2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

MUNICH, GERMANY
OCTOBER 15-16, 2024

Tutorial: A Holistic Approach to RISC-V Processor Verification

Larry Lapides

Oğuzhan Turk

SYNOPSYS®

accellera
SYSTEMS INITIATIVE

Agenda

- Why is RISC-V being broadly adopted?
- Challenge: the RISC-V verification disconnect
- A RISC-V processor verification solution
- Dynamic verification
- Formal verification
- Summary

Agenda

- **Why is RISC-V being broadly adopted?**
- Challenge: the RISC-V verification disconnect
- A RISC-V processor verification solution
- Dynamic verification
- Formal verification
- Summary

Why should we use RISC-V?

Anyone can design their own processor based on the RISC-V ISA

Modular ISA = choice of which features to include/exclude

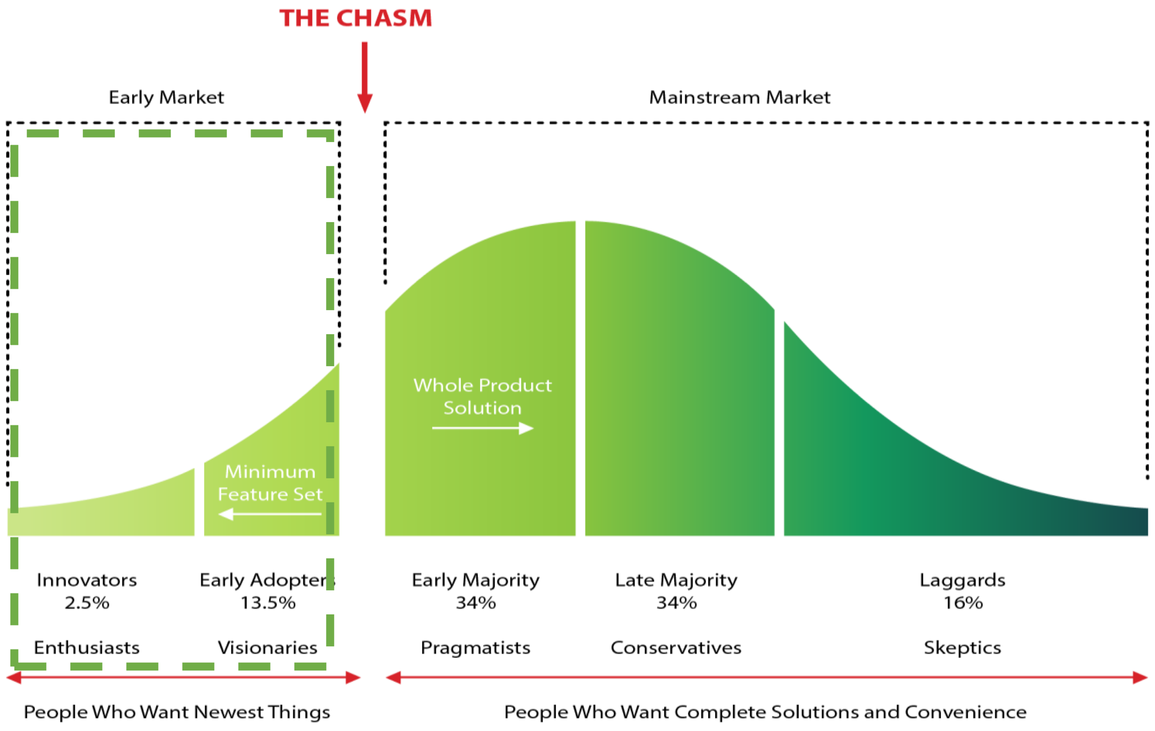
Extensibility and freedom to customize at ISA and micro-architectural levels

RISC-V enables the creation of domain-specific differentiated processors

RISC-V is Crossing the Chasm: 2023-2024

Moving beyond early adopters, into early mainstream

- Initially only used by ‘visionaries’ like SiFive, Andes, Nvidia, Microchip
- Then systems companies wanting domain specific processors
  - Meta Infrastructure, Google, …
  - IoT companies
  - and early adopter semiconductor companies e.g. Qualcomm, Nvidia Networking (Mellanox), Silicon Labs

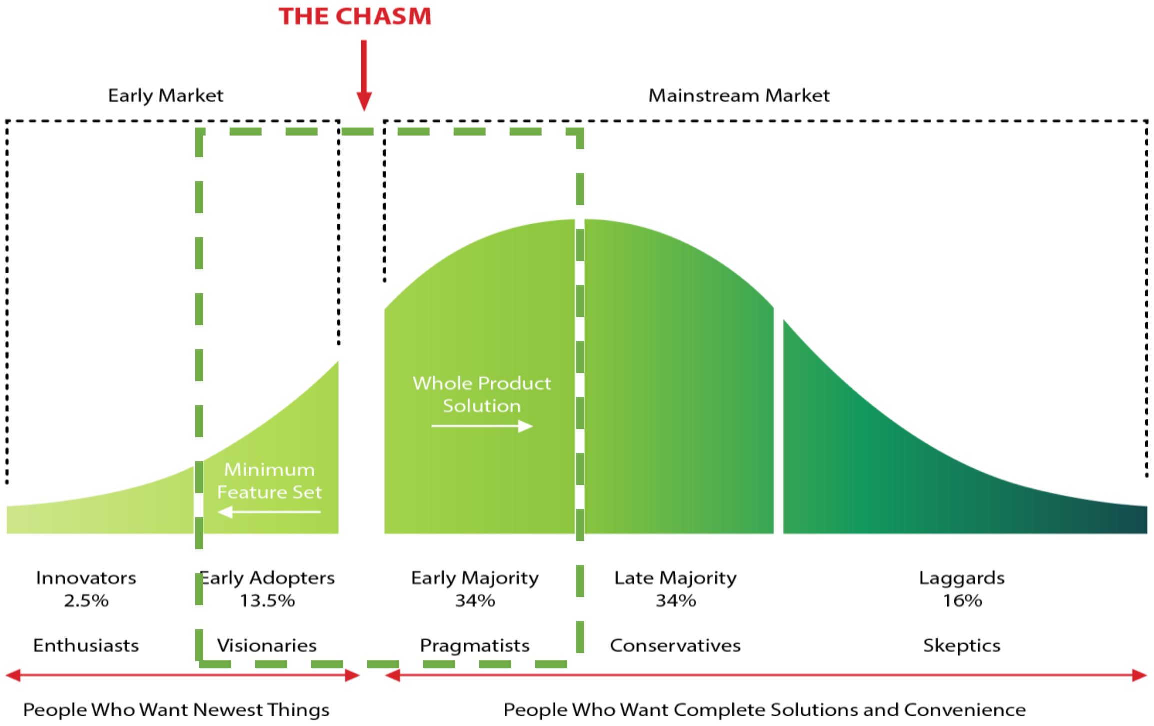


Source: Geoffrey Moore (1991)

RISC-V is Crossing the Chasm: 2023-2024

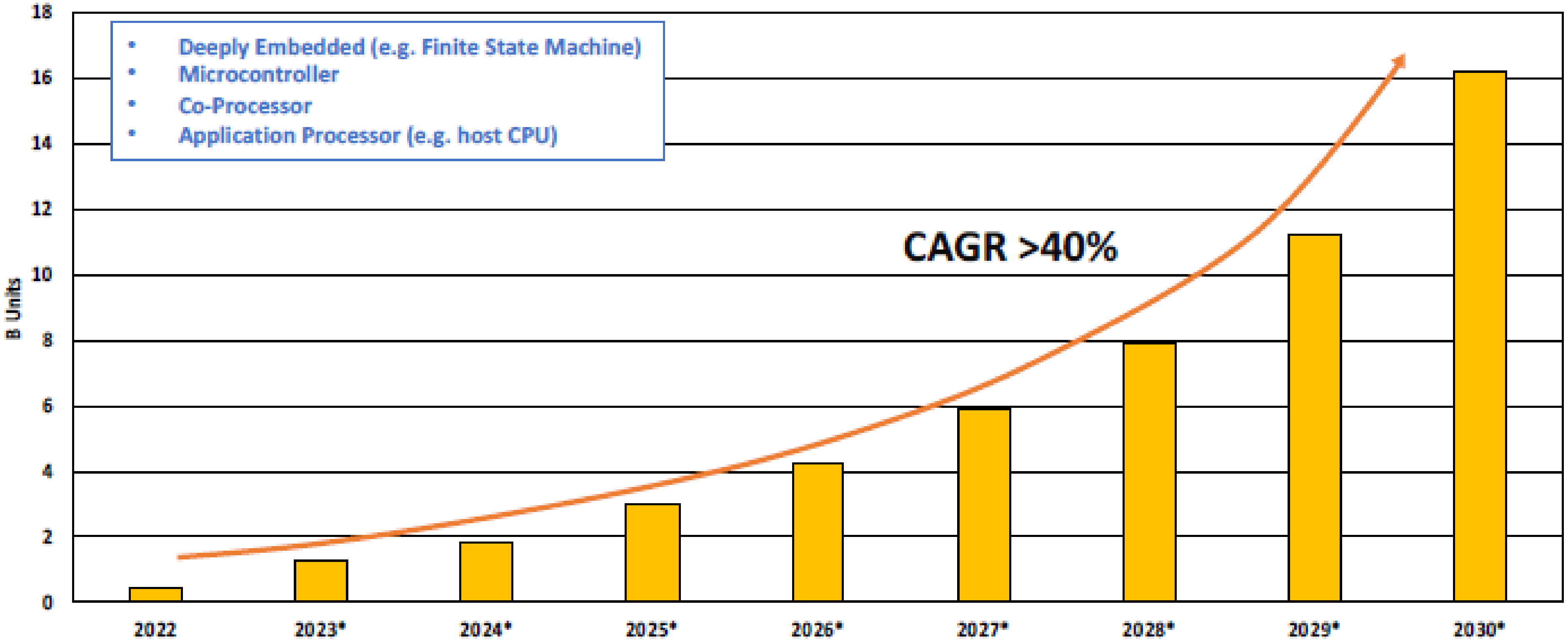
Moving beyond early adopters, into early mainstream

