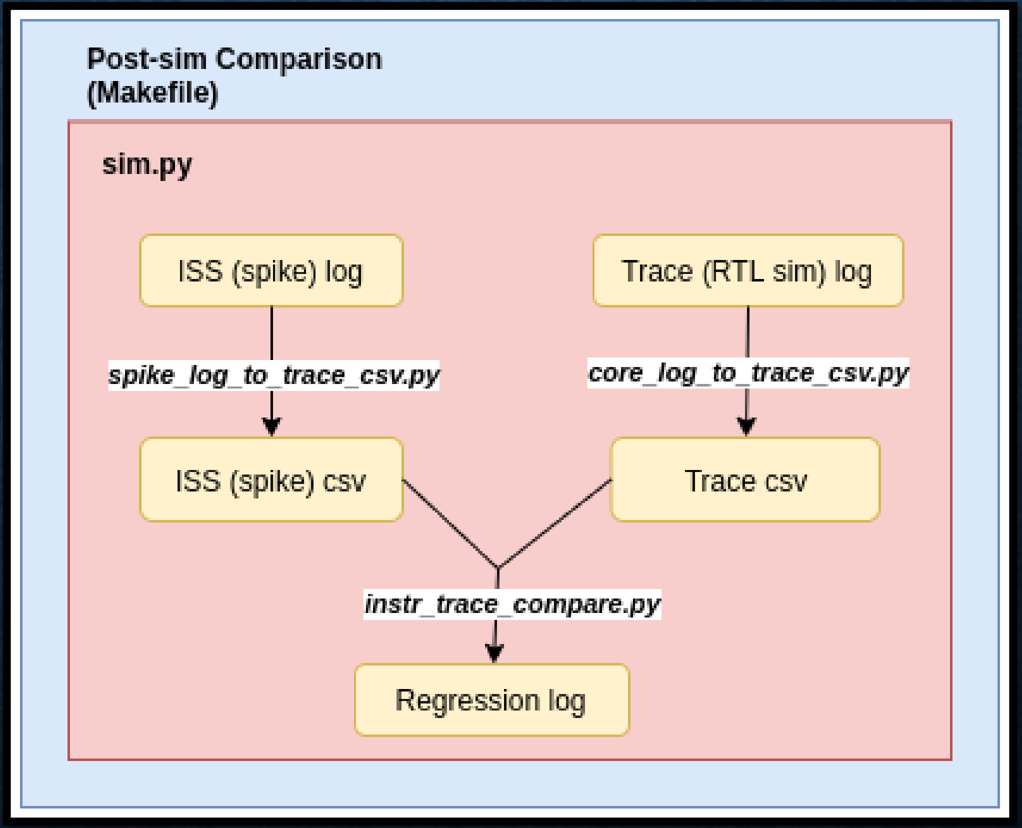
* Some files are not shown here for keeping the tree diagram tidy


Google RISC-V DV Flow for Directed Tests
LEGEND
Python Scripts
YAML Files
Make Targets
RTL compilation
(make compile)
1. Recipe: sim.py with 'sim' as steps
2. Commands for compilation from yaml file
3. Compiled RTL is placed in out/rtl-sim directory
Test Compilation
(make gen TEST=<Test_Name> SEED=<Seed>)
1. Prerequisites: Simulator, custom_extension etc
2. Recipe: run.py with *gen* steps
3. Compiled Tests are placed in out/<seed>/instr-gen/direccted_<c/asm>_tests directory
rtl_simulation.yaml
sim.py
testlist.yaml
List of all the supported tests along their configurations
run.py
RTL Simulatiion with Generated Test
(make rtl_sim TEST=<Test_Name> SEED=<Seed>)
1. Copies compiled RTL from out/rtl-sim to out/<seed>
2. Copies hex from out/<seed>/instr-gen to pwd
3. Recipe: sim.py with 'sim' as steps
4. Commands for simulation from yaml file
Generate Coverage Report
make cov_urg
Creates urgReport from all of the vdb files present in the output directory

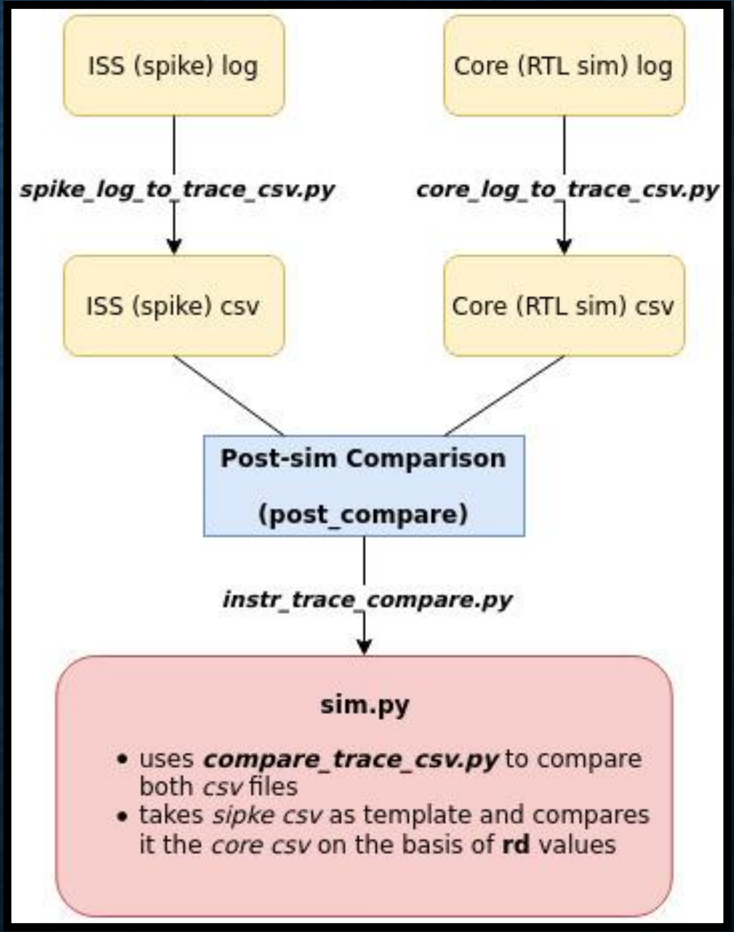


Google RISC-V DV Flow for Random Tests
LEGEND
Python Scripts
YAML Files
Make Targets
RTL compilation
(make compile)
1. Recipe: sim.py with 'sim' as steps
2. Commands for compilation from yaml file
3. Compiled RTL is placed in out/rtl-sim directory
rtl_simulation.yaml
sim.py
RTL Simulatiion with Generated Test
(make rtl_sim TEST=<Test_Name> SEED=<Seed>)
1. Copies compiled RTL from out/rtl-sim to out/<seed>
2. Copies hex from out/<seed>/instr-gen to pwd
3. Recipe: sim.py with 'sim' as steps
4. Commands for simulation from yaml file
Test Generation
(make gen TEST=<Test_Name> SEED=<Seed>)
1. Prerequisites: Simulator, custom_extension etc
2. Recipe: run.py with gen steps
3. Tests are generated in out/<seed>/instr-gen directory
testlist.yaml
List of all the supported tests along their configurations
run.py
Test Compilation
(make gcc_compile TEST=<Test_Name> SEED=<Seed>)
1. Prerequisites: out/<seed>/instr-gen, testlist etc
2. Recipe: run.py with gcc_compile steps
3. Hex files are generated in out/<seed>/instr-gen directory
Generate Coverage Report
make cov_urg
Creates urgReport from all of the vdb files present in the output directory

POST-COMPARISON STEPS



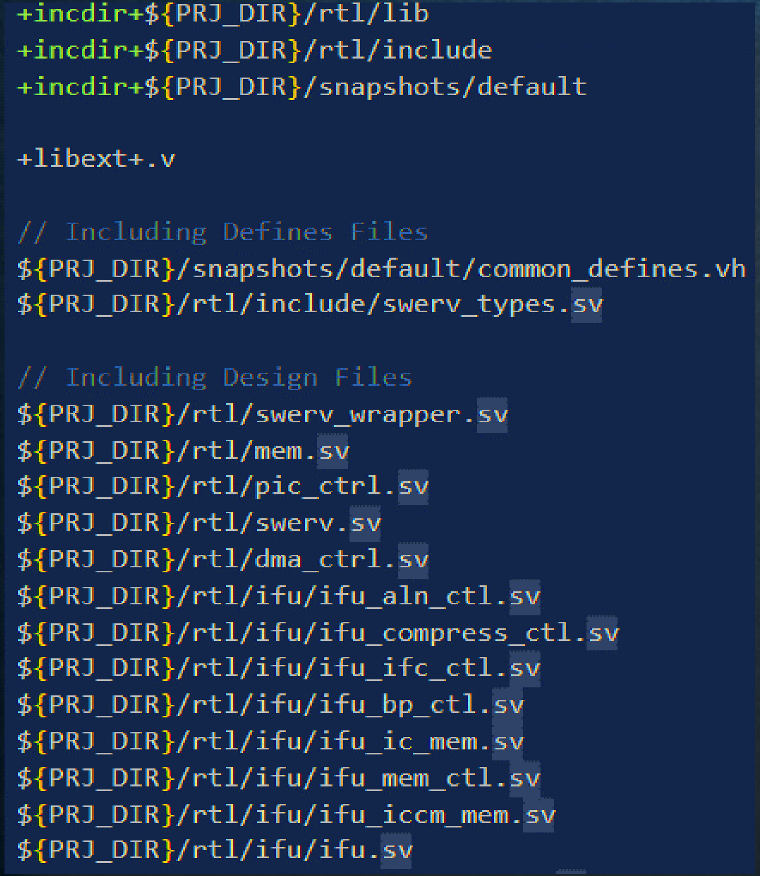
POST-COMPARISON STEPS


ISS (spike) log
Core (RTL sim) log
spike_log_to_trace_csv.py
core_log_to_trace_csv.py
ISS (spike) csv
Core (RTL sim) csv
Post-sim Comparison
(post_compare)
instr_trace_compare.py
sim.py
• uses compare_trace_csv.py to compare both csv files
• takes sipke csv as template and compares it the core csv on the basis of rd values

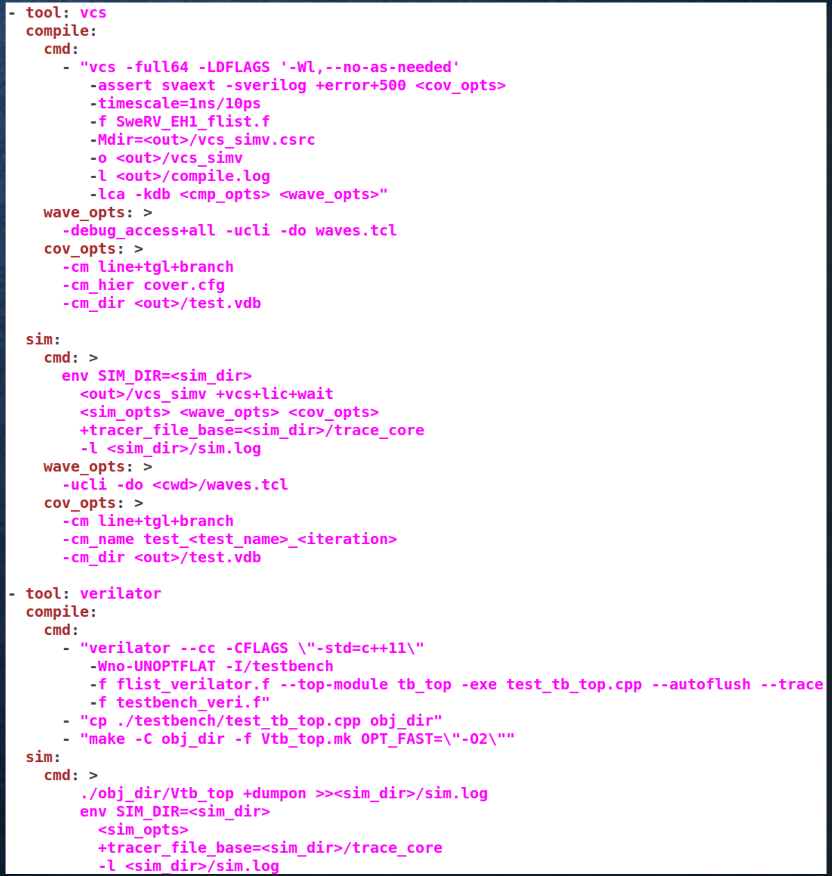
INITIAL SETTINGS

TOOLS SETTINGS

- Add the **VCS** compiler into the **rtl_simulation.yaml** file and add the **flist** there.
- SweRV_flist.f file contains all the files included in the design hierarchy.
- SweRV **flist** & **rtl_simulation.yaml** file is shown in fig:



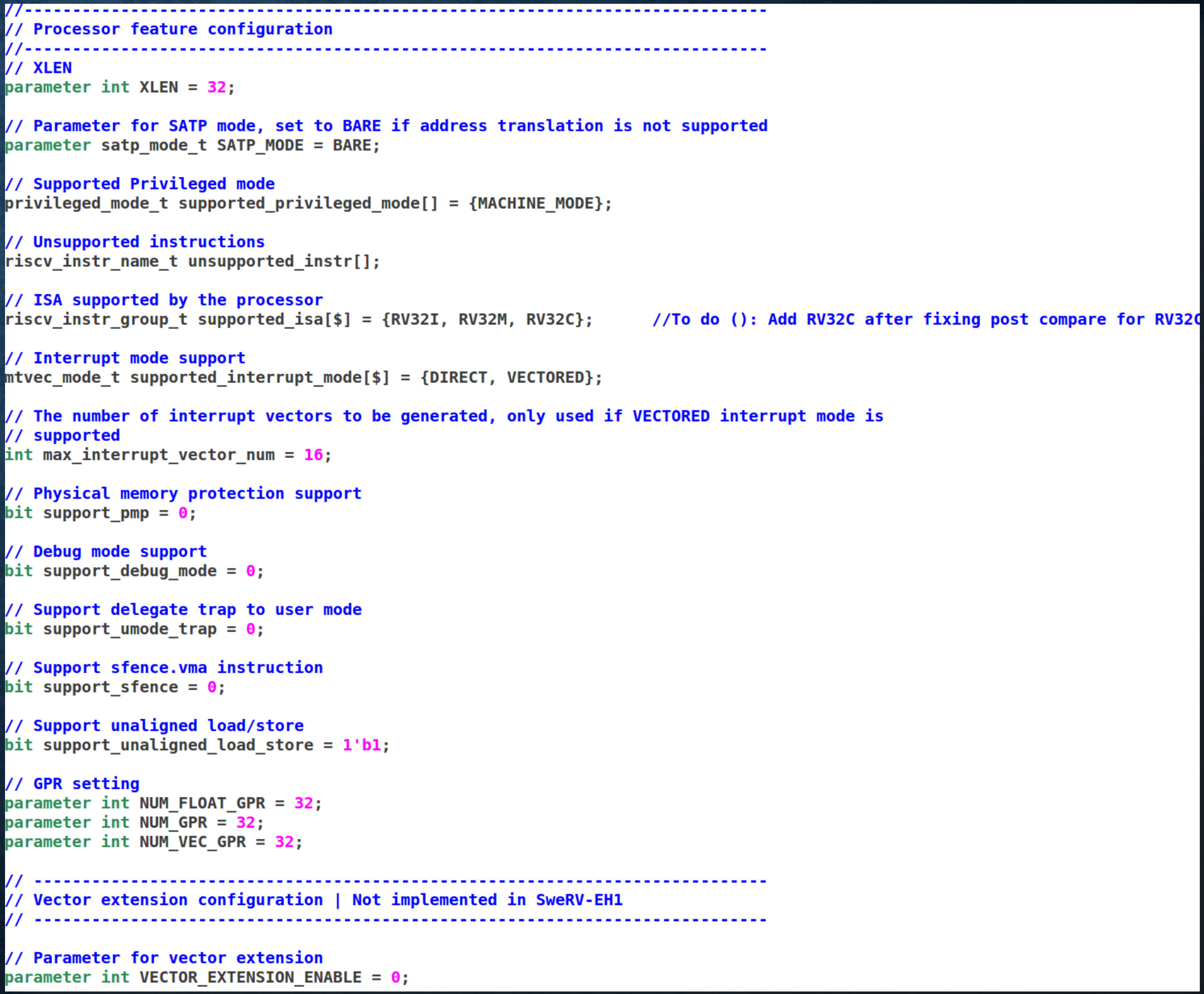
```
+incdir+${PRJ_DIR}/rtl/lib
+incdir+${PRJ_DIR}/rtl/include
+incdir+${PRJ_DIR}/snapshots/default

+libext+.v

// Including Defines Files
${PRJ_DIR}/snapshots/default/common_defines.vh
${PRJ_DIR}/rtl/include/swerv_types.sv

// Including Design Files
${PRJ_DIR}/rtl/swerv_wrapper.sv
${PRJ_DIR}/rtl/mem.sv
${PRJ_DIR}/rtl/pic_ctrl.sv
${PRJ_DIR}/rtl/swerv.sv
${PRJ_DIR}/rtl/dma_ctrl.sv
${PRJ_DIR}/rtl/ifu/ifu_aln_ctl.sv
${PRJ_DIR}/rtl/ifu/ifu_compress_ctl.sv
${PRJ_DIR}/rtl/ifu/ifu_ifc_ctl.sv
${PRJ_DIR}/rtl/ifu/ifu_bp_ctl.sv
${PRJ_DIR}/rtl/ifu/ifu_ic_mem.sv
${PRJ_DIR}/rtl/ifu/ifu_mem_ctl.sv
${PRJ_DIR}/rtl/ifu/ifu_iccm_mem.sv
${PRJ_DIR}/rtl/ifu/ifu.sv
```



```
- tool: vcs
  compile:
    cmd:
      - "vcs -full64 -LDFLAGS '-Wl,--no-as-needed'
         -assert svaext -sverilog +error+500 <cov_opts>
         -timescale=1ns/10ps
         -f SweRV_EH1_flist.f
         -Mdir=<out>/vcs_simv.csrc
         -o <out>/vcs_simv
         -l <out>/compile.log
         -lca -kdb <cmp_opts> <wave_opts>"
    wave_opts: >
      -debug_access+all -ucli -do waves.tcl
    cov_opts: >
      -cm line+tgl+branch
      -cm_hier cover.cfg
      -cm_dir <out>/test.vdb

  sim:
    cmd: >
      env SIM_DIR=<sim_dir>
        <out>/vcs_simv +vcs+lic+wait
        <sim_opts> <wave_opts> <cov_opts>
        +tracer_file_base=<sim_dir>/trace_core
        -l <sim_dir>/sim.log
    wave_opts: >
      -ucli -do <cwd>/waves.tcl
    cov_opts: >
      -cm line+tgl+branch
      -cm_name test_<test_name>_<iteration>
      -cm_dir <out>/test.vdb

- tool: verilator
  compile:
    cmd:
      - "verilator --cc -CFLAGS \"-std=c++11\"
         -Wno-UNOPTFLAT -I/testbench
         -f flist_verilator.f --top-module tb_top -exe test_tb_top.cpp --autoflush --trace
         -f testbench_veri.f"
      - "cp ./testbench/test_tb_top.cpp obj_dir"
      - "make -C obj_dir -f Vtb_top.mk OPT_FAST=\"-O2\""
  sim:
    cmd: >
        ./obj_dir/Vtb_top +dumpon >><sim_dir>/sim.log
        env SIM_DIR=<sim_dir>
          <sim_opts>
          +tracer_file_base=<sim_dir>/trace_core
          -l <sim_dir>/sim.log
```