- Now…
  - Every semiconductor vendor has a RISC-V SoC project in flight
  - Every hyperscaler company has a RISC-V project at least at the test chip phase
  - Every automotive OEM and Tier 1 has a RISC-V project at least at the test chip phase



Source: Geoffrey Moore (1991)

Expected RISC-V Market Growth



Source: RISC-V Market Report: Application Forecasts in a Heterogeneous World-Abridged, SHD Group

Agenda

- Why is RISC-V being broadly adopted?
- **Challenge: the RISC-V verification disconnect**
- A RISC-V processor verification solution
- Dynamic verification
- Formal verification
- Summary

The RISC-V Verification Disconnect

RISC-V Core **User**:

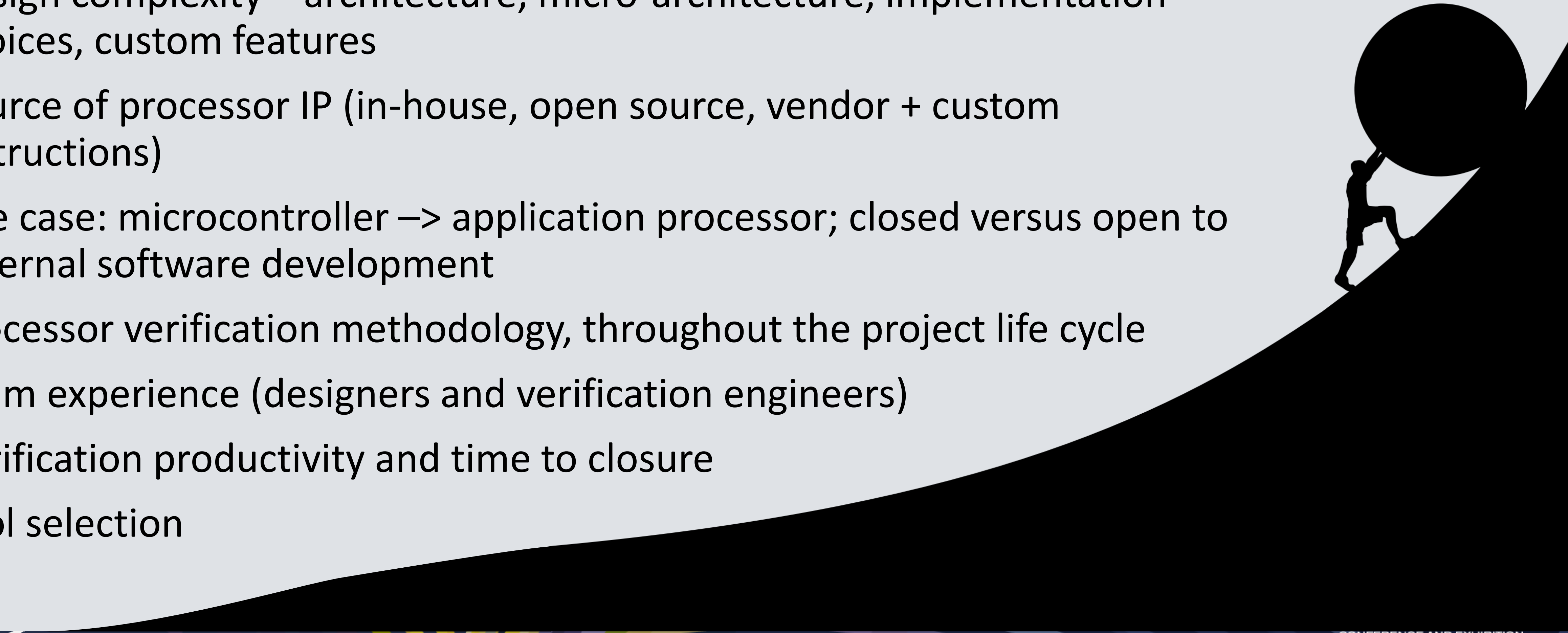
- Expects core quality to be the same as ARM
- 10^{15} verification cycles $=10^{4}$ RTL simulators running 24/7!

RISC-V Core **Developer**:

- Needs to deliver high-quality core
- Potential issues with necessary expertise, methodologies, technologies, resources

Challenges in RISC-V Processor Verification

- Design complexity – architecture, micro-architecture, implementation choices, custom features
- Source of processor IP (in-house, open source, vendor + custom instructions)
- Use case: microcontroller –> application processor; closed versus open to external software development
- Processor verification methodology, throughout the project life cycle
- Team experience (designers and verification engineers)
- Verification productivity and time to closure
- Tool selection



Agenda

- Why is RISC-V being broadly adopted?
- Challenge: the RISC-V verification disconnect
- **A RISC-V processor verification solution**
- Dynamic verification
- Formal verification
- Summary

What have we learned in the last 7 years?

- A verification plan is needed, addressing three orthogonal axes of processor verification:
  - Single core methodology
  - Multi-core processor complexities
  - Project life cycle (pre- to post-silicon)

- Different than with SoC DV, a high-quality, fully functional reference model is needed

- As with SoC DV, the full range of verification technologies is needed
  - Dynamic verification
  - Formal verification
  - Hardware-assisted verification

RISC-V Processor Verification Process

Design verification from unit to SoC

Design Level	Example	Tool/Methodology
Unit	Pipeline, FPU	Formal + predefined assertion IP
	Security	Formal + predefined security assertion IP
Architecture	ISA	Dynamic
		Formal + predefined assertion IP
Custom instructions, CSRs	Custom DSP, matrix	Dynamic
		Formal sequential equivalence checking, register verification, datapath validation
Processing subsystem	Coherent cache, multi- or many-processor accelerator	Dynamic, especially using hardware assisted verification
		Formal property verification for cache coherence verification

Synopsys RISC-V Processor Verification Solutions

Formal Verification

VC Formal FPV + RISC-V ISA AIP
Functional Verification

VC Formal DPV
Verify computational correctness for RISC-V processors

VC Formal SEQ
Verify that custom instructions do not break the original core

VC Formal Portfolio

Dynamic Verification

VSO.ai
Coverage Optimization

STING
Test Generation

ImperasDV
Co-Simulation and Checking
Verification Environment

ImperasFPM
RISC-V Reference Model

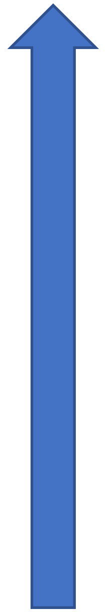
ImperasFC
RISC-V ISA Functional Coverage

VCS & Verdi
Dynamic Simulation

ZeBu & HAPS
HW Assisted Verification

Verdi Verification Planning and Functional Coverage Platform

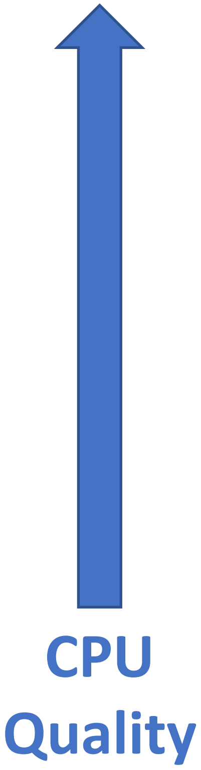
5 Levels of RISC-V Processor DV Methodology



CPU Quality

1) Asynchronous lockstep continuous compare
2) Synchronous step-and-compare
3) Post-simulation trace log file compare
4) Self-checking tests
5) “Hello World”, Linux boot, …

5 Levels of RISC-V Processor DV Methodology



CPU
Quality

1) Asynchronous lockstep continuous compare
2) Synchronous step-and-compare
3) **Post-simulation trace log file compare**
4) Self-checking tests
5) “Hello World”, Linux boot, …

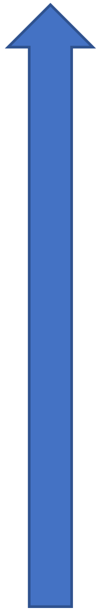
Post-sim Trace Compare (entry level DV): Pros and Cons

- Pros:
  - Simple to set up and use
- Cons:
  - Must run RTL simulation to the end
  - Cannot debug live
  - Incompatible trace formats (between RTL, ISS, …)
  - Easy to skip instructions, and only compare selected few
  - Difficult to verify asynchronous events (e.g. interrupts, debug requests)
  - Not a comprehensive DV strategy
- Key requirement: high quality model of the RISC-V processor
  - ImperasFPM is the high quality commercially supported model
  - Companies/engineers often think they can "easily" build their own model and Instruction Set Simulator (ISS) or use open source as a starting point
    - In our experience, building/maintaining an ISS is not nearly so easy

➢ Post-sim trace compare is widely used

➢ Most effective as a complementary methodology to asynchronous continuous compare

5 Levels of RISC-V Processor DV Methodology

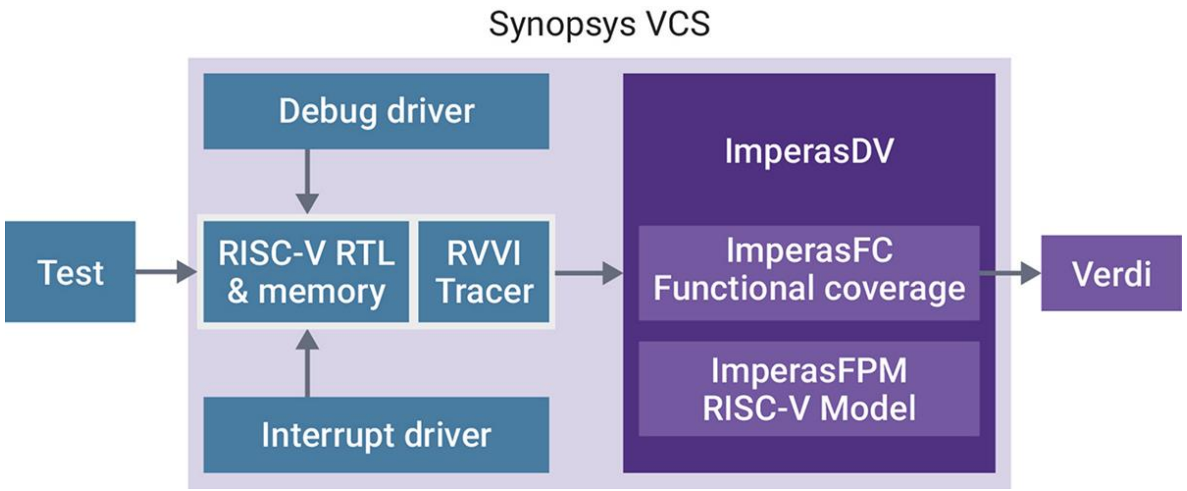


CPU Quality

1) **Asynchronous lockstep continuous compare**
2) Synchronous step-and-compare
3) Post-simulation trace log file compare
4) Self-checking tests
5) “Hello World”, Linux boot, …

Asynchronous Lockstep Continuous Compare Methodology (highest quality processor DV)

- RTL and reference model are run in “lock-step” in the same simulation (co-simulation)
- Asynchronous events are driven into the DUT
- Tracer informs reference model about async events
- ImperasDV handles async events, scoreboarding, comparison, pass/fail
- Test source can be directed test suites for complex features, architecture validation tests, instruction stream generator or other constrained random generator
- Asynchronous events include interrupts, Debug mode, multi-hart processors, multi-issue and Out-of-Order pipeline, …



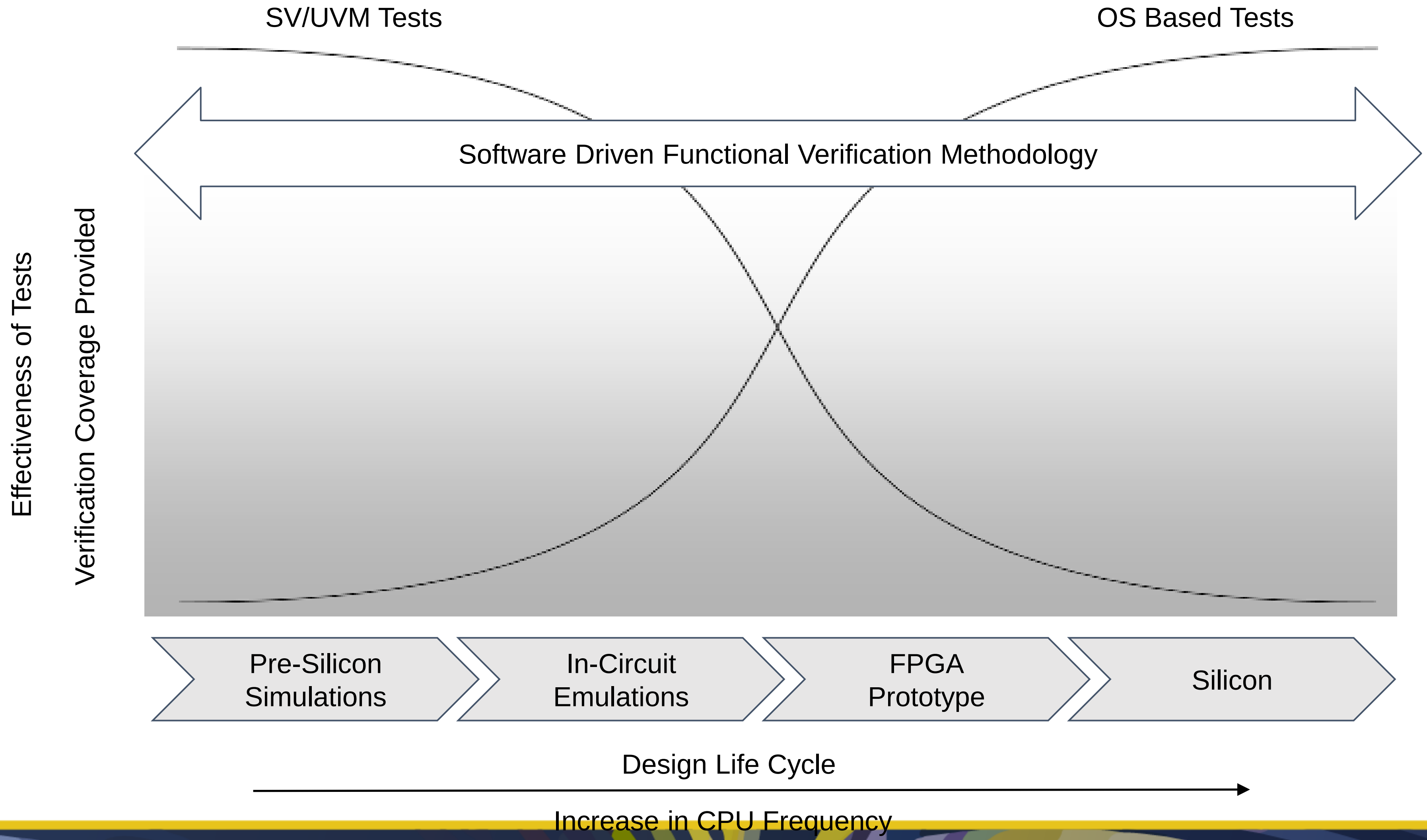
Asynchronous Lockstep Continuous Compare: Pros and Cons

- Pros:
  - Immediate comparison; immediate reporting of bugs
    - Allows for debug introspection at point of failure – very powerful
    - Does not waste execution cycles after failure
  - Most comprehensive DV methodology
    - Enables DV of complex features e.g. interrupts, Debug mode, privilege modes, virtual memory, multi-hart, multi-issue and OoO pipelines
    - Upon instruction retirement, full internal state of the processor is compared to the reference model
- Cons:
  - Users need to develop the RTL RVVI Tracer module, for communication between the DUT and reference model
    - For an engineer familiar with the processor RTL, this is typically 1-2 weeks
- Key requirements: high quality model of the RISC-V processor, co-simulation verification environment
  - ImperasFPM is the high quality commercially supported model
  - Building the verification environment is typically a make versus buy decision
    - ImperasDV provides a commercially supported, easy to use, asynchronous lockstep continuous compare processor verification environment, including functional coverage modules

➢ Asynchronous lockstep continuous compare methodology is used by the leading process IP vendors and companies building their own RISC-V processors

➢ ImperasDV and ImperasFPMs have been used for DV of processors in > 30 SoC tapeouts

Need For Software Driven DV Solution



Allow **consistent execution on all verification environments**

Test stimulus once developed can be reused easily across the design life cycle

Failures hit on silicon can be easily migrated to an earlier stage for faster debug

Early enabling of software based stimulus increases the chances of hitting complex bugs early

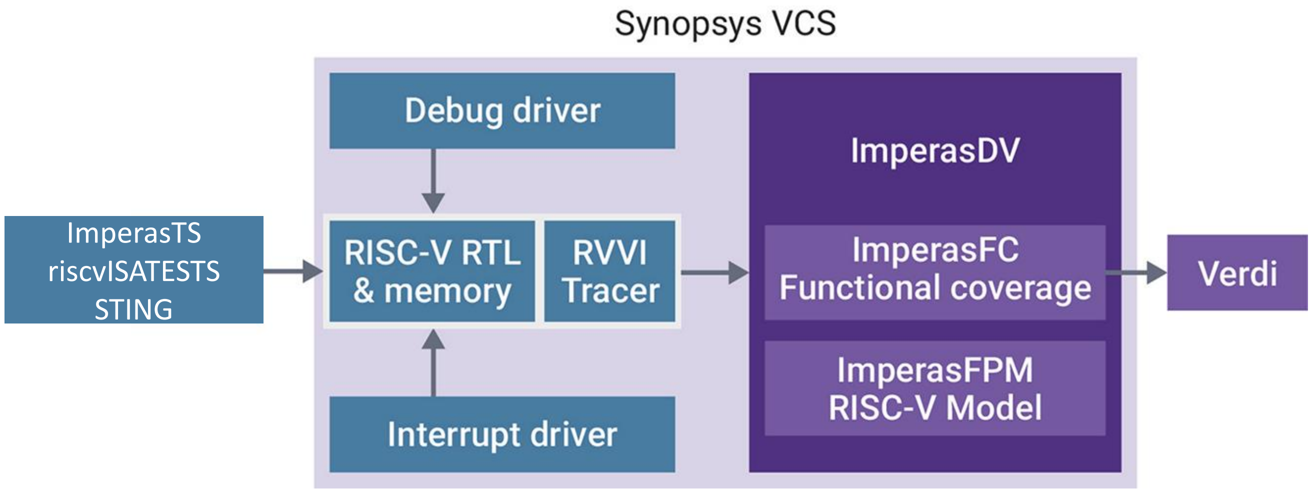
Save on duplicate efforts spent on verification and validation by acting as a bridge between the two methodologies

Agenda

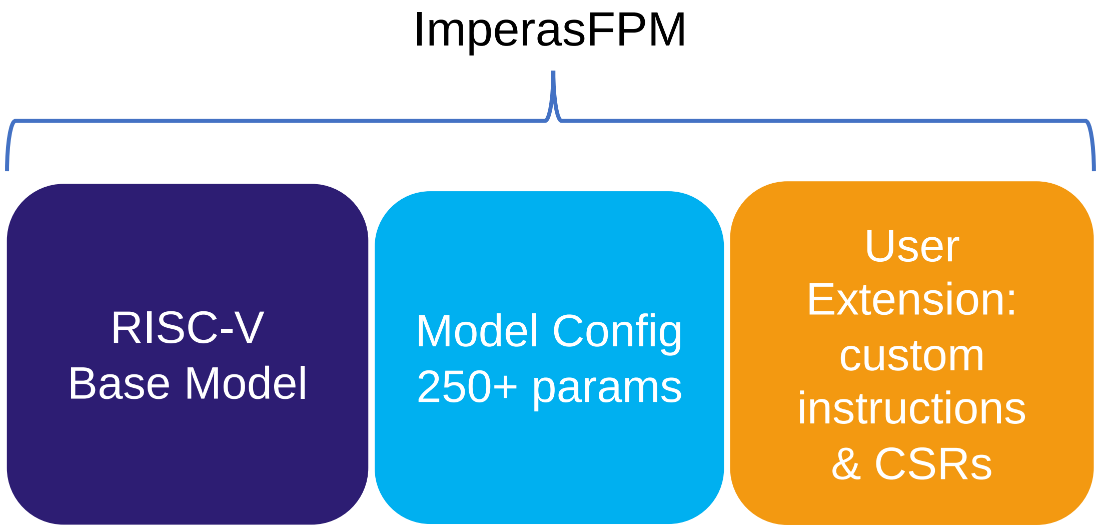
- Why is RISC-V being broadly adopted?
- Challenge: the RISC-V verification disconnect
- A RISC-V processor verification solution
- **Dynamic verification**
- Formal verification
- Summary