RISC-V CORE SETTINGS

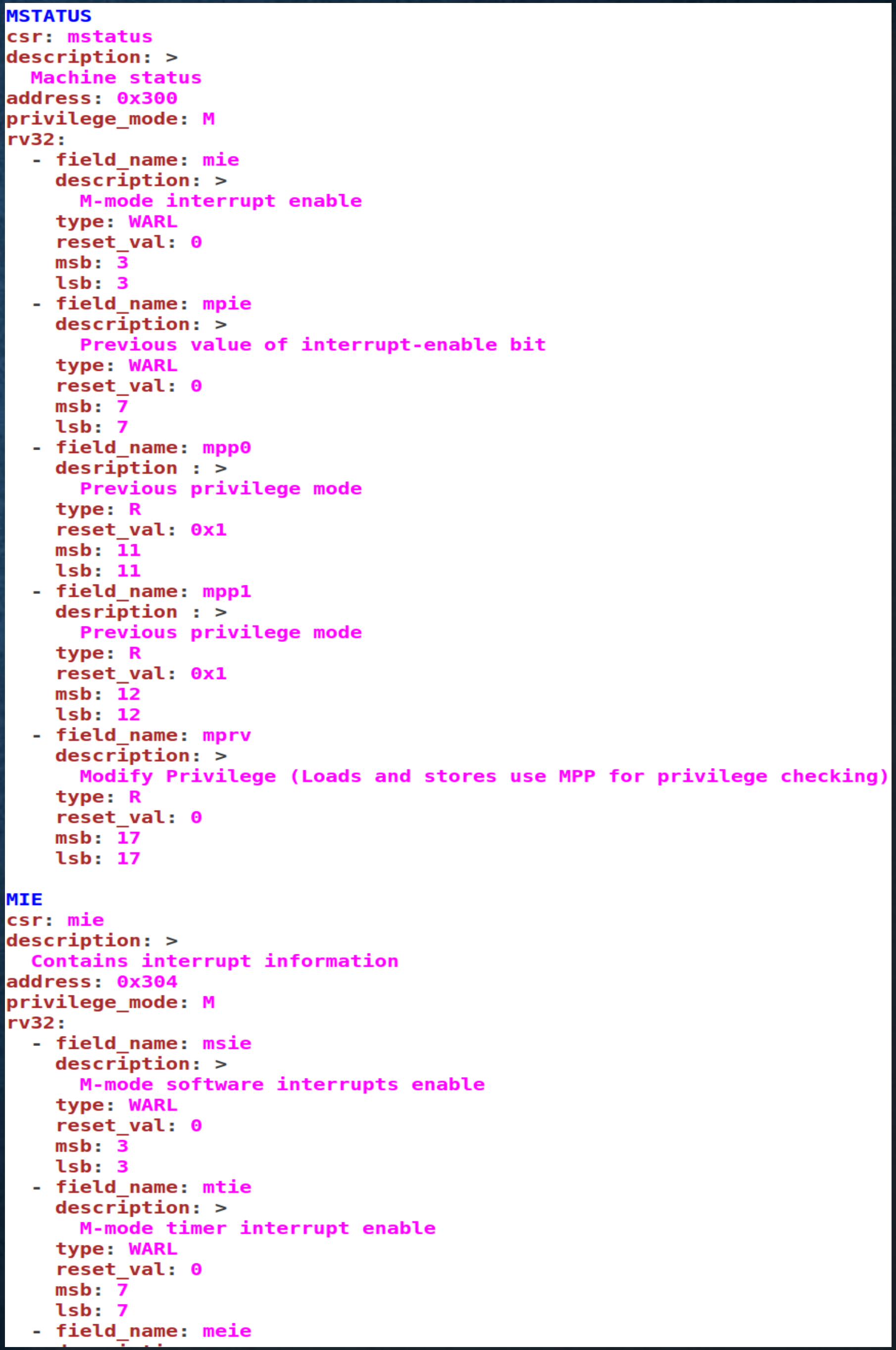
- Configure the riscv_core_setting.sv file according to SweRV core parameters e.g, mode, supported ISA etc.



```
//-----------------------------------------------------------------------------
// Processor feature configuration
//-----------------------------------------------------------------------------
// XLEN
parameter int XLEN = 32;

// Parameter for SATP mode, set to BARE if address translation is not supported
parameter satp_mode_t SATP_MODE = BARE;

// Supported Privileged mode
privileged_mode_t supported_privileged_mode[] = {MACHINE_MODE};

// Unsupported instructions
riscv_instr_name_t unsupported_instr[];

// ISA supported by the processor
riscv_instr_group_t supported_isa[$] = {RV32I, RV32M, RV32C};      //To do (): Add RV32C after fixing post compare for RV32C

// Interrupt mode support
mtvec_mode_t supported_interrupt_mode[$] = {DIRECT, VECTORED};

// The number of interrupt vectors to be generated, only used if VECTORED interrupt mode is
// supported
int max_interrupt_vector_num = 16;

// Physical memory protection support
bit support_pmp = 0;

// Debug mode support
bit support_debug_mode = 0;

// Support delegate trap to user mode
bit support_umode_trap = 0;

// Support sfence.vma instruction
bit support_sfence = 0;

// Support unaligned load/store
bit support_unaligned_load_store = 1'b1;

// GPR setting
parameter int NUM_FLOAT_GPR = 32;
parameter int NUM_GPR = 32;
parameter int NUM_VEC_GPR = 32;

// ----------------------------------------------------------------------------
// Vector extension configuration | Not implemented in SweRV-EH1
// ----------------------------------------------------------------------------

// Parameter for vector extension
parameter int VECTOR_EXTENSION_ENABLE = 0;
```

STANDARD RISC-V SweRV CSRs CONFIGURATIONS

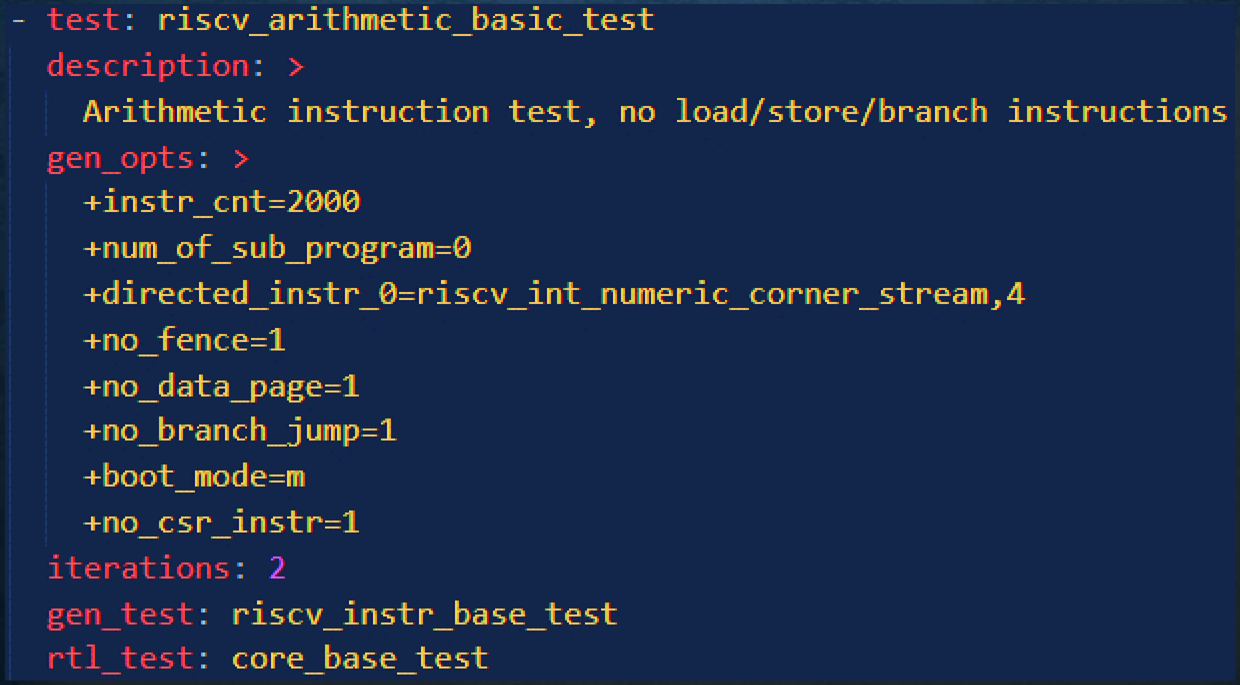
- Configure all the SweRV core CSRs with bit fields in the **SweRV_CSR.yaml** file.
- Screenshot of the SweRV **mstatus** & **mie** CSRs is shown in fig:



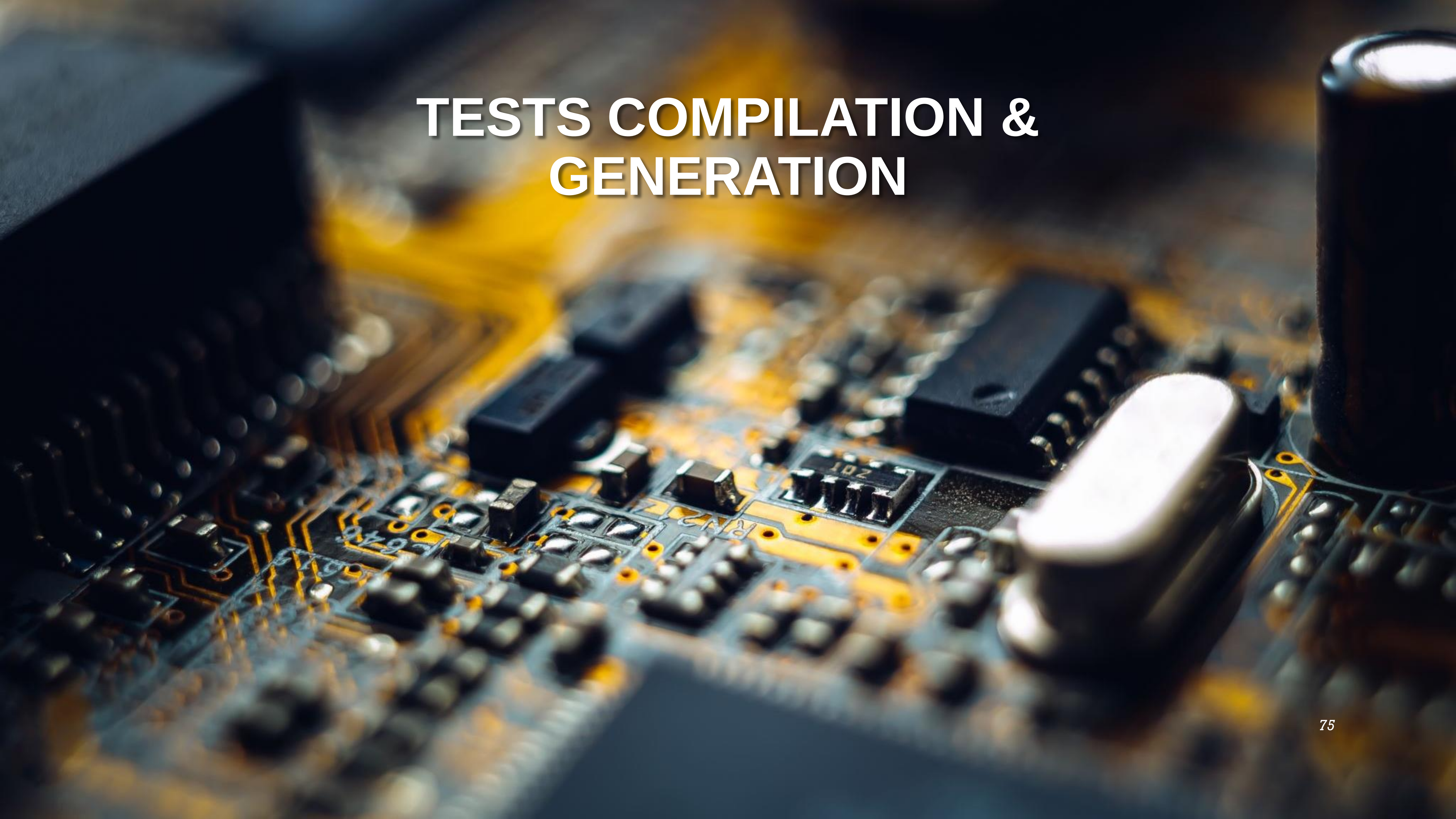
```
MSTATUS
csr: mstatus
description: >
  Machine status
address: 0x300
privilege_mode: M
rv32:
  - field_name: mie
    description: >
      M-mode interrupt enable
    type: WARL
    reset_val: 0
    msb: 3
    lsb: 3
  - field_name: mpie
    description: >
      Previous value of interrupt-enable bit
    type: WARL
    reset_val: 0
    msb: 7
    lsb: 7
  - field_name: mpp0
    desription : >
      Previous privilege mode
    type: R
    reset_val: 0x1
    msb: 11
    lsb: 11
  - field_name: mpp1
    desription : >
      Previous privilege mode
    type: R
    reset_val: 0x1
    msb: 12
    lsb: 12
  - field_name: mprv
    description: >
      Modify Privilege (Loads and stores use MPP for privilege checking)
    type: R
    reset_val: 0
    msb: 17
    lsb: 17

MIE
csr: mie
description: >
  Contains interrupt information
address: 0x304
privilege_mode: M
rv32:
  - field_name: msie
    description: >
      M-mode software interrupts enable
    type: WARL
    reset_val: 0
    msb: 3
    lsb: 3
  - field_name: mtie
    description: >
      M-mode timer interrupt enable
    type: WARL
    reset_val: 0
    msb: 7
    lsb: 7
  - field_name: meie
```

TESTS INCLUSION

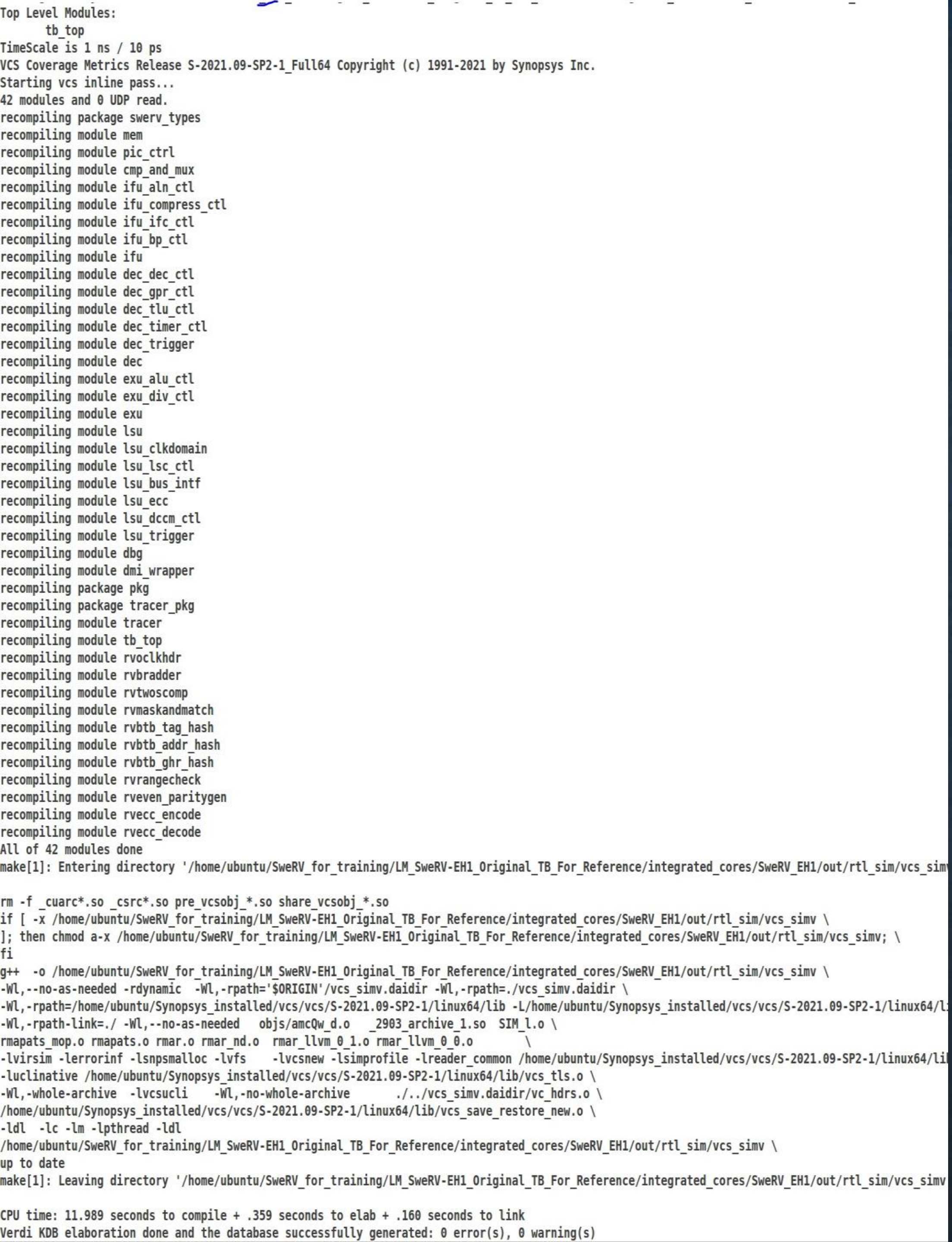
- Tests to be run (Directed/Random) on core should be present in "**testlist.yaml**" file. Parameters like instruction count & iterations are set in this file.
- Screenshot of "**riscv_arithmetic_basic_test**" is shown below:



```
- test: riscv_arithmetic_basic_test
  description: >
    Arithmetic instruction test, no load/store/branch instructions
  gen_opts: >
    +instr_cnt=2000
    +num_of_sub_program=0
    +directed_instr_0=riscv_int_numeric_corner_stream,4
    +no_fence=1
    +no_data_page=1
    +no_branch_jump=1
    +boot_mode=m
    +no_csr_instr=1
  iterations: 2
  gen_test: riscv_instr_base_test
  rtl_test: core_base_test
```



TESTS COMPILATION & GENERATION



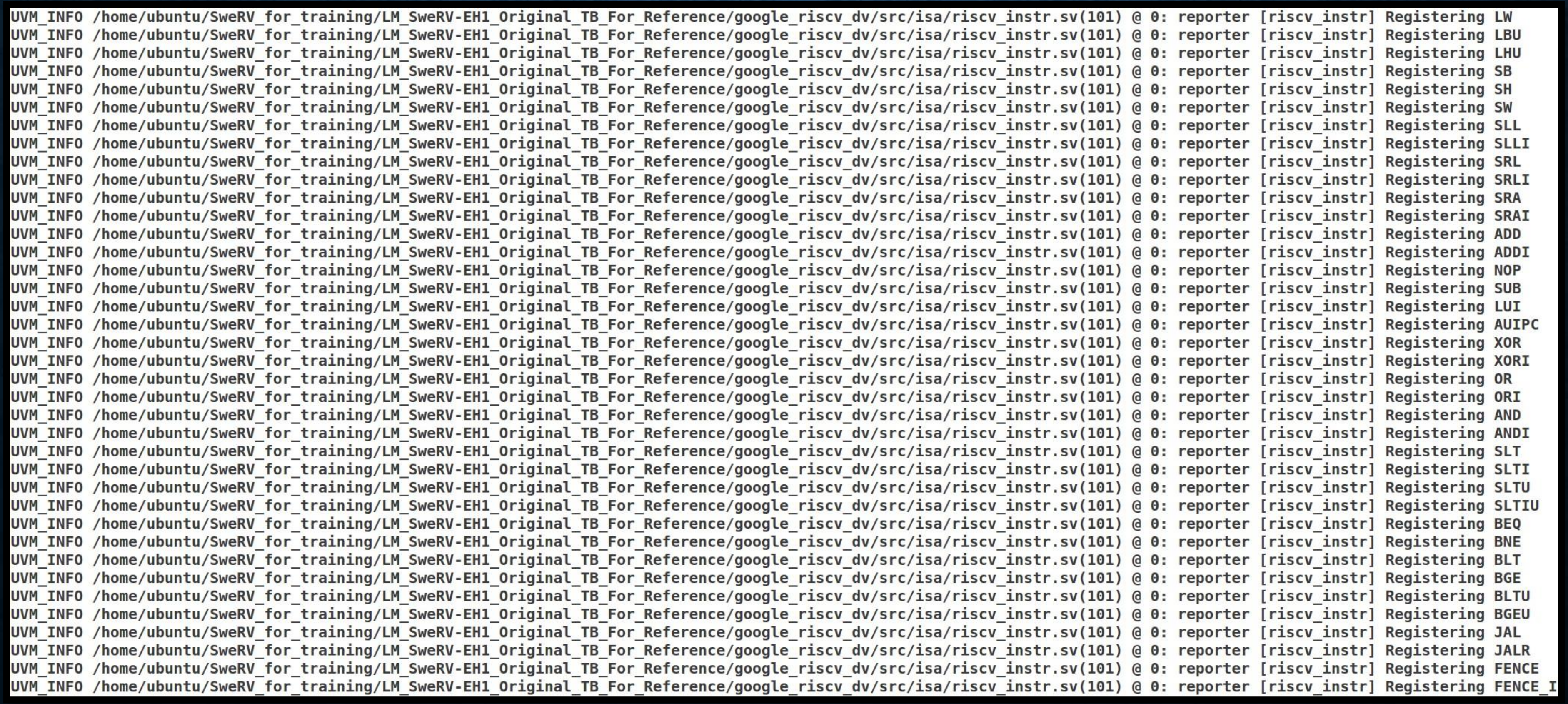
```
Top Level Modules:
       tb_top
TimeScale is 1 ns / 10 ps
VCS Coverage Metrics Release S-2021.09-SP2-1_Full64 Copyright (c) 1991-2021 by Synopsys Inc.
Starting vcs inline pass...
42 modules and 0 UDP read.
recompiling package swerv_types
recompiling module mem
recompiling module pic_ctrl
recompiling module cmp_and_mux
recompiling module ifu_aln_ctl
recompiling module ifu_compress_ctl
recompiling module ifu_ifc_ctl
recompiling module ifu_bp_ctl
recompiling module ifu
recompiling module dec_dec_ctl
recompiling module dec_gpr_ctl
recompiling module dec_tlu_ctl
recompiling module dec_timer_ctl
recompiling module dec_trigger
recompiling module dec
recompiling module exu_alu_ctl
recompiling module exu_div_ctl
recompiling module exu
recompiling module lsu
recompiling module lsu_clkdomain
recompiling module lsu_lsc_ctl
recompiling module lsu_bus_intf
recompiling module lsu_ecc
recompiling module lsu_dccm_ctl
recompiling module lsu_trigger
recompiling module dbg
recompiling module dmi_wrapper
recompiling package pkg
recompiling package tracer_pkg
recompiling module tracer
recompiling module tb_top
recompiling module rvoclkhdr
recompiling module rvbradder
recompiling module rvtwoscomp
recompiling module rvmaskandmatch
recompiling module rvbtb_tag_hash
recompiling module rvbtb_addr_hash
recompiling module rvbtb_ghr_hash
recompiling module rvrangecheck
recompiling module rveven_paritygen
recompiling module rvecc_encode
recompiling module rvecc_decode
All of 42 modules done
make[1]: Entering directory '/home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/integrated_cores/SweRV_EH1/out/rtl_sim/vcs_sim

rm -f _cuarc*.so _csrc*.so pre_vcsobj_*.so share_vcsobj_*.so
if [ -x /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/integrated_cores/SweRV_EH1/out/rtl_sim/vcs_simv \
]; then chmod a-x /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/integrated_cores/SweRV_EH1/out/rtl_sim/vcs_simv; \
fi
g++  -o /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/integrated_cores/SweRV_EH1/out/rtl_sim/vcs_simv \
-Wl,--no-as-needed -rdynamic  -Wl,-rpath='$ORIGIN'/vcs_simv.daidir -Wl,-rpath=./vcs_simv.daidir \
-Wl,-rpath=/home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1/linux64/lib -L/home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1/linux64/l
-Wl,-rpath-link=./ -Wl,--no-as-needed   objs/amcQw_d.o   _2903_archive_1.so  SIM_l.o \
rmapats_mop.o rmapats.o rmar.o rmar_nd.o  rmar_llvm_0_1.o rmar_llvm_0_0.o        \
-lvirsim -lerrorinf -lsnpsmalloc -lvfs    -lvcsnew -lsimprofile -lreader_common /home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1/linux64/li
-luclinative /home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1/linux64/lib/vcs_tls.o \
-Wl,-whole-archive  -lvcsucli    -Wl,-no-whole-archive       ./../vcs_simv.daidir/vc_hdrs.o \
/home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1/linux64/lib/vcs_save_restore_new.o \
-ldl  -lc -lm -lpthread -ldl
/home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/integrated_cores/SweRV_EH1/out/rtl_sim/vcs_simv \
up to date
make[1]: Leaving directory '/home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/integrated_cores/SweRV_EH1/out/rtl_sim/vcs_simv

CPU time: 11.989 seconds to compile + .359 seconds to elab + .160 seconds to link
Verdi KDB elaboration done and the database successfully generated: 0 error(s), 0 warning(s)
```

COMPILING THE TESTBENCH FRAMEWORK

- *Framework Compilation Command:*
- *"**make compile**"*
- *Screenshot of **compile_log** is shown in fig:*

RISC-V ARITHMETIC BASIC TEST GENERATION

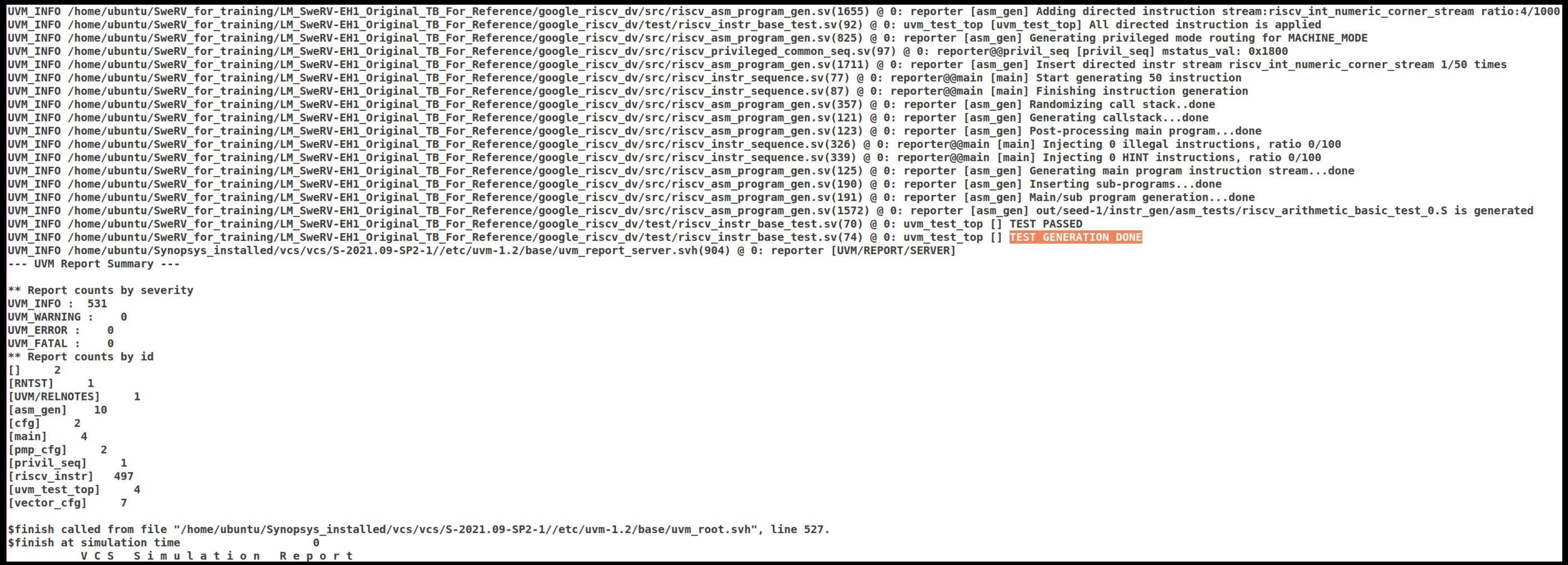
- RISC-V test generation Command:
- “***make gen TEST=riscv_arithmetic_basic_test SEED=1***”
- Screenshot of ***test_generation_log*** is shown in fig:



```
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering LW
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering LBU
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering LHU
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SB
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SH
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SW
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SLL
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SLLI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SRL
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SRLI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SRA
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SRAI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering ADD
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering ADDI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering NOP
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SUB
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering LUI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering AUIPC
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering XOR
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering XORI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering OR
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering ORI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering AND
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering ANDI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SLT
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SLTI
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SLTU
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering SLTIU
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering BEQ
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering BNE
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering BLT
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering BGE
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering BLTU
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering BGEU
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering JAL
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering JALR
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering FENCE
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/isa/riscv_instr.sv(101) @ 0: reporter [riscv_instr] Registering FENCE_I
```

RISCV ARITHMETIC BASIC TESTGENERATION

- RISC-V test generation Command:
- “***make gen TEST=riscv_arithmetic_basic_test SEED=1***”
- Screenshot of ***test_generation_log*** is shown in fig:

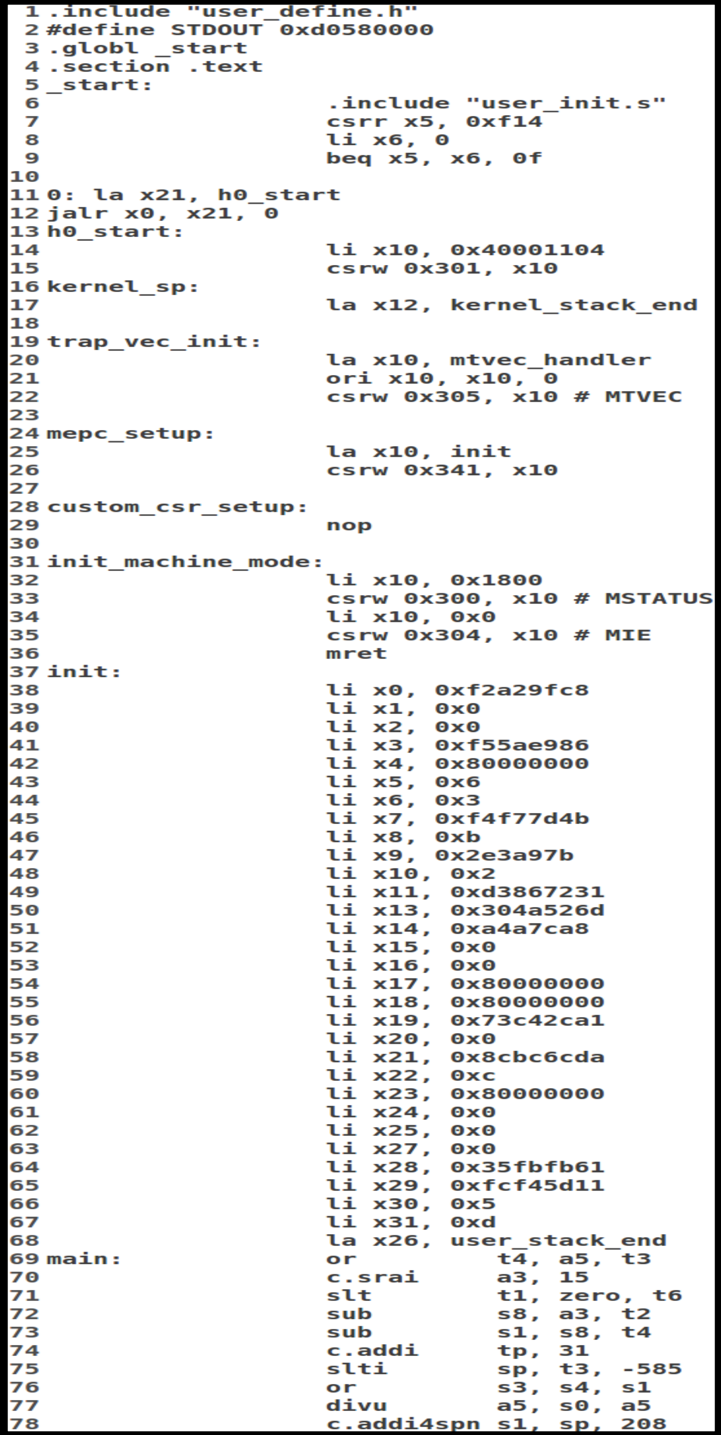


```
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(1655) @ 0: reporter [asm_gen] Adding directed instruction stream:riscv_int_numeric_corner_stream ratio:4/1000
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/test/riscv_instr_base_test.sv(92) @ 0: uvm_test_top [uvm_test_top] All directed instruction is applied
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(825) @ 0: reporter [asm_gen] Generating privileged mode routing for MACHINE_MODE
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_privileged_common_seq.sv(97) @ 0: reporter@@privil_seq [privil_seq] mstatus_val: 0x1800
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(1711) @ 0: reporter [asm_gen] Insert directed instr stream riscv_int_numeric_corner_stream 1/50 times
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_instr_sequence.sv(77) @ 0: reporter@@main [main] Start generating 50 instruction
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_instr_sequence.sv(87) @ 0: reporter@@main [main] Finishing instruction generation
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(357) @ 0: reporter [asm_gen] Randomizing call stack..done
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(121) @ 0: reporter [asm_gen] Generating callstack...done
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(123) @ 0: reporter [asm_gen] Post-processing main program...done
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_instr_sequence.sv(326) @ 0: reporter@@main [main] Injecting 0 illegal instructions, ratio 0/100
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_instr_sequence.sv(339) @ 0: reporter@@main [main] Injecting 0 HINT instructions, ratio 0/100
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(125) @ 0: reporter [asm_gen] Generating main program instruction stream...done
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(190) @ 0: reporter [asm_gen] Inserting sub-programs...done
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(191) @ 0: reporter [asm_gen] Main/sub program generation...done
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/src/riscv_asm_program_gen.sv(1572) @ 0: reporter [asm_gen] out/seed-1/instr_gen/asm_tests/riscv_arithmetic_basic_test_0.S is generated
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/test/riscv_instr_base_test.sv(70) @ 0: uvm_test_top [] TEST PASSED
UVM_INFO /home/ubuntu/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Reference/google_riscv_dv/test/riscv_instr_base_test.sv(74) @ 0: uvm_test_top [] TEST GENERATION DONE
UVM_INFO /home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1//etc/uvm-1.2/base/uvm_report_server.svh(904) @ 0: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :  531
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[]     2
[RNTST]     1
[UVM/RELNOTES]     1
[asm_gen]    10
[cfg]     2
[main]     4
[pmp_cfg]     2
[privil_seq]     1
[riscv_instr]   497
[uvm_test_top]     4
[vector_cfg]     7

$finish called from file "/home/ubuntu/Synopsys_installed/vcs/vcs/S-2021.09-SP2-1//etc/uvm-1.2/base/uvm_root.svh", line 527.
$finish at simulation time                    0
           V C S   S i m u l a t i o n   R e p o r t
```

RISC-V GENERATED ASSEMBLY TEST

- Generates RISC-V test in the form of assembly code with the file named “***riscv_arithmetic_basic_test_0.S***”.

- Screenshot of generated test file is shown in fig:



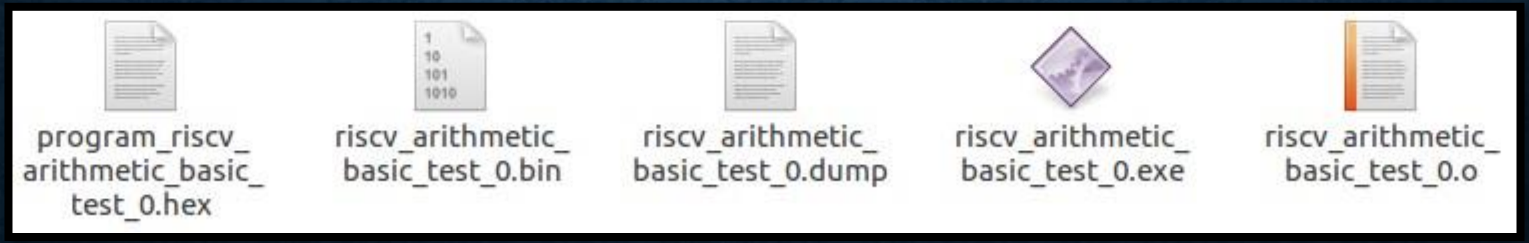
```
.include "user_define.h"
#define STDOUT 0xd0580000
.globl _start
.section .text
_start:
                   .include "user_init.s"
                   csrr x5, 0xf14
                   li x6, 0
                   beq x5, x6, 0f

0: la x21, h0_start
jalr x0, x21, 0
h0_start:
                   li x10, 0x40001104
                   csrw 0x301, x10
kernel_sp:
                   la x12, kernel_stack_end

trap_vec_init:
                   la x10, mtvec_handler
                   ori x10, x10, 0
                   csrw 0x305, x10 # MTVEC

mepc_setup:
                   la x10, init
                   csrw 0x341, x10

custom_csr_setup:
                   nop

init_machine_mode:
                   li x10, 0x1800
                   csrw 0x300, x10 # MSTATUS
                   li x10, 0x0
                   csrw 0x304, x10 # MIE
                   mret
init:
                   li x0, 0xf2a29fc8
                   li x1, 0x0
                   li x2, 0x0
                   li x3, 0xf55ae986
                   li x4, 0x80000000
                   li x5, 0x6
                   li x6, 0x3
                   li x7, 0xf4f77d4b
                   li x8, 0xb
                   li x9, 0x2e3a97b
                   li x10, 0x2
                   li x11, 0xd3867231
                   li x13, 0x304a526d
                   li x14, 0xa4a7ca8
                   li x15, 0x0
                   li x16, 0x0
                   li x17, 0x80000000
                   li x18, 0x80000000
                   li x19, 0x73c42ca1
                   li x20, 0x0
                   li x21, 0x8cbc6cda
                   li x22, 0xc
                   li x23, 0x80000000
                   li x24, 0x0
                   li x25, 0x0
                   li x27, 0x0
                   li x28, 0x35fbfb61
                   li x29, 0xfcf45d11
                   li x30, 0x5
                   li x31, 0xd
                   la x26, user_stack_end
main:              or          t4, a5, t3
                   c.srai      a3, 15
                   slt         t1, zero, t6
                   sub         s8, a3, t2
                   sub         s1, s8, t4
                   c.addi      tp, 31
                   slti        sp, t3, -585
                   or          s3, s4, s1
                   divu        a5, s0, a5
                   c.addi4spn s1, sp, 208
```

RISC-V TEST COMPILATION

- To generate executables and hex program file to load in Core RAM, following command is given:

  “**make gcc_compile TEST=riscv_arithmetic_basic_test SEED=1**”

- Screenshot of generated files are shown in fig:



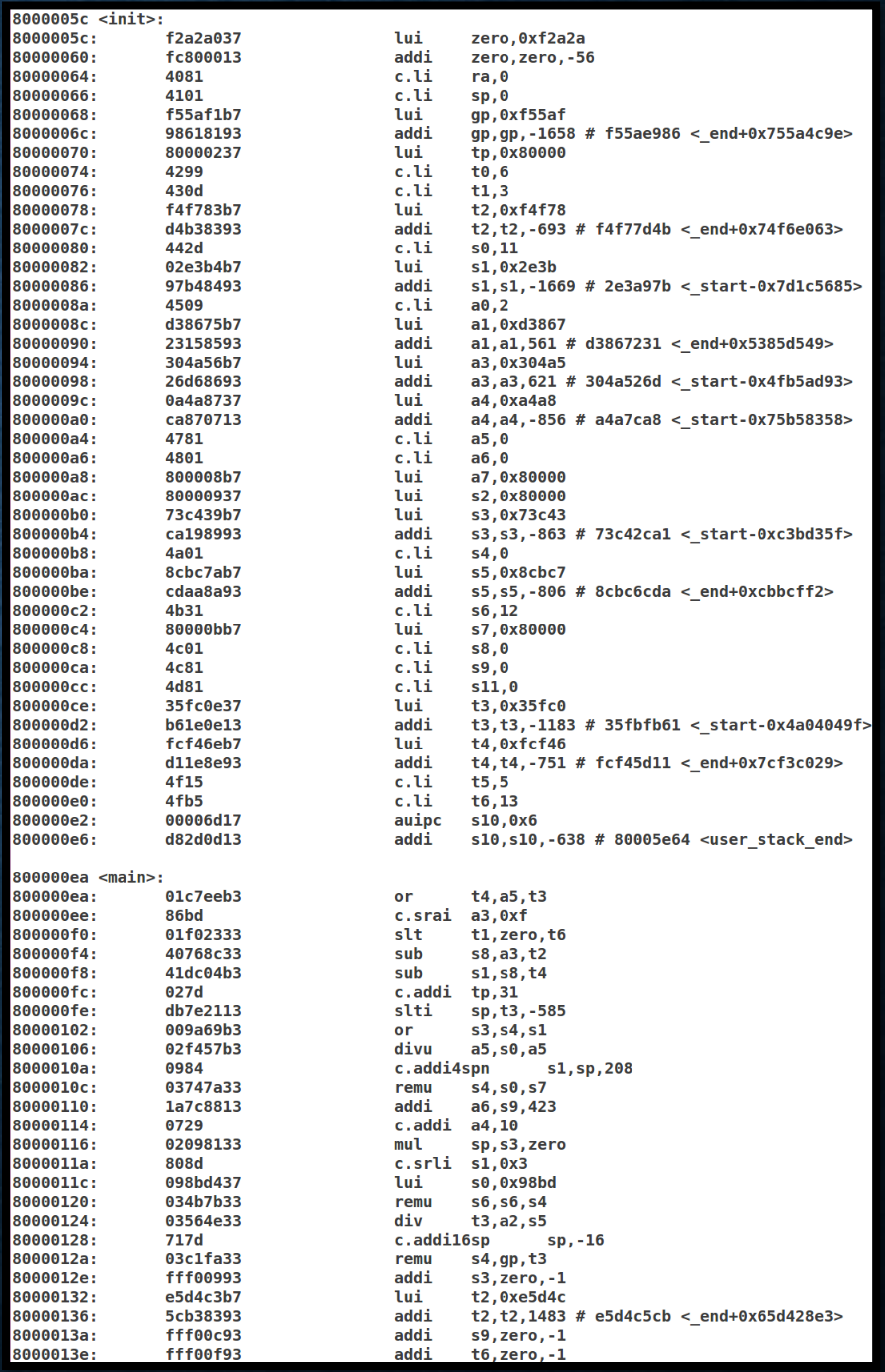
program_riscv_arithmetic_basic_test_0.hex
riscv_arithmetic_basic_test_0.bin
riscv_arithmetic_basic_test_0.dump
riscv_arithmetic_basic_test_0.exe
riscv_arithmetic_basic_test_0.o

RISC-V TEST COMPILATION

- To generate executables and hex program file to load in Core RAM, following command is given:

  "**make gcc_compile TEST=riscv_arithmetic_basic_test SEED=1**"

- Screenshot of generated test **dump** file is shown in fig:



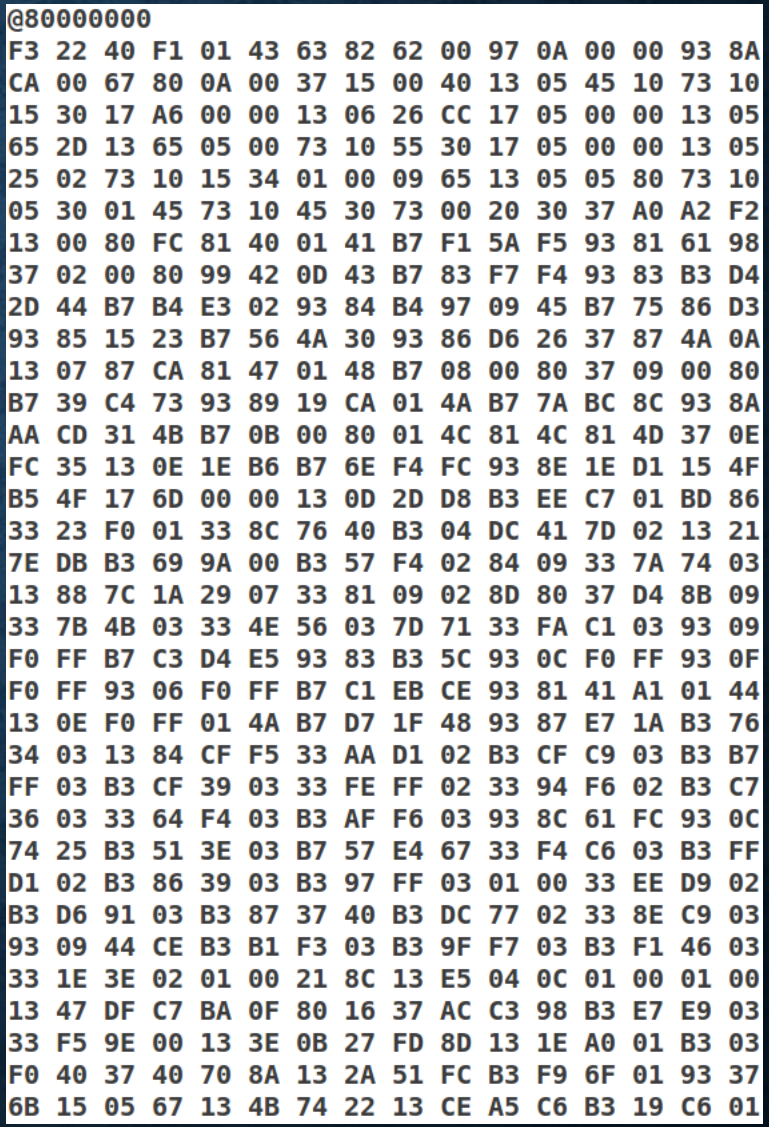
```
8000005c <init>:
8000005c:       f2a2a037                lui     zero,0xf2a2a
80000060:       fc800013                addi    zero,zero,-56
80000064:       4081                    c.li    ra,0
80000066:       4101                    c.li    sp,0
80000068:       f55af1b7                lui     gp,0xf55af
8000006c:       98618193                addi    gp,gp,-1658 # f55ae986 <_end+0x755a4c9e>
80000070:       80000237                lui     tp,0x80000
80000074:       4299                    c.li    t0,6
80000076:       430d                    c.li    t1,3
80000078:       f4f783b7                lui     t2,0xf4f78
8000007c:       d4b38393                addi    t2,t2,-693 # f4f77d4b <_end+0x74f6e063>
80000080:       442d                    c.li    s0,11
80000082:       02e3b4b7                lui     s1,0x2e3b
80000086:       97b48493                addi    s1,s1,-1669 # 2e3a97b <_start-0x7d1c5685>
8000008a:       4509                    c.li    a0,2
8000008c:       d38675b7                lui     a1,0xd3867
80000090:       23158593                addi    a1,a1,561 # d3867231 <_end+0x5385d549>
80000094:       304a56b7                lui     a3,0x304a5
80000098:       26d68693                addi    a3,a3,621 # 304a526d <_start-0x4fb5ad93>
8000009c:       0a4a8737                lui     a4,0xa4a8
800000a0:       ca870713                addi    a4,a4,-856 # a4a7ca8 <_start-0x75b58358>
800000a4:       4781                    c.li    a5,0
800000a6:       4801                    c.li    a6,0
800000a8:       800008b7                lui     a7,0x80000
800000ac:       80000937                lui     s2,0x80000
800000b0:       73c439b7                lui     s3,0x73c43
800000b4:       ca198993                addi    s3,s3,-863 # 73c42ca1 <_start-0xc3bd35f>
800000b8:       4a01                    c.li    s4,0
800000ba:       8cbc7ab7                lui     s5,0x8cbc7
800000be:       cdaa8a93                addi    s5,s5,-806 # 8cbc6cda <_end+0xcbbcff2>
800000c2:       4b31                    c.li    s6,12
800000c4:       80000bb7                lui     s7,0x80000
800000c8:       4c01                    c.li    s8,0
800000ca:       4c81                    c.li    s9,0
800000cc:       4d81                    c.li    s11,0
800000ce:       35fc0e37                lui     t3,0x35fc0
800000d2:       b61e0e13                addi    t3,t3,-1183 # 35fbfb61 <_start-0x4a04049f>
800000d6:       fcf46eb7                lui     t4,0xfcf46
800000da:       d11e8e93                addi    t4,t4,-751 # fcf45d11 <_end+0x7cf3c029>
800000de:       4f15                    c.li    t5,5
800000e0:       4fb5                    c.li    t6,13
800000e2:       00006d17                auipc   s10,0x6
800000e6:       d82d0d13                addi    s10,s10,-638 # 80005e64 <user_stack_end>

800000ea <main>:
800000ea:       01c7eeb3                or      t4,a5,t3
800000ee:       86bd                    c.srai  a3,0xf
800000f0:       01f02333                slt     t1,zero,t6
800000f4:       40768c33                sub     s8,a3,t2
800000f8:       41dc04b3                sub     s1,s8,t4
800000fc:       027d                    c.addi  tp,31
800000fe:       db7e2113                slti    sp,t3,-585
80000102:       009a69b3                or      s3,s4,s1
80000106:       02f457b3                divu    a5,s0,a5
8000010a:       0984                    c.addi4spn      s1,sp,208
8000010c:       03747a33                remu    s4,s0,s7
80000110:       1a7c8813                addi    a6,s9,423
80000114:       0729                    c.addi  a4,10
80000116:       02098133                mul     sp,s3,zero
8000011a:       808d                    c.srli  s1,0x3
8000011c:       098bd437                lui     s0,0x98bd
80000120:       034b7b33                remu    s6,s6,s4
80000124:       03564e33                div     t3,a2,s5
80000128:       717d                    c.addi16sp      sp,-16
8000012a:       03c1fa33                remu    s4,gp,t3
8000012e:       fff00993                addi    s3,zero,-1
80000132:       e5d4c3b7                lui     t2,0xe5d4c
80000136:       5cb38393                addi    t2,t2,1483 # e5d4c5cb <_end+0x65d428e3>
8000013a:       fff00c93                addi    s9,zero,-1
8000013e:       fff00f93                addi    t6,zero,-1
```

RISC-V TEST COMPILATION

- To generate executables and hex program file to load in Core RAM, following command is given:

  "**make gcc_compile TEST=riscv_arithmetic_basic_test SEED=1**"

- Screenshot of generated **program_hex** file is shown in fig:



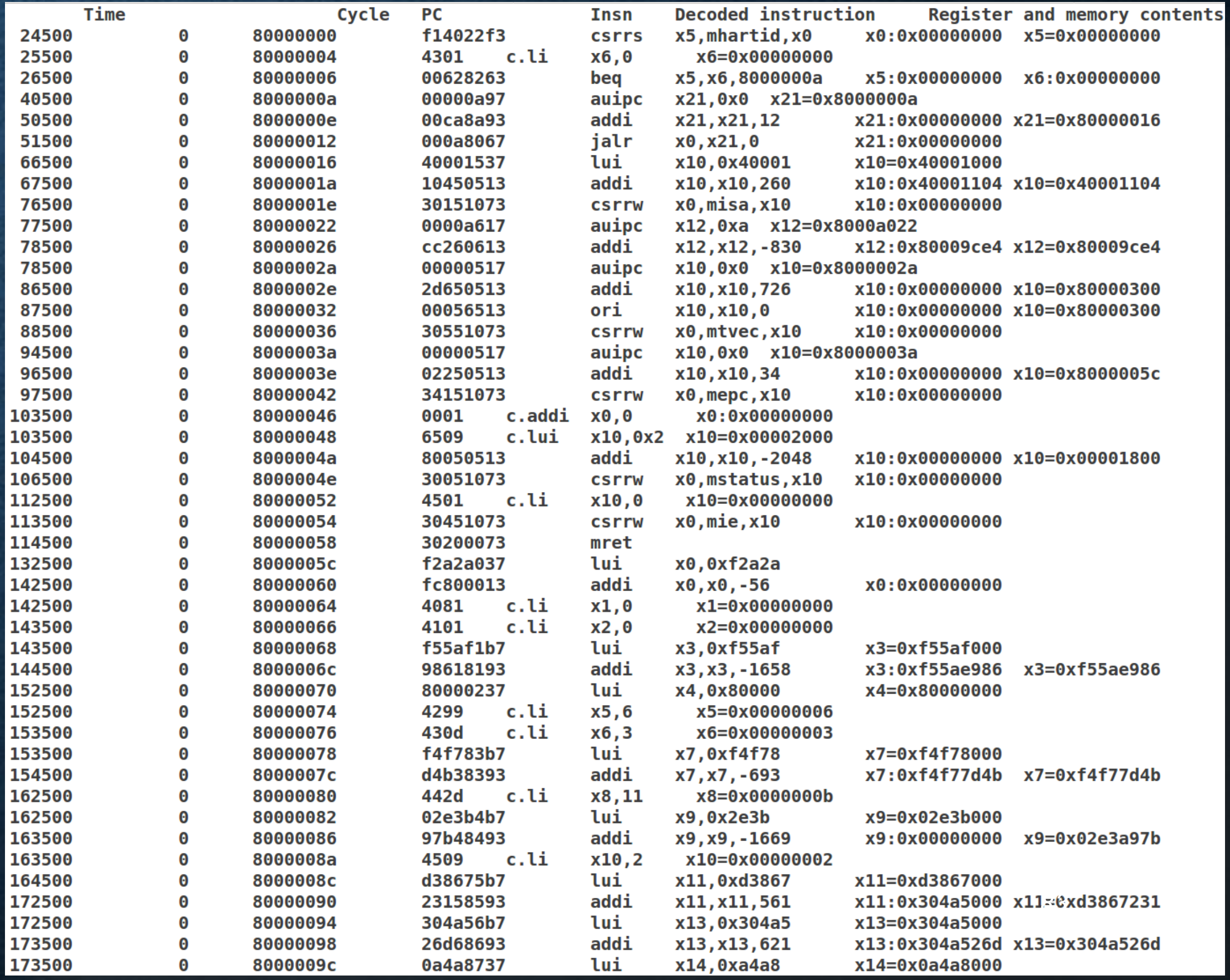
```
@80000000
F3 22 40 F1 01 43 63 82 62 00 97 0A 00 00 93 8A
CA 00 67 80 0A 00 37 15 00 40 13 05 45 10 73 10
15 30 17 A6 00 00 13 06 26 CC 17 05 00 00 13 05
65 2D 13 65 05 00 73 10 55 30 17 05 00 00 13 05
25 02 73 10 15 34 01 00 09 65 13 05 05 80 73 10
05 30 01 45 73 10 45 30 73 00 20 30 37 A0 A2 F2
13 00 80 FC 81 40 01 41 B7 F1 5A F5 93 81 61 98
37 02 00 80 99 42 0D 43 B7 83 F7 F4 93 83 B3 D4
2D 44 B7 B4 E3 02 93 84 B4 97 09 45 B7 75 86 D3
93 85 15 23 B7 56 4A 30 93 86 D6 26 37 87 4A 0A
13 07 87 CA 81 47 01 48 B7 08 00 80 37 09 00 80
B7 39 C4 73 93 89 19 CA 01 4A B7 7A BC 8C 93 8A
AA CD 31 4B B7 0B 00 80 01 4C 81 4C 81 4D 37 0E
FC 35 13 0E 1E B6 B7 6E F4 FC 93 8E 1E D1 15 4F
B5 4F 17 6D 00 00 13 0D 2D D8 B3 EE C7 01 BD 86
33 23 F0 01 33 8C 76 40 B3 04 DC 41 7D 02 13 21
7E DB B3 69 9A 00 B3 57 F4 02 84 09 33 7A 74 03
13 88 7C 1A 29 07 33 81 09 02 8D 80 37 D4 8B 09
33 7B 4B 03 33 4E 56 03 7D 71 33 FA C1 03 93 09
F0 FF B7 C3 D4 E5 93 83 B3 5C 93 0C F0 FF 93 0F
F0 FF 93 06 F0 FF B7 C1 EB CE 93 81 41 A1 01 44
13 0E F0 FF 01 4A B7 D7 1F 48 93 87 E7 1A B3 76
34 03 13 84 CF F5 33 AA D1 02 B3 CF C9 03 B3 B7
FF 03 B3 CF 39 03 33 FE FF 02 33 94 F6 02 B3 C7
36 03 33 64 F4 03 B3 AF F6 03 93 8C 61 FC 93 0C
74 25 B3 51 3E 03 B7 57 E4 67 33 F4 C6 03 B3 FF
D1 02 B3 86 39 03 B3 97 FF 03 01 00 33 EE D9 02
B3 D6 91 03 B3 87 37 40 B3 DC 77 02 33 8E C9 03
93 09 44 CE B3 B1 F3 03 B3 9F F7 03 B3 F1 46 03
33 1E 3E 02 01 00 21 8C 13 E5 04 0C 01 00 01 00
13 47 DF C7 BA 0F 80 16 37 AC C3 98 B3 E7 E9 03
33 F5 9E 00 13 3E 0B 27 FD 8D 13 1E A0 01 B3 03
F0 40 37 40 70 8A 13 2A 51 FC B3 F9 6F 01 93 37
6B 15 05 67 13 4B 74 22 13 CE A5 C6 B3 19 C6 01
```

RTL TEST SIMULATION

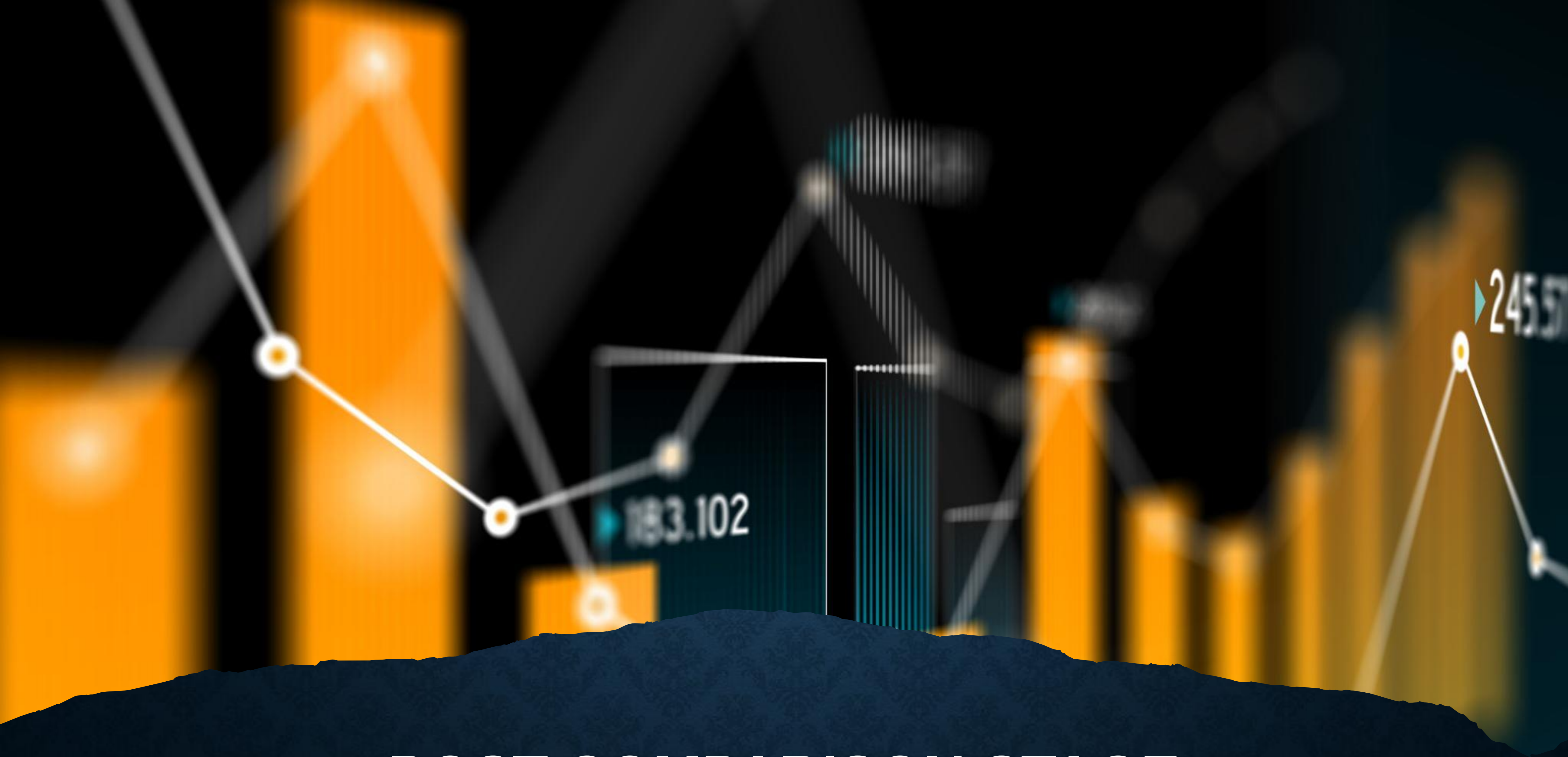
- To run the generated program in hex file on core, run the following command:

  “**make rtl_sim TEST=riscv_arithmetic_basic_test SEED=1**”