Dynamic Verification: ImperasDV + Test Stimuli

- **ImperasFPM RISC-V Processor Model**: for comparison of correct behavior; extendable for custom instructions
- **ImperasDV**: provides configuration, comparison and checking, pipeline synchronization and scoreboarding
- **ImperasFC**: deploys SystemVerilog functional coverage code for each ISA extension
- **riscvISATESTS/ImperasTS**: provides directed test suites
- **STING**: constrained random instruction stream generator

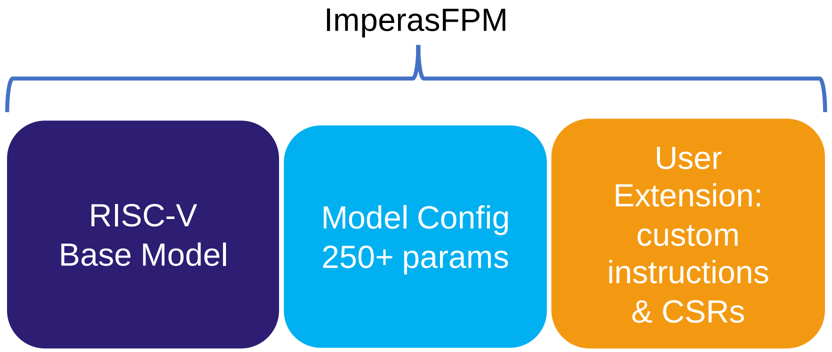


ImperasFPMs (Fast Processor Models) for RISC-V



- Base Model implements RISC-V specification in full
- Fully user configurable to select ISA extensions and versions
- Pre-defined configurations and custom instructions for processor IP vendors
- User extensions built in a separate library do not perturb the verified Base Model, help reduce maintenance
- Because every ImperasFPM uses the RISC-V Base Model, and including users of both commercial and free tools, over 150 companies, organizations and universities have used the ImperasFPM

ImperasFPMs (Fast Processor Models) for RISC-V


ImperasFPM

RISC-V Base Model

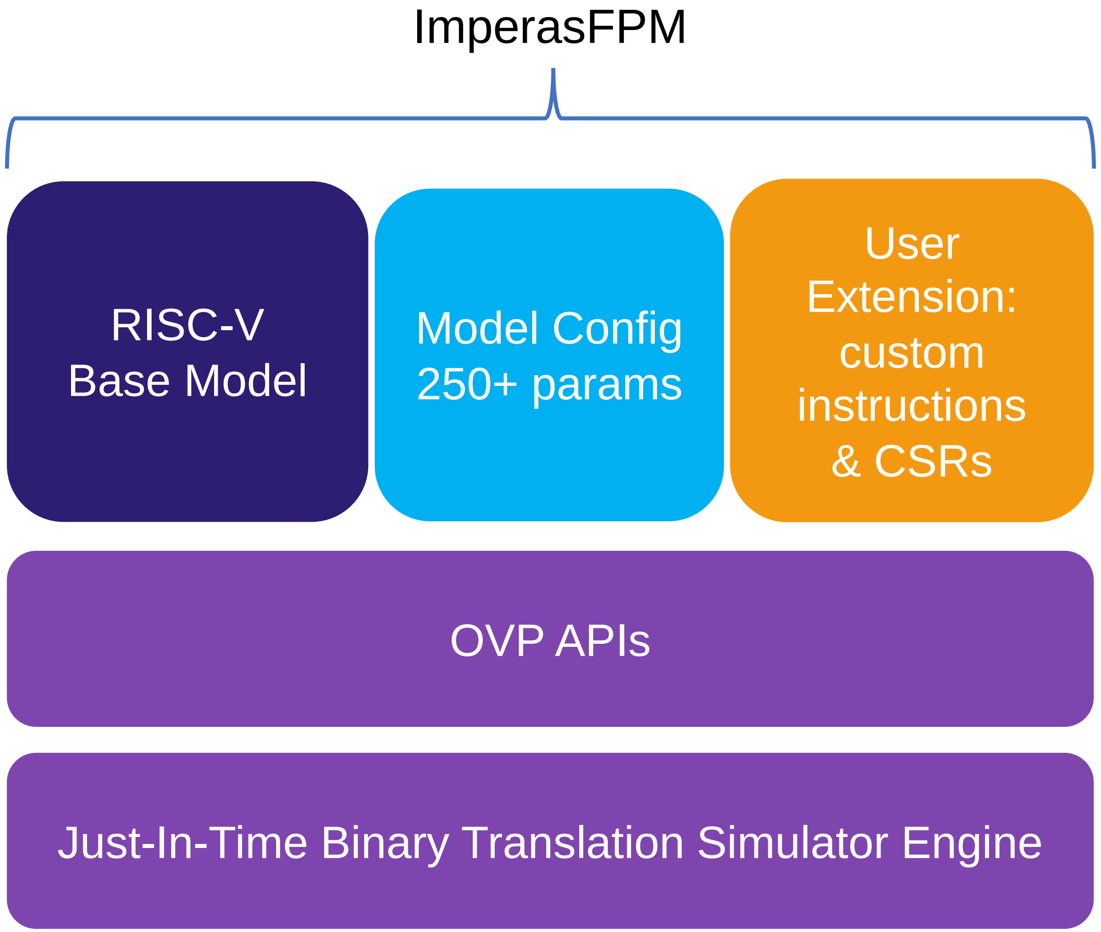
Model Config 250+ params

User Extension: custom instructions & CSRs

- Model architecture and features
  - Base Model implements RISC-V specification in full
  - Fully user configurable to select ISA extensions and versions
  - Pre-defined configurations and custom instructions for processor IP vendors
  - User extensions built in a separate library do not perturb the verified Base Model, help reduce maintenance
- ImperasFPM model testing and validation
  - ImperasFPMs are built using Test Driven Development methodology
  - Synopsys uses Continuous Integration flow – ~20,000 tests run each time engineer checks in code
  - Code coverage metrics and mutation testing tools also used internally
  - Because every ImperasFPM uses the RISC-V Base Model, and including users of both commercial and free tools, over 150 companies, organizations and universities have used the ImperasFPM
  - Models of processor IP vendor cores are validated together with the vendor

RISC-V Base Model is used by 150+ Companies and Organizations

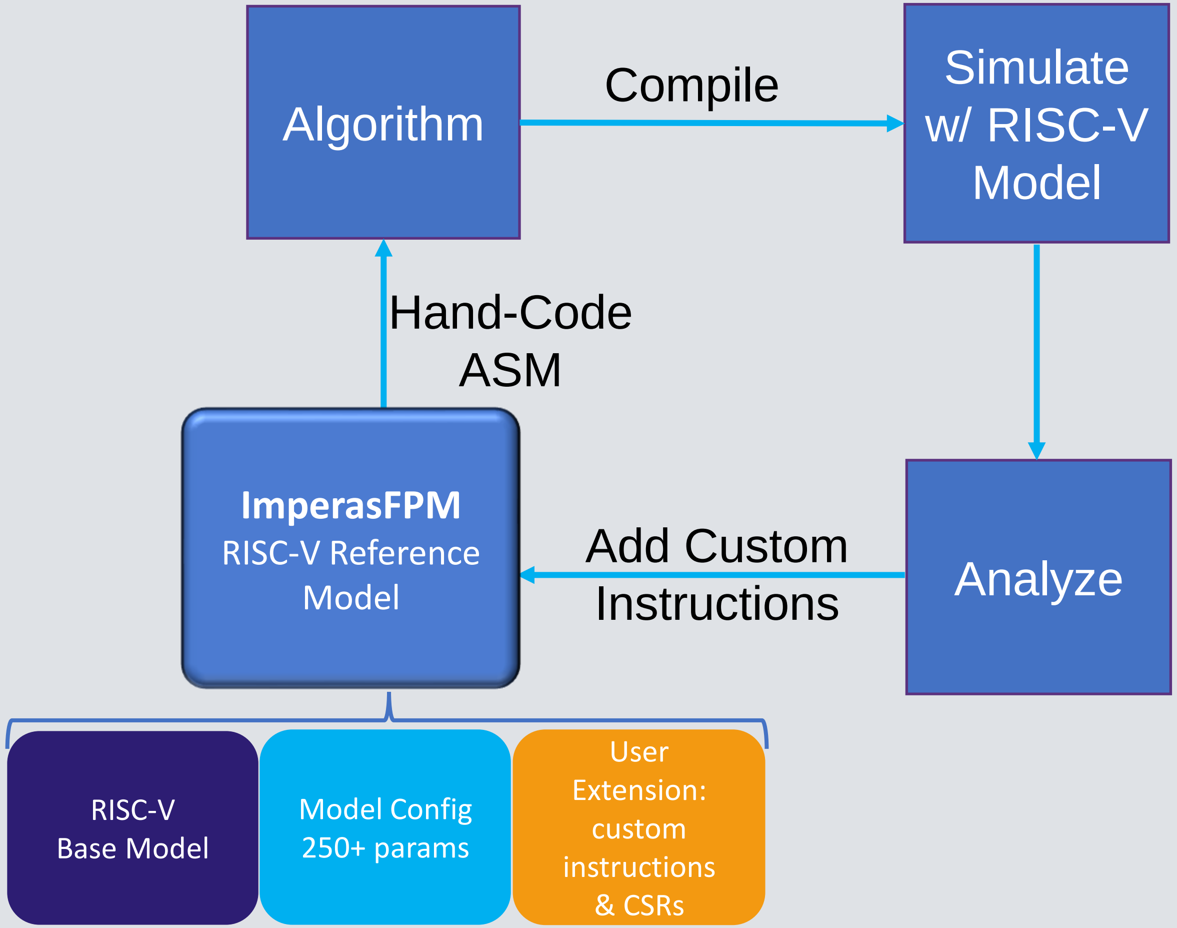
ImperasFPM Architecture


ImperasFPM
RISC-V Base Model
Model Config 250+ params
User Extension: custom instructions & CSRs
OVP APIs
Just-In-Time Binary Translation Simulator Engine