- Screenshot of generated **core_trace** log is shown in fig:



```
      Time                      Cycle  PC              Insn    Decoded instruction     Register and memory contents
 24500           0      80000000       f14022f3        csrrs   x5,mhartid,x0     x0:0x00000000  x5=0x00000000
 25500           0      80000004       4301     c.li   x6,0      x6=0x00000000
 26500           0      80000006       00628263        beq     x5,x6,8000000a    x5:0x00000000  x6:0x00000000
 40500           0      8000000a       00000a97        auipc   x21,0x0  x21=0x8000000a
 50500           0      8000000e       00ca8a93        addi    x21,x21,12       x21:0x00000000 x21=0x80000016
 51500           0      80000012       000a8067        jalr    x0,x21,0         x21:0x00000000
 66500           0      80000016       40001537        lui     x10,0x40001      x10=0x40001000
 67500           0      8000001a       10450513        addi    x10,x10,260      x10:0x40001104 x10=0x40001104
 76500           0      8000001e       30151073        csrrw   x0,misa,x10      x10:0x00000000
 77500           0      80000022       0000a617        auipc   x12,0xa  x12=0x8000a022
 78500           0      80000026       cc260613        addi    x12,x12,-830     x12:0x80009ce4 x12=0x80009ce4
 78500           0      8000002a       00000517        auipc   x10,0x0  x10=0x8000002a
 86500           0      8000002e       2d650513        addi    x10,x10,726      x10:0x00000000 x10=0x80000300
 87500           0      80000032       00056513        ori     x10,x10,0        x10:0x00000000 x10=0x80000300
 88500           0      80000036       30551073        csrrw   x0,mtvec,x10     x10:0x00000000
 94500           0      8000003a       00000517        auipc   x10,0x0  x10=0x8000003a
 96500           0      8000003e       02250513        addi    x10,x10,34       x10:0x00000000 x10=0x8000005c
 97500           0      80000042       34151073        csrrw   x0,mepc,x10      x10:0x00000000
103500           0      80000046       0001     c.addi x0,0      x0:0x00000000
103500           0      80000048       6509     c.lui  x10,0x2  x10=0x00002000
104500           0      8000004a       80050513        addi    x10,x10,-2048    x10:0x00000000 x10=0x00001800
106500           0      8000004e       30051073        csrrw   x0,mstatus,x10   x10:0x00000000
112500           0      80000052       4501     c.li   x10,0    x10=0x00000000
113500           0      80000054       30451073        csrrw   x0,mie,x10       x10:0x00000000
114500           0      80000058       30200073        mret
132500           0      8000005c       f2a2a037        lui     x0,0xf2a2a
142500           0      80000060       fc800013        addi    x0,x0,-56         x0:0x00000000
142500           0      80000064       4081     c.li   x1,0      x1=0x00000000
143500           0      80000066       4101     c.li   x2,0      x2=0x00000000
143500           0      80000068       f55af1b7        lui     x3,0xf55af        x3=0xf55af000
144500           0      8000006c       98618193        addi    x3,x3,-1658       x3:0xf55ae986  x3=0xf55ae986
152500           0      80000070       80000237        lui     x4,0x80000        x4=0x80000000
152500           0      80000074       4299     c.li   x5,6      x5=0x00000006
153500           0      80000076       430d     c.li   x6,3      x6=0x00000003
153500           0      80000078       f4f783b7        lui     x7,0xf4f78        x7=0xf4f78000
154500           0      8000007c       d4b38393        addi    x7,x7,-693        x7:0xf4f77d4b  x7=0xf4f77d4b
162500           0      80000080       442d     c.li   x8,11     x8=0x0000000b
162500           0      80000082       02e3b4b7        lui     x9,0x2e3b         x9=0x02e3b000
163500           0      80000086       97b48493        addi    x9,x9,-1669       x9:0x00000000  x9=0x02e3a97b
163500           0      8000008a       4509     c.li   x10,2    x10=0x00000002
164500           0      8000008c       d38675b7        lui     x11,0xd3867      x11=0xd3867000
172500           0      80000090       23158593        addi    x11,x11,561      x11:0x304a5000 x11=0xd3867231
172500           0      80000094       304a56b7        lui     x13,0x304a5      x13=0x304a5000
173500           0      80000098       26d68693        addi    x13,x13,621      x13:0x304a526d x13=0x304a526d
173500           0      8000009c       0a4a8737        lui     x14,0xa4a8       x14=0x0a4a8000
```



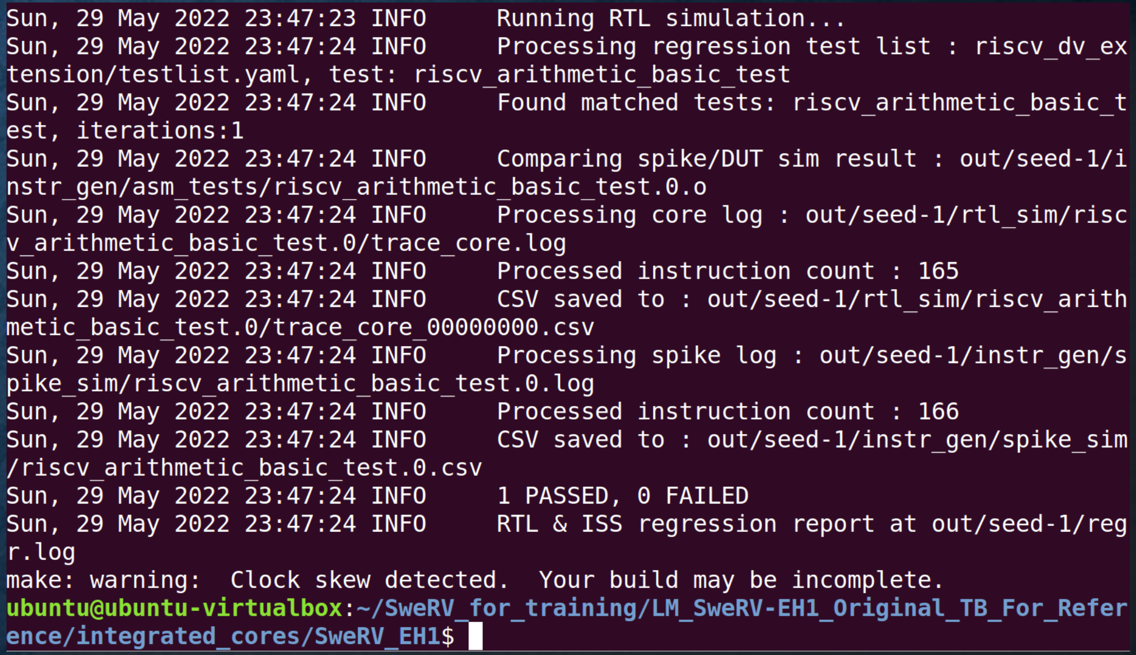
POST COMPARISON STAGE

POST-COMPARISON

- To run the same generated program in hex file on spike, run the following command:

  “**make post_compare TEST=riscv_arithmetic_basic_test SEED=1**”

- The command makes .csv files from both core & spike logs, compares them and generate final regression log.
- Screenshot of result of run test on terminal is shown in fig:



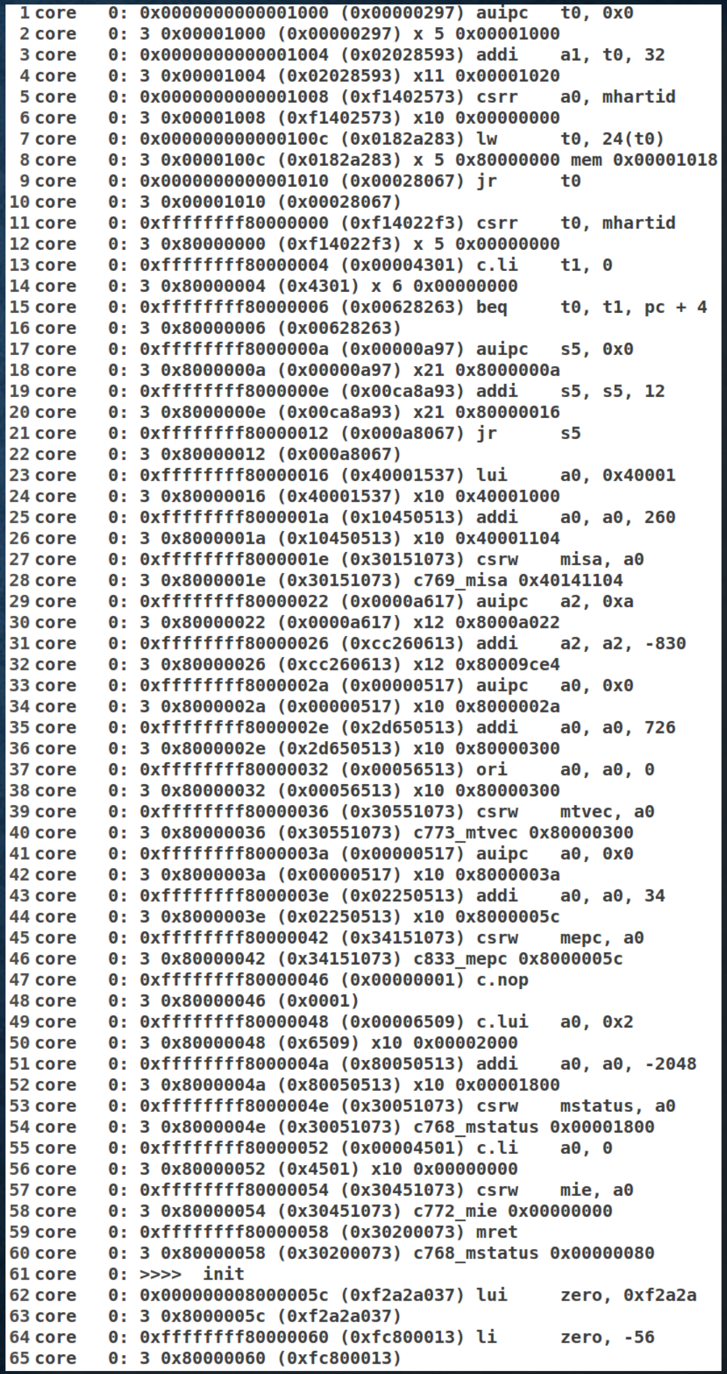
```
Sun, 29 May 2022 23:47:23 INFO     Running RTL simulation...
Sun, 29 May 2022 23:47:24 INFO     Processing regression test list : riscv_dv_ex
tension/testlist.yaml, test: riscv_arithmetic_basic_test
Sun, 29 May 2022 23:47:24 INFO     Found matched tests: riscv_arithmetic_basic_t
est, iterations:1
Sun, 29 May 2022 23:47:24 INFO     Comparing spike/DUT sim result : out/seed-1/i
nstr_gen/asm_tests/riscv_arithmetic_basic_test.0.o
Sun, 29 May 2022 23:47:24 INFO     Processing core log : out/seed-1/rtl_sim/risc
v_arithmetic_basic_test.0/trace_core.log
Sun, 29 May 2022 23:47:24 INFO     Processed instruction count : 165
Sun, 29 May 2022 23:47:24 INFO     CSV saved to : out/seed-1/rtl_sim/riscv_arith
metic_basic_test.0/trace_core_00000000.csv
Sun, 29 May 2022 23:47:24 INFO     Processing spike log : out/seed-1/instr_gen/s
pike_sim/riscv_arithmetic_basic_test.0.log
Sun, 29 May 2022 23:47:24 INFO     Processed instruction count : 166
Sun, 29 May 2022 23:47:24 INFO     CSV saved to : out/seed-1/instr_gen/spike_sim
/riscv_arithmetic_basic_test.0.csv
Sun, 29 May 2022 23:47:24 INFO     1 PASSED, 0 FAILED
Sun, 29 May 2022 23:47:24 INFO     RTL & ISS regression report at out/seed-1/reg
r.log
make: warning:  Clock skew detected.  Your build may be incomplete.
ubuntu@ubuntu-virtualbox:~/SweRV_for_training/LM_SweRV-EH1_Original_TB_For_Refer
ence/integrated_cores/SweRV_EH1$
```

POST-COMPARISON

- To run the same generated program in hex file on spike, run the following command:

  “**make post_compare TEST=riscv_arithmetic_basic_test SEED=1**”

- The command makes .csv files from both core & spike logs, compares them and generate final regression log.
- Screenshot of generated **spike_log** is shown in fig:



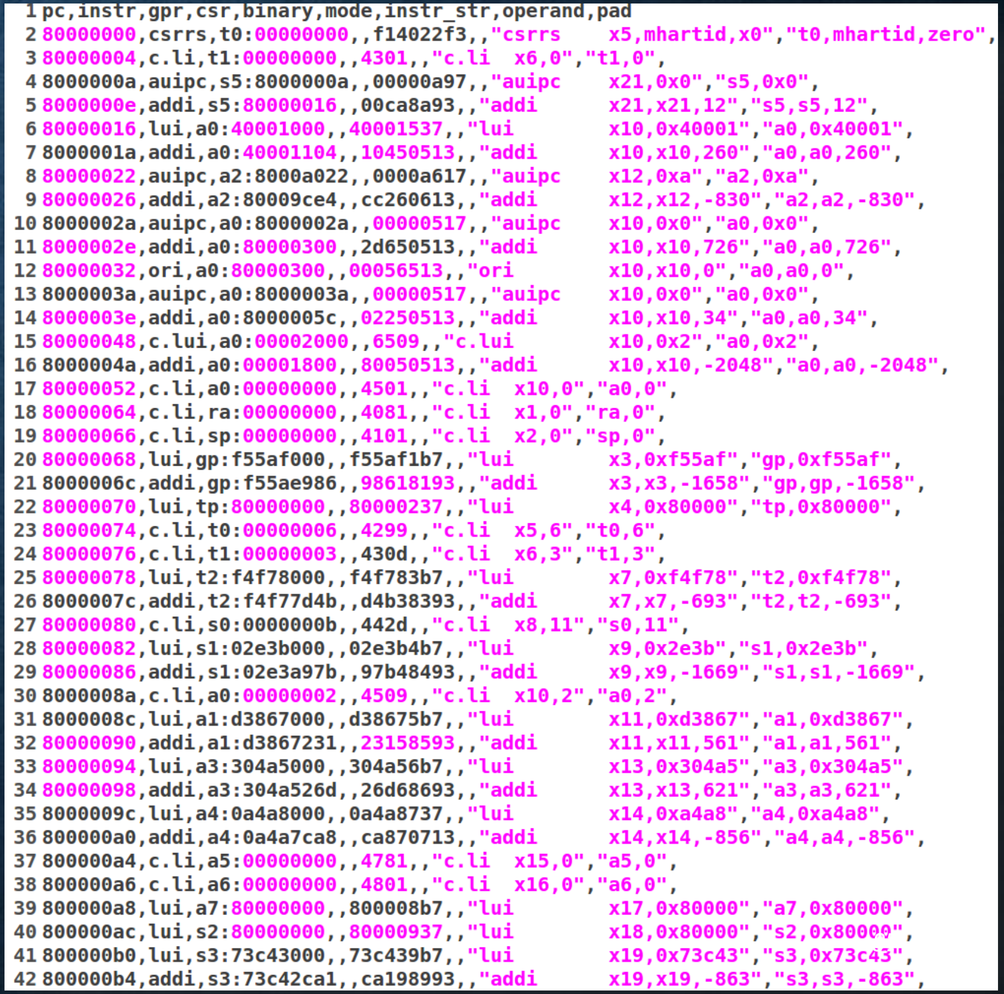
```
core   0: 0x0000000000001000 (0x00000297) auipc   t0, 0x0
core   0: 3 0x00001000 (0x00000297) x 5 0x00001000
core   0: 0x0000000000001004 (0x02028593) addi    a1, t0, 32
core   0: 3 0x00001004 (0x02028593) x11 0x00001020
core   0: 0x0000000000001008 (0xf1402573) csrr    a0, mhartid
core   0: 3 0x00001008 (0xf1402573) x10 0x00000000
core   0: 0x000000000000100c (0x0182a283) lw      t0, 24(t0)
core   0: 3 0x0000100c (0x0182a283) x 5 0x80000000 mem 0x00001018
core   0: 0x0000000000001010 (0x00028067) jr      t0
core   0: 3 0x00001010 (0x00028067)
core   0: 0xffffffff80000000 (0xf14022f3) csrr    t0, mhartid
core   0: 3 0x80000000 (0xf14022f3) x 5 0x00000000
core   0: 0xffffffff80000004 (0x00004301) c.li    t1, 0
core   0: 3 0x80000004 (0x4301) x 6 0x00000000
core   0: 0xffffffff80000006 (0x00628263) beq     t0, t1, pc + 4
core   0: 3 0x80000006 (0x00628263)
core   0: 0xffffffff8000000a (0x00000a97) auipc   s5, 0x0
core   0: 3 0x8000000a (0x00000a97) x21 0x8000000a
core   0: 0xffffffff8000000e (0x00ca8a93) addi    s5, s5, 12
core   0: 3 0x8000000e (0x00ca8a93) x21 0x80000016
core   0: 0xffffffff80000012 (0x000a8067) jr      s5
core   0: 3 0x80000012 (0x000a8067)
core   0: 0xffffffff80000016 (0x40001537) lui     a0, 0x40001
core   0: 3 0x80000016 (0x40001537) x10 0x40001000
core   0: 0xffffffff8000001a (0x10450513) addi    a0, a0, 260
core   0: 3 0x8000001a (0x10450513) x10 0x40001104
core   0: 0xffffffff8000001e (0x30151073) csrw    misa, a0
core   0: 3 0x8000001e (0x30151073) c769_misa 0x40141104
core   0: 0xffffffff80000022 (0x0000a617) auipc   a2, 0xa
core   0: 3 0x80000022 (0x0000a617) x12 0x8000a022
core   0: 0xffffffff80000026 (0xcc260613) addi    a2, a2, -830
core   0: 3 0x80000026 (0xcc260613) x12 0x80009ce4
core   0: 0xffffffff8000002a (0x00000517) auipc   a0, 0x0
core   0: 3 0x8000002a (0x00000517) x10 0x8000002a
core   0: 0xffffffff8000002e (0x2d650513) addi    a0, a0, 726
core   0: 3 0x8000002e (0x2d650513) x10 0x80000300
core   0: 0xffffffff80000032 (0x00056513) ori     a0, a0, 0
core   0: 3 0x80000032 (0x00056513) x10 0x80000300
core   0: 0xffffffff80000036 (0x30551073) csrw    mtvec, a0
core   0: 3 0x80000036 (0x30551073) c773_mtvec 0x80000300
core   0: 0xffffffff8000003a (0x00000517) auipc   a0, 0x0
core   0: 3 0x8000003a (0x00000517) x10 0x8000003a
core   0: 0xffffffff8000003e (0x02250513) addi    a0, a0, 34
core   0: 3 0x8000003e (0x02250513) x10 0x8000005c
core   0: 0xffffffff80000042 (0x34151073) csrw    mepc, a0
core   0: 3 0x80000042 (0x34151073) c833_mepc 0x8000005c
core   0: 0xffffffff80000046 (0x00000001) c.nop
core   0: 3 0x80000046 (0x0001)
core   0: 0xffffffff80000048 (0x00006509) c.lui   a0, 0x2
core   0: 3 0x80000048 (0x6509) x10 0x00002000
core   0: 0xffffffff8000004a (0x80050513) addi    a0, a0, -2048
core   0: 3 0x8000004a (0x80050513) x10 0x00001800
core   0: 0xffffffff8000004e (0x30051073) csrw    mstatus, a0
core   0: 3 0x8000004e (0x30051073) c768_mstatus 0x00001800
core   0: 0xffffffff80000052 (0x00004501) c.li    a0, 0
core   0: 3 0x80000052 (0x4501) x10 0x00000000
core   0: 0xffffffff80000054 (0x30451073) csrw    mie, a0
core   0: 3 0x80000054 (0x30451073) c772_mie 0x00000000
core   0: 0xffffffff80000058 (0x30200073) mret
core   0: 3 0x80000058 (0x30200073) c768_mstatus 0x00000080
core   0: >>>>  init
core   0: 0x000000008000005c (0xf2a2a037) lui     zero, 0xf2a2a
core   0: 3 0x8000005c (0xf2a2a037)
core   0: 0xffffffff80000060 (0xfc800013) li      zero, -56
core   0: 3 0x80000060 (0xfc800013)
```