- OVP APIs support …
  - Model functionality
  - Processor analysis tools
- APIs are supported by a Just-In-Time (JIT) binary translation simulator engine
  - Translates RISC-V instructions to x86 on host PC
  - Adds in analysis “instrumentation” to the simulator, so analysis is non-intrusive
- APIs are publicly available: https://github.com/OVPworld/information
- The OVP APIs have been used to develop models of 18 different instruction set architectures (ISAs), including 3 proprietary ISAs
  - Matured by supporting ISAs such as Arm and MIPS before being used for RISC-V

Models Drive Customization

- In the RISC-V world, custom instructions are added to optimize a specific application or set of applications within a domain
  - “Domain-Specific Processors”
- Models let you explore custom instructions quickly
  - Much faster to develop/analyze custom instructions in the model than by writing RTL
  - Better profiling data and other analytical tools
  - Better software debug capabilities
- Methodology
  - Start by characterizing the application to be optimized
  - Then add custom instructions, evaluate and iterate



Algorithm

Compile

Simulate w/ RISC-V Model

Analyze

Add Custom Instructions

ImperasFPM RISC-V Reference Model

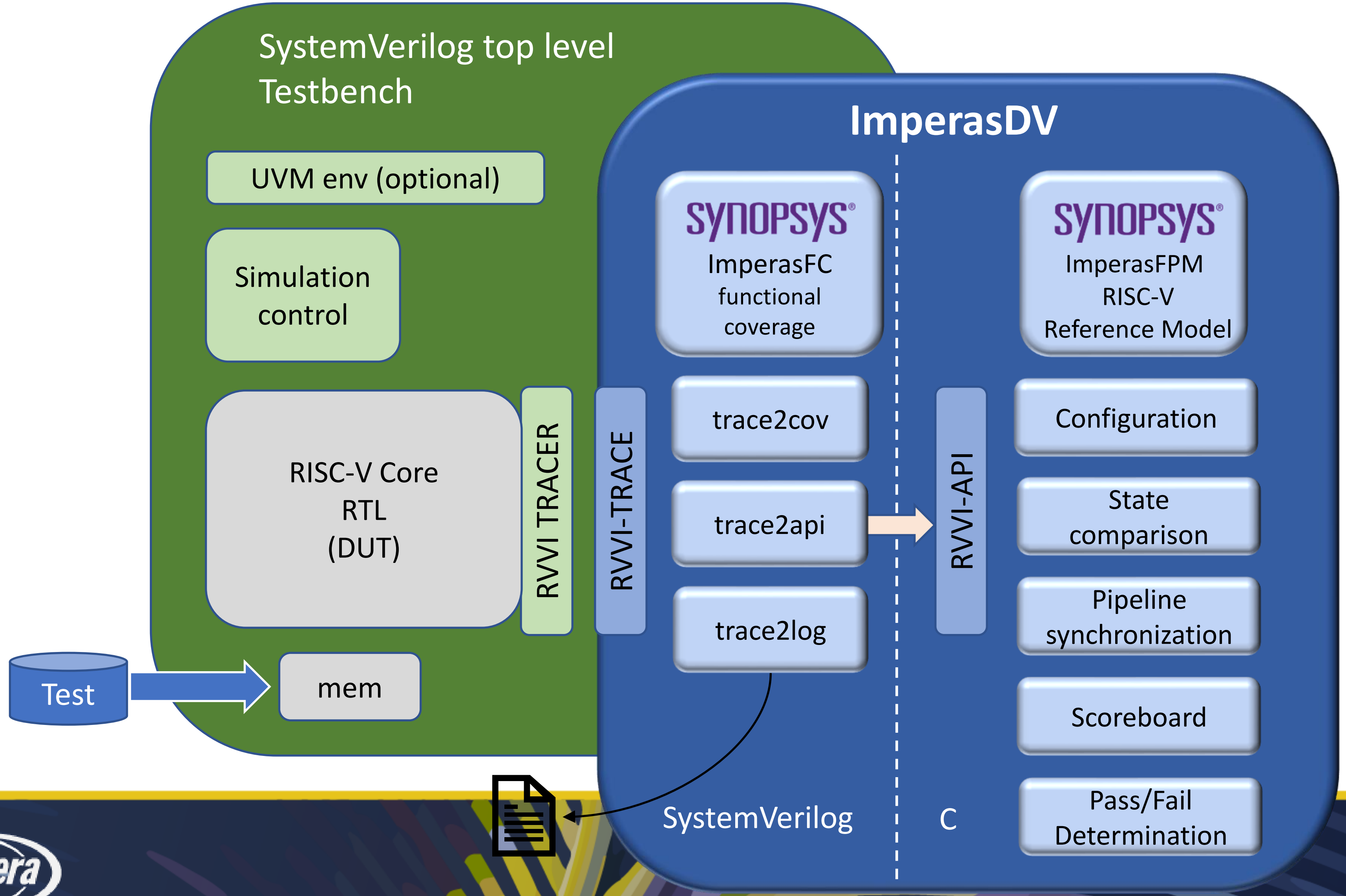
Hand-Code ASM

RISC-V Base Model

Model Config 250+ params

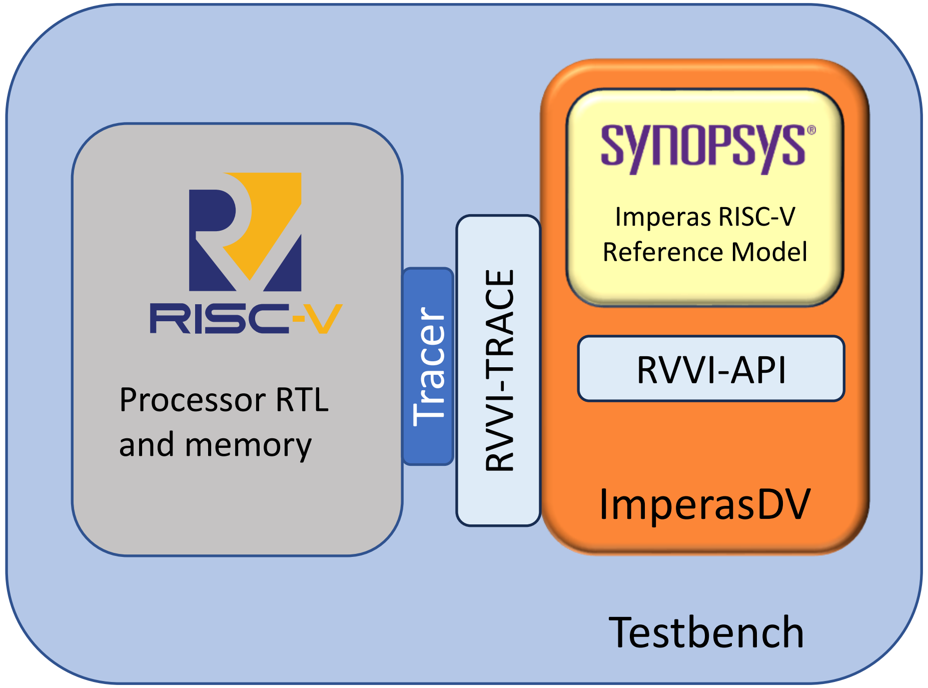
User Extension: custom instructions & CSRs

ImperasDV RISC-V Processor Verification Environment



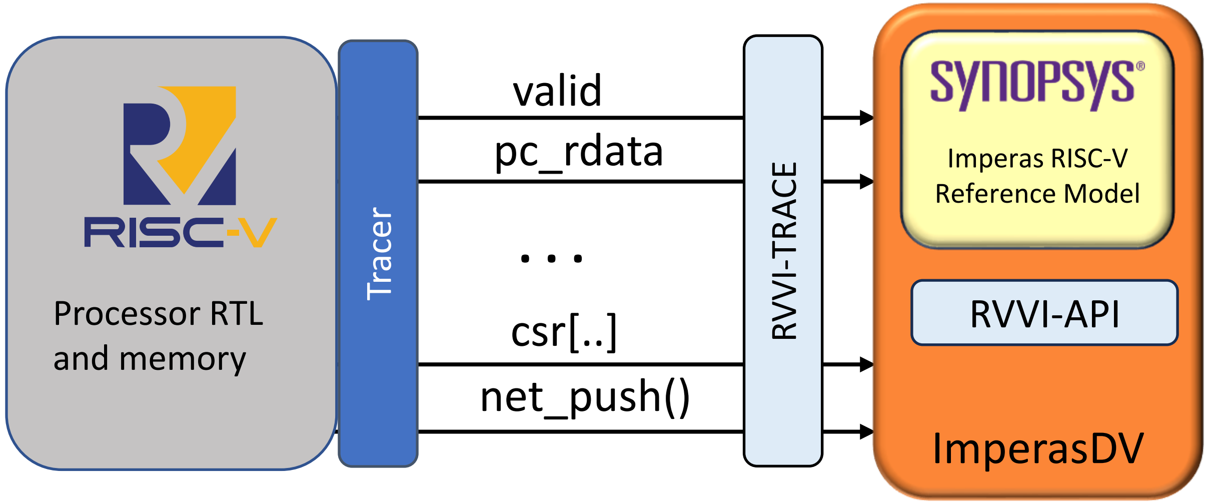
RISC-V Processor Verification using RVVI