POST-COMPARISON

- *To run the same generated program in hex file on spike, run the following command:*

  *"***make post_compare TEST=riscv_arithmetic_basic_test SEED=1***"*

- *The command makes .csv files from both core & spike logs, compares them and generate final regression log.*
- *Screenshot of generated* **core.csv** *is shown in fig:*



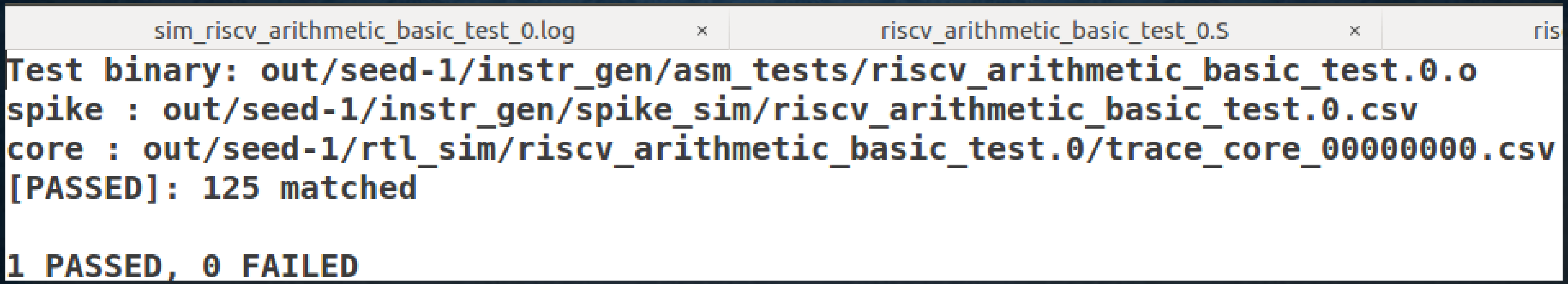
```
pc,instr,gpr,csr,binary,mode,instr_str,operand,pad
80000000,csrrs,t0:00000000,,f14022f3,,"csrrs    x5,mhartid,x0","t0,mhartid,zero",
80000004,c.li,t1:00000000,,4301,,"c.li  x6,0","t1,0",
8000000a,auipc,s5:8000000a,,00000a97,,"auipc    x21,0x0","s5,0x0",
8000000e,addi,s5:80000016,,00ca8a93,,"addi     x21,x21,12","s5,s5,12",
80000016,lui,a0:40001000,,40001537,,"lui        x10,0x40001","a0,0x40001",
8000001a,addi,a0:40001104,,10450513,,"addi     x10,x10,260","a0,a0,260",
80000022,auipc,a2:8000a022,,0000a617,,"auipc    x12,0xa","a2,0xa",
80000026,addi,a2:80009ce4,,cc260613,,"addi     x12,x12,-830","a2,a2,-830",
8000002a,auipc,a0:8000002a,,00000517,,"auipc    x10,0x0","a0,0x0",
8000002e,addi,a0:80000300,,2d650513,,"addi     x10,x10,726","a0,a0,726",
80000032,ori,a0:80000300,,00056513,,"ori        x10,x10,0","a0,a0,0",
8000003a,auipc,a0:8000003a,,00000517,,"auipc    x10,0x0","a0,0x0",
8000003e,addi,a0:8000005c,,02250513,,"addi     x10,x10,34","a0,a0,34",
80000048,c.lui,a0:00002000,,6509,,"c.lui        x10,0x2","a0,0x2",
8000004a,addi,a0:00001800,,80050513,,"addi     x10,x10,-2048","a0,a0,-2048",
80000052,c.li,a0:00000000,,4501,,"c.li  x10,0","a0,0",
80000064,c.li,ra:00000000,,4081,,"c.li  x1,0","ra,0",
80000066,c.li,sp:00000000,,4101,,"c.li  x2,0","sp,0",
80000068,lui,gp:f55af000,,f55af1b7,,"lui        x3,0xf55af","gp,0xf55af",
8000006c,addi,gp:f55ae986,,98618193,,"addi     x3,x3,-1658","gp,gp,-1658",
80000070,lui,tp:80000000,,80000237,,"lui        x4,0x80000","tp,0x80000",
80000074,c.li,t0:00000006,,4299,,"c.li  x5,6","t0,6",
80000076,c.li,t1:00000003,,430d,,"c.li  x6,3","t1,3",
80000078,lui,t2:f4f78000,,f4f783b7,,"lui        x7,0xf4f78","t2,0xf4f78",
8000007c,addi,t2:f4f77d4b,,d4b38393,,"addi     x7,x7,-693","t2,t2,-693",
80000080,c.li,s0:0000000b,,442d,,"c.li  x8,11","s0,11",
80000082,lui,s1:02e3b000,,02e3b4b7,,"lui        x9,0x2e3b","s1,0x2e3b",
80000086,addi,s1:02e3a97b,,97b48493,,"addi     x9,x9,-1669","s1,s1,-1669",
8000008a,c.li,a0:00000002,,4509,,"c.li  x10,2","a0,2",
8000008c,lui,a1:d3867000,,d38675b7,,"lui        x11,0xd3867","a1,0xd3867",
80000090,addi,a1:d3867231,,23158593,,"addi     x11,x11,561","a1,a1,561",
80000094,lui,a3:304a5000,,304a56b7,,"lui        x13,0x304a5","a3,0x304a5",
80000098,addi,a3:304a526d,,26d68693,,"addi     x13,x13,621","a3,a3,621",
8000009c,lui,a4:0a4a8000,,0a4a8737,,"lui        x14,0xa4a8","a4,0xa4a8",
800000a0,addi,a4:0a4a7ca8,,ca870713,,"addi     x14,x14,-856","a4,a4,-856",
800000a4,c.li,a5:00000000,,4781,,"c.li  x15,0","a5,0",
800000a6,c.li,a6:00000000,,4801,,"c.li  x16,0","a6,0",
800000a8,lui,a7:80000000,,800008b7,,"lui        x17,0x80000","a7,0x80000",
800000ac,lui,s2:80000000,,80000937,,"lui        x18,0x80000","s2,0x80000",
800000b0,lui,s3:73c43000,,73c439b7,,"lui        x19,0x73c43","s3,0x73c43",
800000b4,addi,s3:73c42ca1,,ca198993,,"addi     x19,x19,-863","s3,s3,-863",
```

POST-COMPARISON

- To run the same generated program in hex file on spike, run the following command:

  "**make post_compare TEST=riscv_arithmetic_basic_test SEED=1**"

- The command makes .csv files from both core & spike logs, compares them and generate final regression log.
- Screenshot of generated **regr_log** is shown in fig:


sim_riscv_arithmetic_basic_test_0.log ×
riscv_arithmetic_basic_test_0.S ×
ris

```
Test binary: out/seed-1/instr_gen/asm_tests/riscv_arithmetic_basic_test.0.o
spike : out/seed-1/instr_gen/spike_sim/riscv_arithmetic_basic_test.0.csv
core : out/seed-1/rtl_sim/riscv_arithmetic_basic_test.0/trace_core_00000000.csv
[PASSED]: 125 matched

1 PASSED, 0 FAILED
```

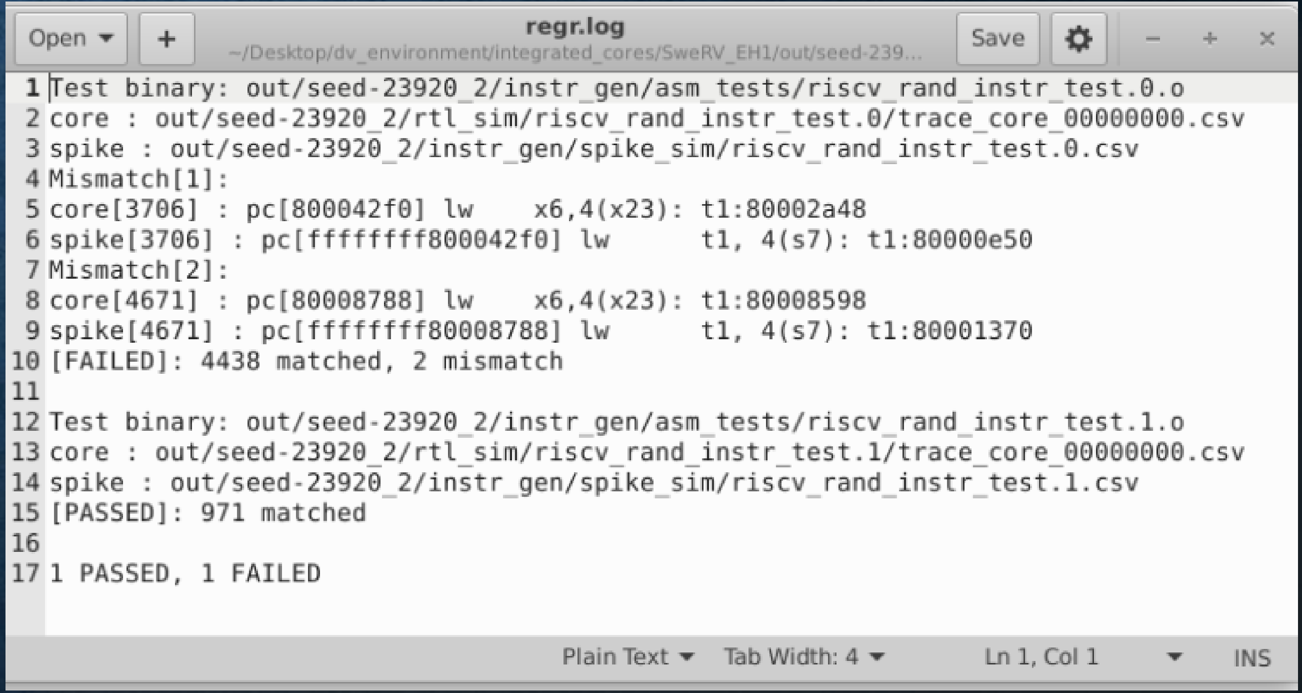
POST-COMPARISON

- *To run the same generated program in hex file on spike, run the following command:*

  "**make post_compare TEST=riscv_arithmetic_basic_test SEED=1**"

- *The command makes .csv files from both core & spike logs, compares them and generate final regression log.*

- *Screenshot of generated* **regr_log [In case of any mismatches found]** *is shown in fig:*


Open
+
regr.log
~/Desktop/dv_environment/integrated_cores/SweRV_EH1/out/seed-239...
Save
Plain Text
Tab Width: 4
Ln 1, Col 1
INS

```
Test binary: out/seed-23920_2/instr_gen/asm_tests/riscv_rand_instr_test.0.o
core : out/seed-23920_2/rtl_sim/riscv_rand_instr_test.0/trace_core_00000000.csv
spike : out/seed-23920_2/instr_gen/spike_sim/riscv_rand_instr_test.0.csv
Mismatch[1]:
core[3706] : pc[800042f0] lw    x6,4(x23): t1:80002a48
spike[3706] : pc[ffffffff800042f0] lw      t1, 4(s7): t1:80000e50
Mismatch[2]:
core[4671] : pc[80008788] lw    x6,4(x23): t1:80008598
spike[4671] : pc[ffffffff80008788] lw      t1, 4(s7): t1:80001370
[FAILED]: 4438 matched, 2 mismatch

Test binary: out/seed-23920_2/instr_gen/asm_tests/riscv_rand_instr_test.1.o
core : out/seed-23920_2/rtl_sim/riscv_rand_instr_test.1/trace_core_00000000.csv
spike : out/seed-23920_2/instr_gen/spike_sim/riscv_rand_instr_test.1.csv
[PASSED]: 971 matched

1 PASSED, 1 FAILED
```

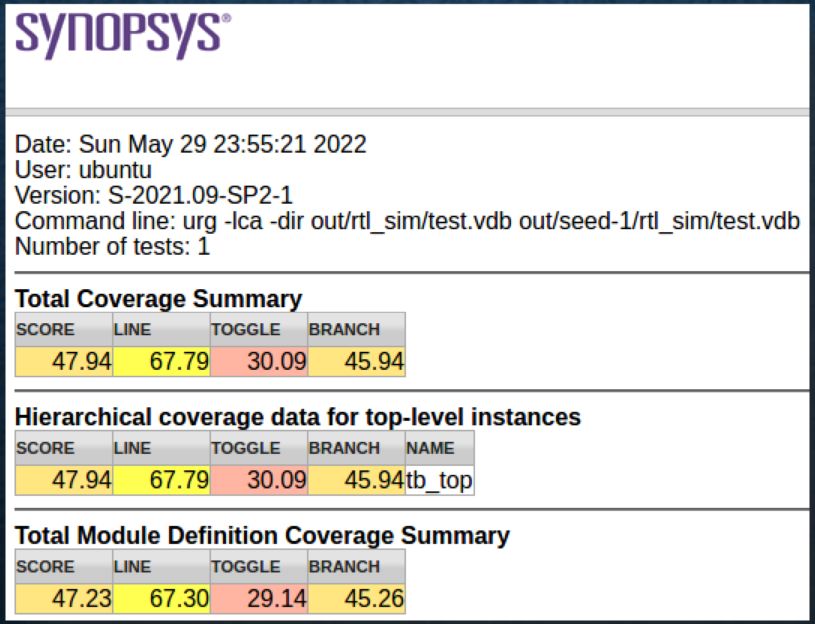
CORE COVERAGE

CORE CODE COVERAGE [HTML]

- To see how much code coverage is achieved by running the following command:

  “make cov_urg_all”

- Screenshot of **html-based** code coverage & no. of tests run on core is shown in fig:



Date: Sun May 29 23:55:21 2022
User: ubuntu
Version: S-2021.09-SP2-1
Command line: urg -lca -dir out/rtl_sim/test.vdb out/seed-1/rtl_sim/test.vdb
Number of tests: 1

Total Coverage Summary

SCORE	LINE	TOGGLE	BRANCH
47.94	67.79	30.09	45.94

Hierarchical coverage data for top-level instances

SCORE	LINE	TOGGLE	BRANCH	NAME
47.94	67.79	30.09	45.94	tb_top

Total Module Definition Coverage Summary

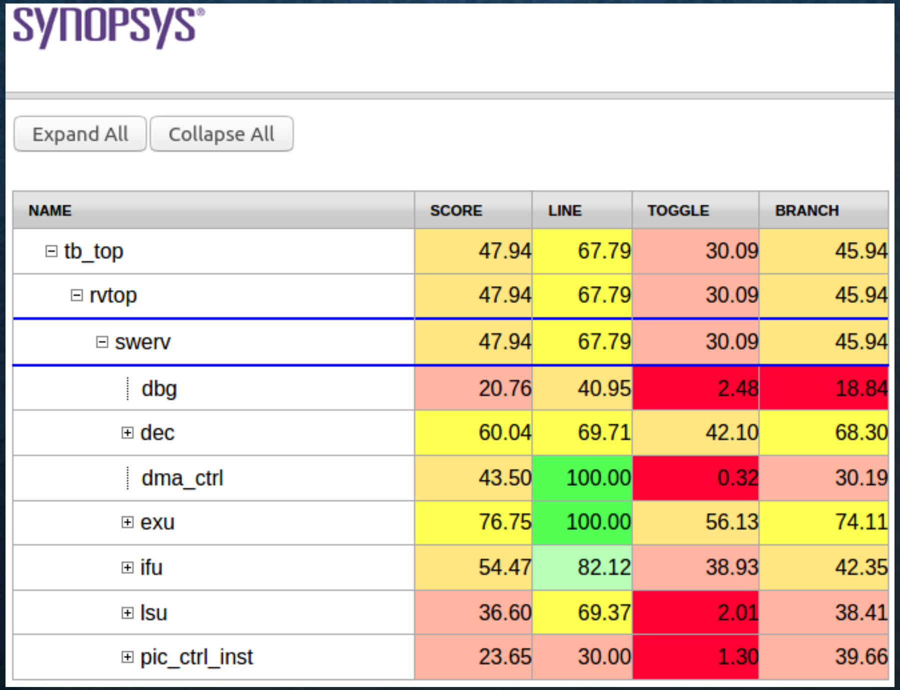
SCORE	LINE	TOGGLE	BRANCH
47.23	67.30	29.14	45.26

CORE CODE COVERAGE [HTML]

- To see how much code coverage is achieved by running the riscv_arithmetic_basic_test, run the following cmnd:

  "**make cov_urg_all**"

- Screenshot of html-based **detailed code_coverage** of core is shown in fig:



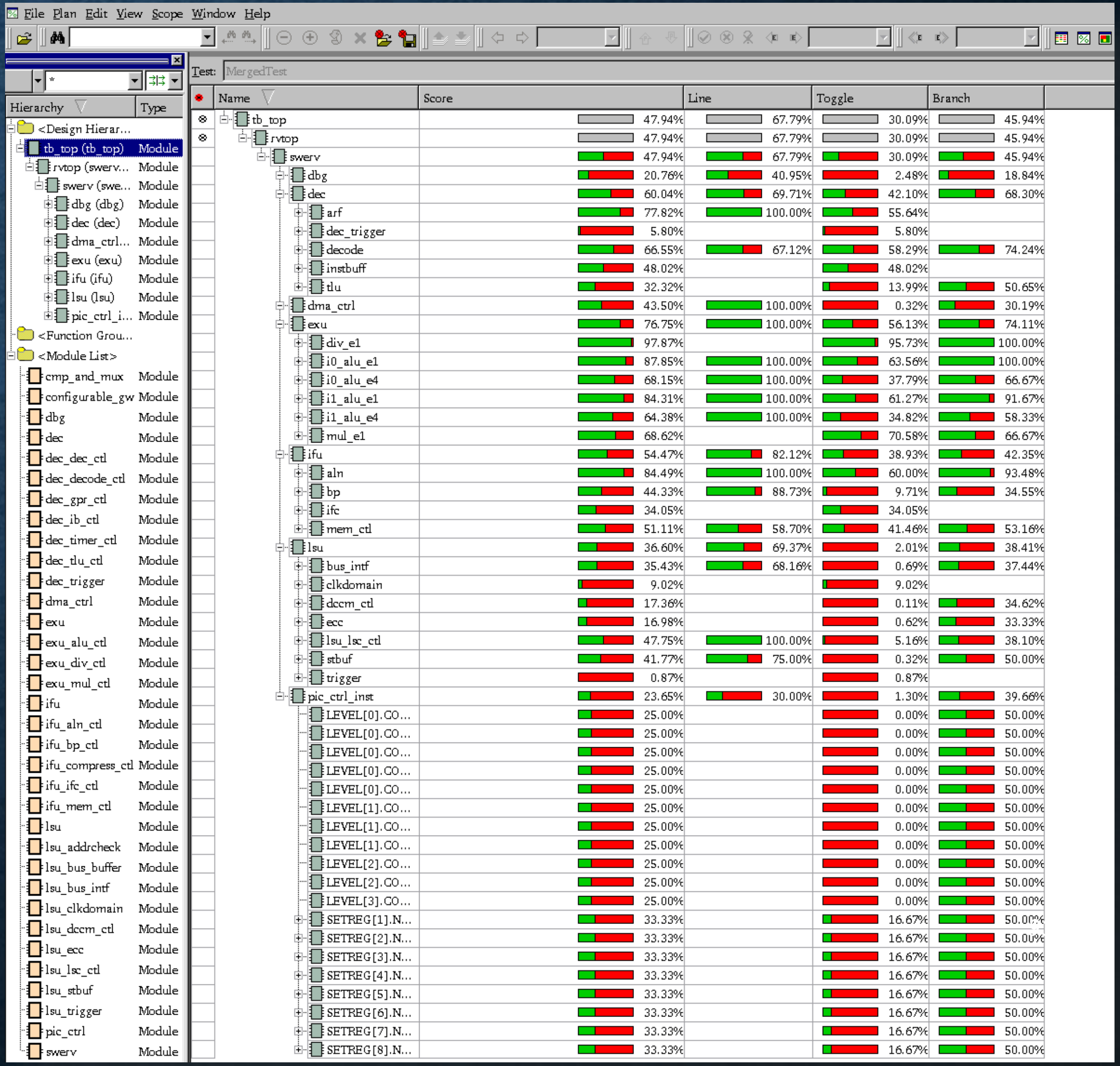
NAME	SCORE	LINE	TOGGLE	BRANCH
⊟ tb_top	47.94	67.79	30.09	45.94
⊟ rvtop	47.94	67.79	30.09	45.94
⊟ swerv	47.94	67.79	30.09	45.94
dbg	20.76	40.95	2.48	18.84
⊞ dec	60.04	69.71	42.10	68.30
dma_ctrl	43.50	100.00	0.32	30.19
⊞ exu	76.75	100.00	56.13	74.11
⊞ ifu	54.47	82.12	38.93	42.35
⊞ lsu	36.60	69.37	2.01	38.41
⊞ pic_ctrl_inst	23.65	30.00	1.30	39.66

CORE CODE COVERAGE [DVE]

- To see how much code coverage is achieved by running the following command:

  "make cov_all"

- Screenshot of **DVE-based** detailed **code_coverage** of core is shown in fig:



Hierarchy	Type
<Design Hierar...	
tb_top (tb_top)	Module
rvtop (swerv...	Module
swerv (swe...	Module
dbg (dbg)	Module
dec (dec)	Module
dma_ctrl...	Module
exu (exu)	Module
ifu (ifu)	Module
lsu (lsu)	Module
pic_ctrl_i...	Module
<Function Grou...	
<Module List>	
cmp_and_mux	Module
configurable_gw	Module
dbg	Module
dec	Module
dec_dec_ctl	Module
dec_decode_ctl	Module
dec_gpr_ctl	Module
dec_ib_ctl	Module
dec_timer_ctl	Module
dec_tlu_ctl	Module
dec_trigger	Module
dma_ctrl	Module
exu	Module
exu_alu_ctl	Module
exu_div_ctl	Module
exu_mul_ctl	Module
ifu	Module
ifu_aln_ctl	Module
ifu_bp_ctl	Module
ifu_compress_ctl	Module
ifu_ifc_ctl	Module
ifu_mem_ctl	Module
lsu	Module
lsu_addrcheck	Module
lsu_bus_buffer	Module
lsu_bus_intf	Module
lsu_clkdomain	Module
lsu_dccm_ctl	Module
lsu_ecc	Module
lsu_lsc_ctl	Module
lsu_stbuf	Module
lsu_trigger	Module
pic_ctrl	Module
swerv	Module

Test: MergedTest

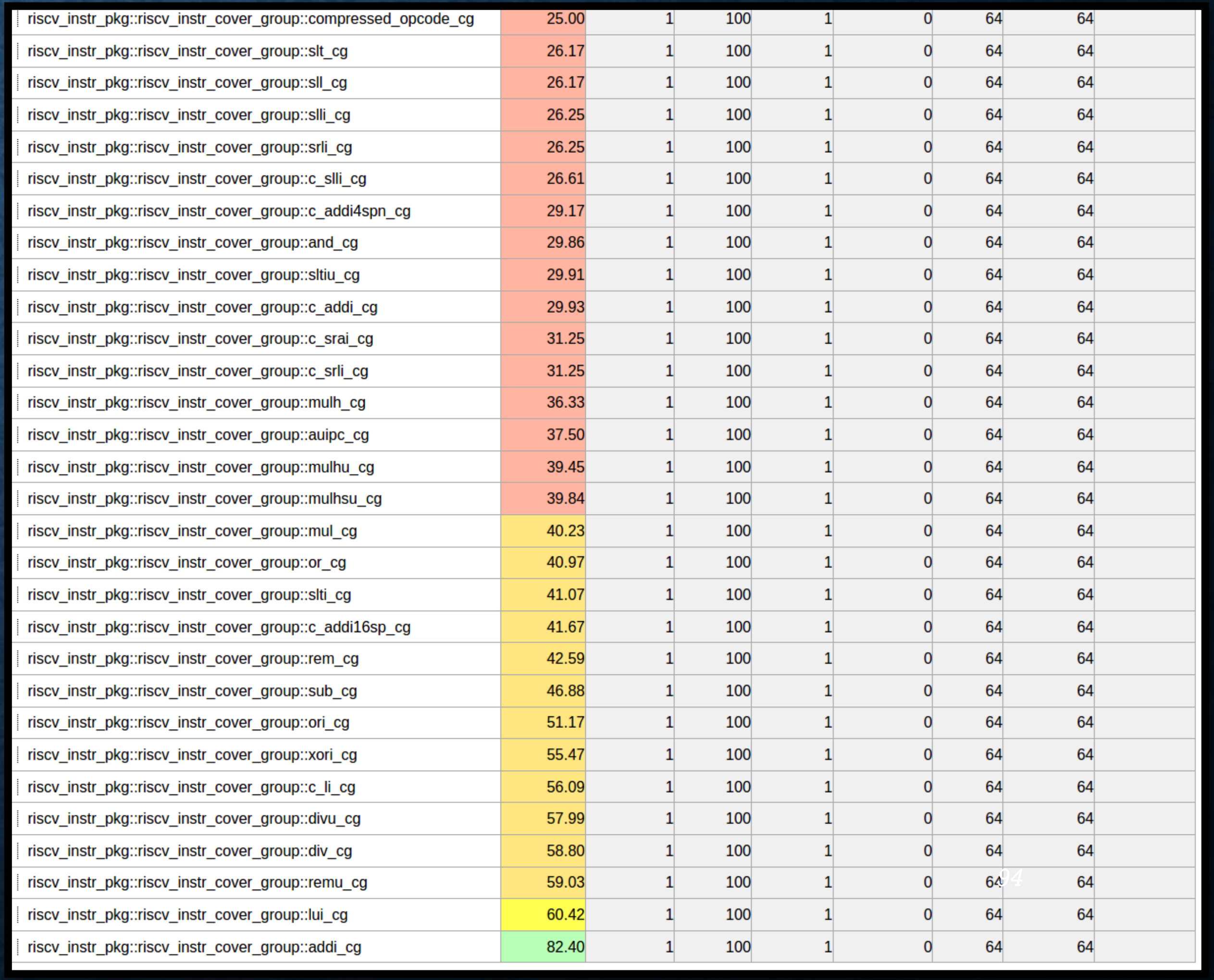
Name	Score	Line	Toggle	Branch
tb_top	47.94%	67.79%	30.09%	45.94%
rvtop	47.94%	67.79%	30.09%	45.94%
swerv	47.94%	67.79%	30.09%	45.94%
dbg	20.76%	40.95%	2.48%	18.84%
dec	60.04%	69.71%	42.10%	68.30%
arf	77.82%	100.00%	55.64%	
dec_trigger	5.80%		5.80%	
decode	66.55%	67.12%	58.29%	74.24%
instbuff	48.02%		48.02%	
tlu	32.32%		13.99%	50.65%
dma_ctrl	43.50%	100.00%	0.32%	30.19%
exu	76.75%	100.00%	56.13%	74.11%
div_e1	97.87%		95.73%	100.00%
i0_alu_e1	87.85%	100.00%	63.56%	100.00%
i0_alu_e4	68.15%	100.00%	37.79%	66.67%
i1_alu_e1	84.31%	100.00%	61.27%	91.67%
i1_alu_e4	64.38%	100.00%	34.82%	58.33%
mul_e1	68.62%		70.58%	66.67%
ifu	54.47%	82.12%	38.93%	42.35%
aln	84.49%	100.00%	60.00%	93.48%
bp	44.33%	88.73%	9.71%	34.55%
ifc	34.05%		34.05%	
mem_ctl	51.11%	58.70%	41.46%	53.16%
lsu	36.60%	69.37%	2.01%	38.41%
bus_intf	35.43%	68.16%	0.69%	37.44%
clkdomain	9.02%		9.02%	
dccm_ctl	17.36%		0.11%	34.62%
ecc	16.98%		0.62%	33.33%
lsu_lsc_ctl	47.75%	100.00%	5.16%	38.10%
stbuf	41.77%	75.00%	0.32%	50.00%
trigger	0.87%		0.87%	
pic_ctrl_inst	23.65%	30.00%	1.30%	39.66%
LEVEL[0].CO...	25.00%		0.00%	50.00%
LEVEL[0].CO...	25.00%		0.00%	50.00%
LEVEL[0].CO...	25.00%		0.00%	50.00%
LEVEL[0].CO...	25.00%		0.00%	50.00%
LEVEL[0].CO...	25.00%		0.00%	50.00%
LEVEL[1].CO...	25.00%		0.00%	50.00%
LEVEL[1].CO...	25.00%		0.00%	50.00%
LEVEL[1].CO...	25.00%		0.00%	50.00%
LEVEL[2].CO...	25.00%		0.00%	50.00%
LEVEL[2].CO...	25.00%		0.00%	50.00%
LEVEL[3].CO...	25.00%		0.00%	50.00%
SETREG[1].N...	33.33%		16.67%	50.0[illegible]%
SETREG[2].N...	33.33%		16.67%	50.00%
SETREG[3].N...	33.33%		16.67%	50.00%
SETREG[4].N...	33.33%		16.67%	50.00%
SETREG[5].N...	33.33%		16.67%	50.00%
SETREG[6].N...	33.33%		16.67%	50.00%
SETREG[7].N...	33.33%		16.67%	50.00%
SETREG[8].N...	33.33%		16.67%	50.00%

CORE FUNCTIONAL COVERAGE

- To see how much code coverage is achieved by running the riscv_arithmetic_basic_test, run the following cmnd:

  “**make fcov_core TEST=riscv_arithmetic_basic_test SEED=1**”

- Screenshot of html-based **detailed functional_coverage** of core is shown in fig:



riscv_instr_pkg::riscv_instr_cover_group::compressed_opcode_cg	25.00	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::slt_cg	26.17	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::sll_cg	26.17	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::slli_cg	26.25	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::srli_cg	26.25	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_slli_cg	26.61	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_addi4spn_cg	29.17	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::and_cg	29.86	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::sltiu_cg	29.91	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_addi_cg	29.93	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_srai_cg	31.25	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_srli_cg	31.25	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::mulh_cg	36.33	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::auipc_cg	37.50	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::mulhu_cg	39.45	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::mulhsu_cg	39.84	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::mul_cg	40.23	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::or_cg	40.97	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::slti_cg	41.07	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_addi16sp_cg	41.67	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::rem_cg	42.59	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::sub_cg	46.88	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::ori_cg	51.17	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::xori_cg	55.47	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::c_li_cg	56.09	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::divu_cg	57.99	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::div_cg	58.80	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::remu_cg	59.03	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::lui_cg	60.42	1	100	1	0	64	64
riscv_instr_pkg::riscv_instr_cover_group::addi_cg	82.40	1	100	1	0	64	64

DEMO