- RVVI = RISC-V Verification Interface
  - https://github.com/riscv-verification/RVVI
- Open standard: result of collaboration between industry and open-source
- Standardizes communication between testbench and RISC-V VIP
  - **RVVI-TRACE**: interface between tracer and testbench
  - **RVVI-API**: interface between RISC-V verification component and reference model



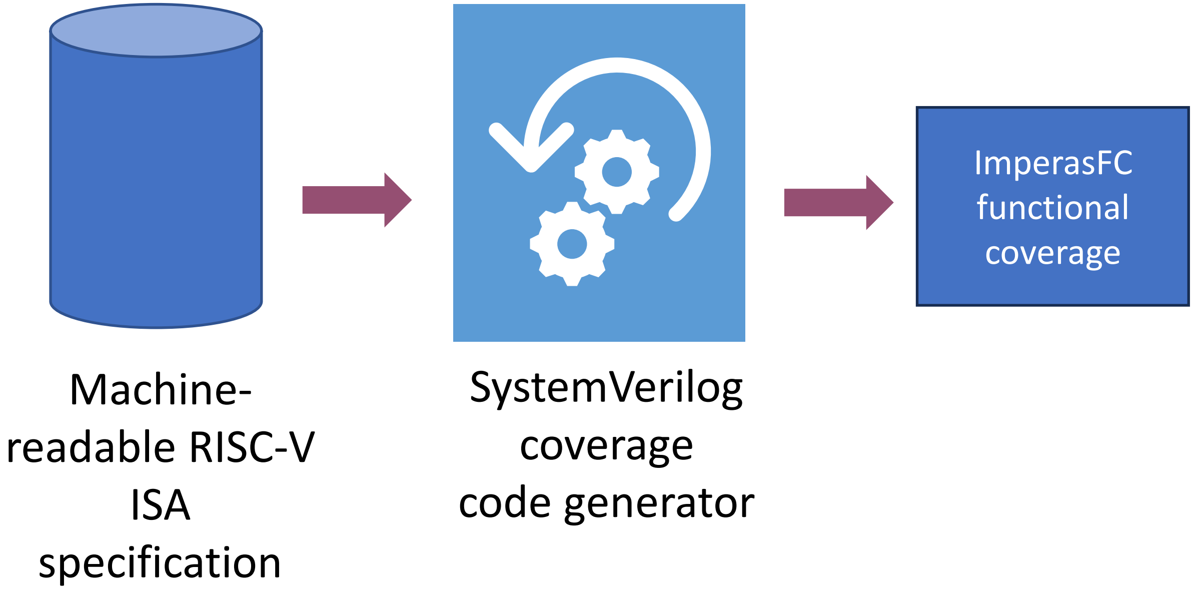
RVVI-TRACE Enables Verification of DUT internal state

- Defines information to be extracted by tracer
  - e.g. PC, CSRs, GPRs, instr. binary
- SystemVerilog interface
- Includes functions to handle asynchronous events
  - e.g. interrupts, debug requests



ImperasFC: SystemVerilog Functional Coverage for RISC-V

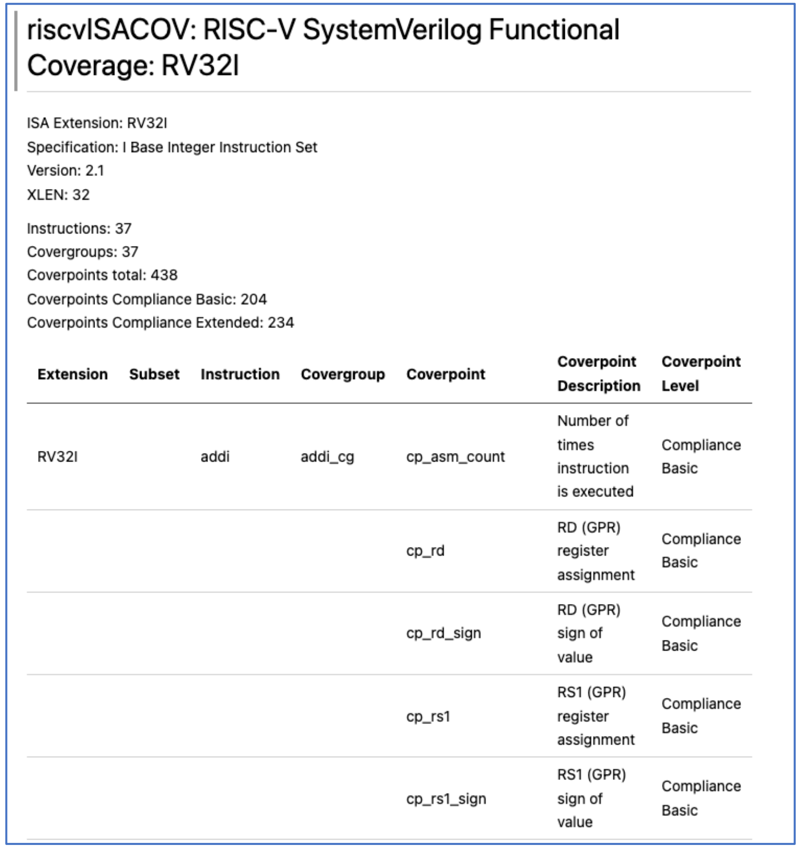
- Functional coverage code generation
  - Manual creation would be tedious, time consuming and error prone
  - >100K lines of code
  - Synopsys tools can automatically generate functional coverage code for custom instructions


Machine-readable RISC-V ISA specification
SystemVerilog coverage code generator
ImperasFC functional coverage

- Functional coverage is the key verification metric

https://github.com/riscv-verification/riscvISACOV/tree/v20240124/documentation for list of covered extensions

Integrating ImperasDV with Verdi

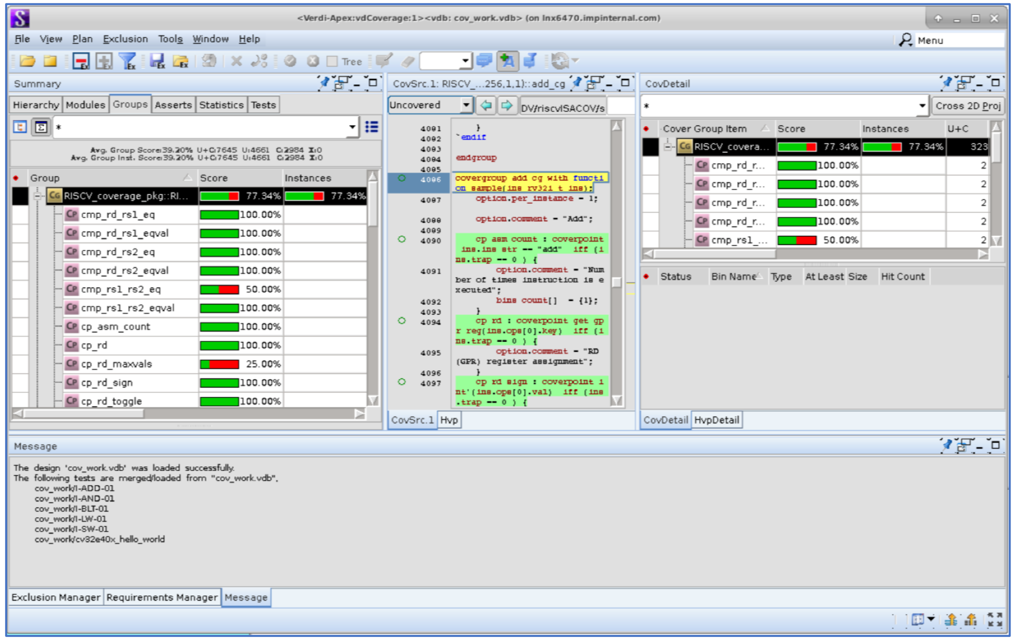


riscvISACOV: RISC-V SystemVerilog Functional Coverage: RV32I

ISA Extension: RV32I
Specification: I Base Integer Instruction Set
Version: 2.1
XLEN: 32

Instructions: 37
Covergroups: 37
Coverpoints total: 438
Coverpoints Compliance Basic: 204
Coverpoints Compliance Extended: 234

Extension	Subset	Instruction	Covergroup	Coverpoint	Coverpoint Description	Coverpoint Level
RV32I		addi	addi_cg	cp_asm_count	Number of times instruction is executed	Compliance Basic
				cp_rd	RD (GPR) register assignment	Compliance Basic
				cp_rd_sign	RD (GPR) sign of value	Compliance Basic
				cp_rs1	RS1 (GPR) register assignment	Compliance Basic
				cp_rs1_sign	RS1 (GPR) sign of value	Compliance Basic



<Verdi-Apex:vdCoverage:1><vdb: cov_work.vdb> (on lnx6470.impinternal.com)
File View Plan Exclusion Tools Window Help
Menu
Tree
Summary
Hierarchy Modules Groups Asserts Statistics Tests
CovSrc.1: RISCV_...256,1,1)::add_cg
Uncovered
DV/riscvISACOV/s
CovDetail
Cross 2D Proj
CovSrc.1 Hvp
CovDetail HvpDetail
Message
Exclusion Manager Requirements Manager Message

Avg. Group Score:39.20% U+C:7645 U:4661 C:2984 X:0
Avg. Group Inst. Score:39.20% U+C:7645 U:4661 C:2984 X:0

Group	Score	Instances
RISCV_coverage_pkg::RI...	77.34%	77.34%
cmp_rd_rs1_eq	100.00%	
cmp_rd_rs1_eqval	100.00%	
cmp_rd_rs2_eq	100.00%	
cmp_rd_rs2_eqval	100.00%	
cmp_rs1_rs2_eq	50.00%	
cmp_rs1_rs2_eqval	100.00%	
cp_asm_count	100.00%	
cp_rd	100.00%	
cp_rd_maxvals	25.00%	
cp_rd_sign	100.00%	
cp_rd_toggle	100.00%	

```
    }
`endif

endgroup

covergroup add_cg with function sample(ins_rv32i_t ins);
    option.per_instance = 1;

    option.comment = "Add";

    cp_asm_count : coverpoint ins.ins_str == "add"  iff (ins.trap == 0 ) {
        option.comment = "Number of times instruction is executed";
        bins count[]  = {1};
    }
    cp_rd : coverpoint get_gpr_reg(ins.ops[0].key)  iff (ins.trap == 0 ) {
        option.comment = "RD (GPR) register assignment";
    }
    cp_rd_sign : coverpoint int'(ins.ops[0].val)  iff (ins.trap == 0 ) {
```

Cover Group Item	Score	Instances	U+C
RISCV_covera...	77.34%	77.34%	323
cmp_rd_r...	100.00%		2
cmp_rd_r...	100.00%		2
cmp_rd_r...	100.00%		2
cmp_rd_r...	100.00%		2
cmp_rs1_...	50.00%		2

Status	Bin Name	Type	At Least	Size	Hit Count

```
The design 'cov_work.vdb' was loaded successfully.
The following tests are merged/loaded from "cov_work.vdb".
    cov_work/I-ADD-01
    cov_work/I-AND-01
    cov_work/I-BLT-01
    cov_work/I-LW-01
    cov_work/I-SW-01
    cov_work/cv32e40x_hello_world
```

- Auto-generated documentation in markdown and csv formats for inclusion in Verification Plans
- Functional coverage data is reported in verification tools such as Verdi

riscvISATESTS & ImperasTS

Test Stimulus

riscvISATESTS

- Directed test suites for architectural validation ("compliance")
- Provided free to licensed users

ImperasTS

- Directed test suites for complex, configurable extensions (Vector, MMU, PMP)
- Test suite generated to match customer’s core configuration

ImperasTS

- MMU
  - Supports Sv32, Sv39, and Sv48 virtual memory systems
  - Separate tests for User and Supervisor modes
  - Tests are generated for a specific physical memory location, will run on a bare metal platform
  - Tests are all self-checking
- PMP, EPMP
  - Supports 32 bit and 64 bit PMP
  - Tests are generated to target specific pmpcfg/pmpaddr regions
  - Allows read-only fields and custom reset values in CSRs
- Vector
  - Configured for specific core setup
  - 7 separate suites

Vector tests

7 test suites

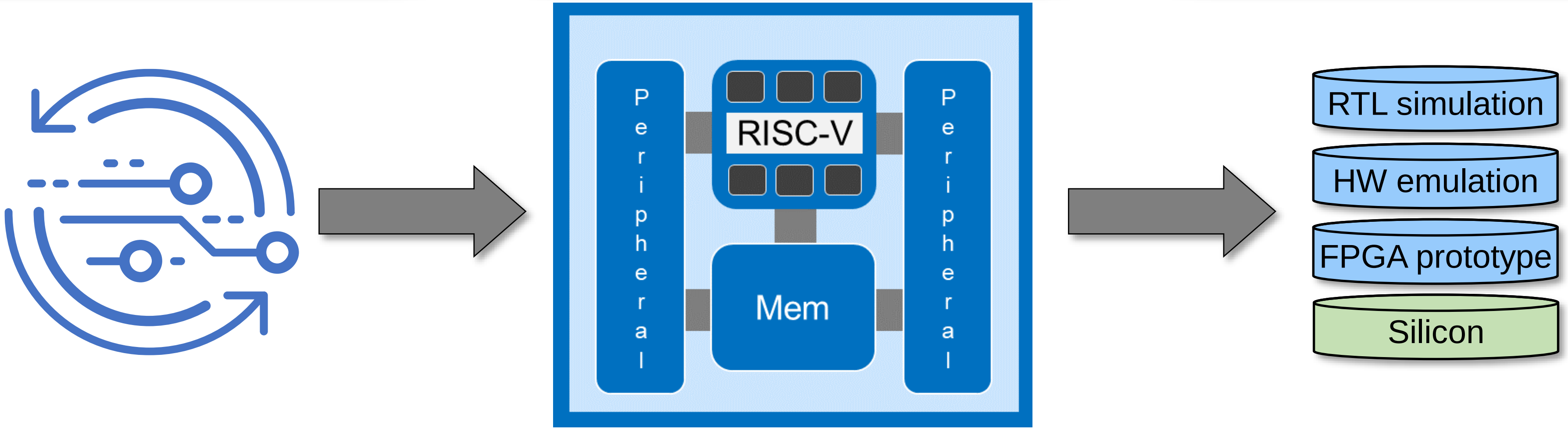
Test Suites	Test Files	Ins. Types	Unique Ins.	Total Ins.	Basic Coverage
Vb	324	2	48	412,064	89.79%
Vf	698	17	91	575,164	86.86%
Vi	1402	12	137	1,112,780	93.54%
Vm	180	2	15	160,680	99.92%
Vp	184	4	21	141,228	91.90%
Vr	146	2	16	110,676	91.67%
Vx	348	6	32	277,504	96.70%

STING Generates Self-Checking Tests for RISC-V Processors and Systems

STING

RISC-V design

Target Platforms



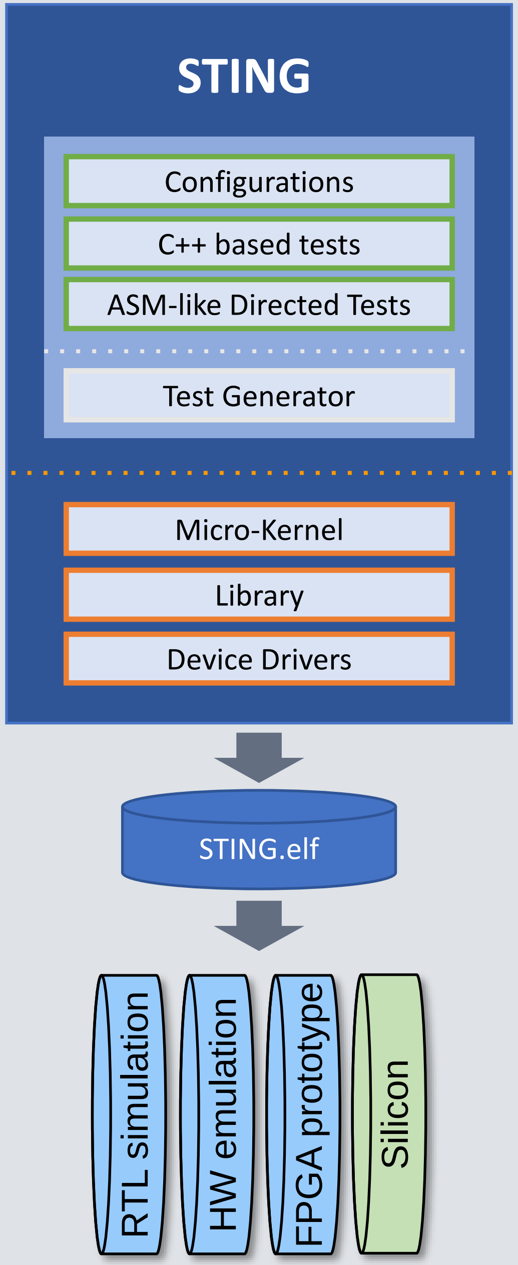
Generate **constrained random, directed and graph-based tests**

Verify **RISC-V ISA, custom extensions, multi-hart, memory coherence, concurrency**

Test on **multiple platforms** including silicon devices

Software-Driven Verification

- Supporting verification throughout the design life cycle, pre- to post-silicon
  - Portable across simulation, emulation, prototyping and silicon
- Supports higher levels of processor complexity and integration, including multi-hart, coherent cache, processing subsystem
  - Full support for the RISC-V ISA specification
  - Extensible to custom instructions and peripheral devices
  - Addressing single CPU and complex many core SoC designs
- Self-checking test generation (use standalone) or instruction stream generation (use with ImperasDV) for RISC-V



Accelerating RISC-V Processor Verification Using Hardware Assisted Verification (HAV) Tools

ImperasDV + HAPS (FPGA prototyping)

✓ Execution speed

- Faster than simulation

✓ Large designs

- Impractical to simulate

✓ Highest quality verification

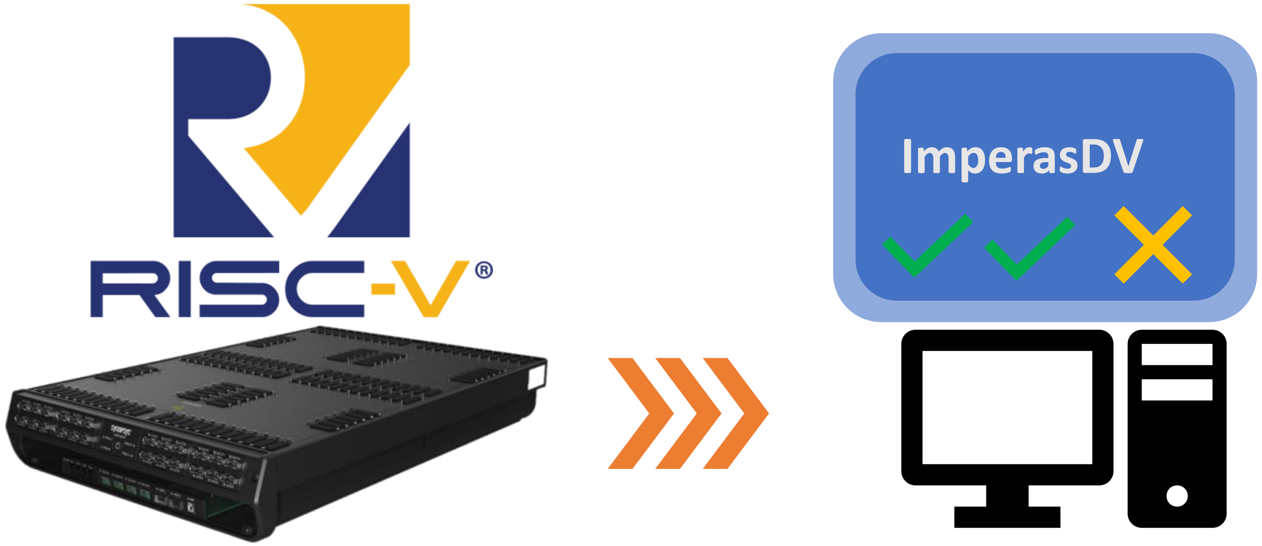
- ImperasDV compare technology

✓ Verification metrics

- ImperasFC functional coverage

RISC-V design running on HAPS

RVVI-TRACE data sent to ImperasDV



Agenda

- Why is RISC-V being broadly adopted?
- Challenge: the RISC-V verification disconnect
- A RISC-V processor verification solution
- Dynamic verification
- **Formal verification**
- Summary

Synopsys VC Formal – Leading Formal Innovations

Unified Compile with VCS

Unified Formal Debugger with Verdi

FPV Property Verification	FTA Testbench Analyzer	SEQ Sequential Equivalence	DPV Datapath Validation
CC Connectivity Checking	FCA Coverage Analyzer	FXP X-Propagation Verification	FRV Register Verification
AEP Auto Checks	FuSa Functional Safety	FSV Security Verification	FLP Low Power

Rich Set of Assertion IPs (Including RISC-V AIP)

ML-Enabled Formal Engines and Orchestrations

Industry's Fastest Growing Formal Solution!

Deliver highest performance

Innovative formal engines and ML-based orchestrations find more bugs and achieve more proofs on larger designs

Enable formal signoff

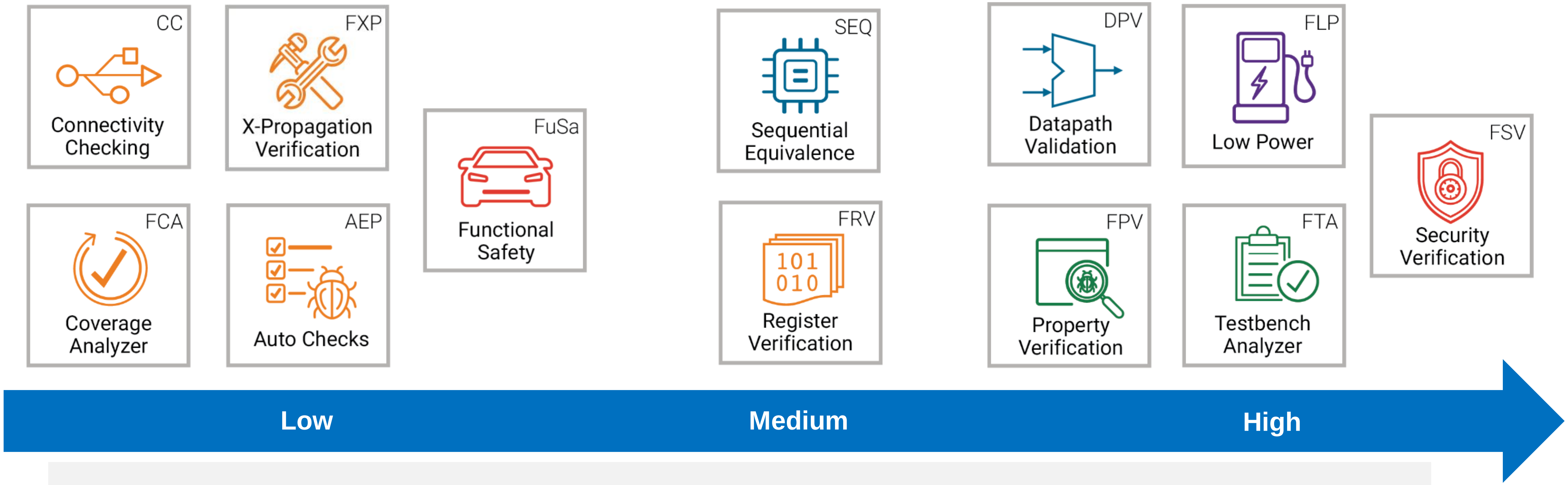
Exhaustive formal analysis catches corner-case bugs and enables formal signoff for control and datapath blocks

Ease Formal adoption

Easy-to-use formal apps, native integration with VCS and Verdi, and Formal Consulting Services reduce formal adoption effort

Synopsys VC Formal: Innovative Formal Verification Solutions

VC Formal Apps Adoption Effort – Formal Expertise Not Always Required



CC Connectivity Checking

FXP X-Propagation Verification

FCA Coverage Analyzer

AEP Auto Checks

FuSa Functional Safety

SEQ Sequential Equivalence

FRV Register Verification

DPV Datapath Validation

FPV Property Verification

FLP Low Power

FTA Testbench Analyzer

FSV Security Verification

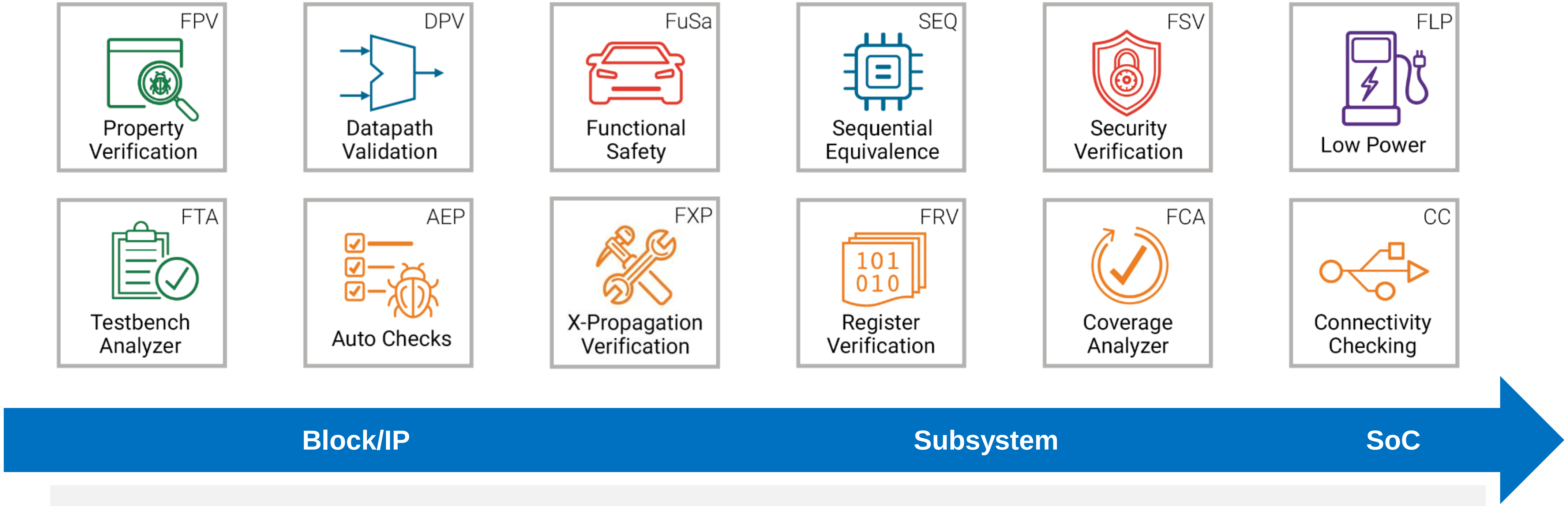
Low → Medium → High

Verification Complexity: In terms of exhaustive computation analysis required to verify the DUT

Adoption Effort: In terms of formal expertise and testbench required to apply the specific APP

Synopsys VC Formal: Innovative Formal Verification Solutions

VC Formal Apps Can Be Used Throughout the SoC Flow

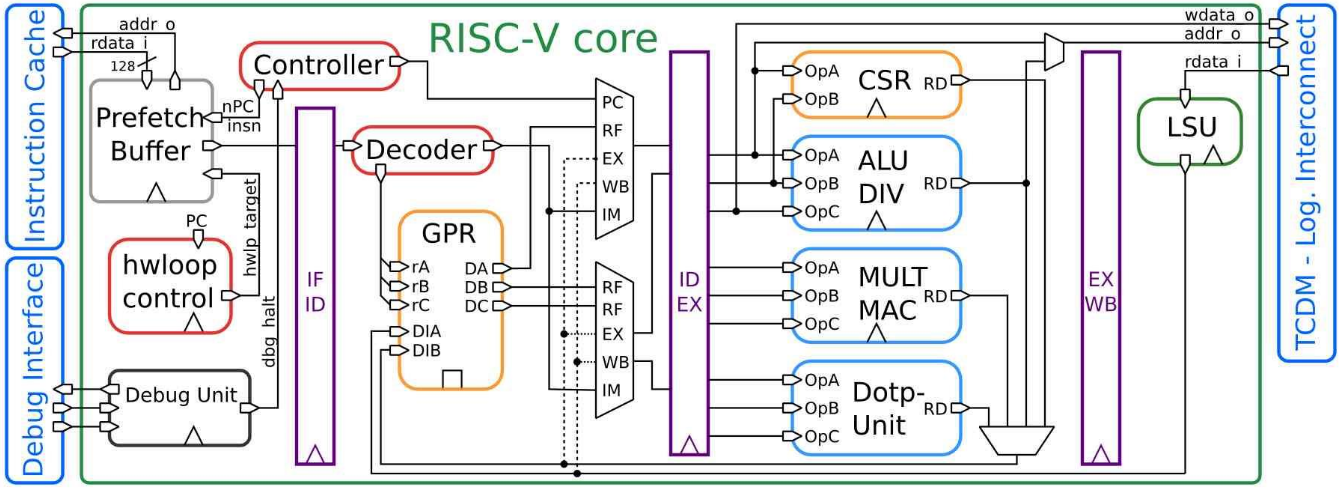


<u>High Performance</u>: ML powered proprietary engines for hard proofs, liveness, and deep bug-hunting

<u>High Confidence Formal Signoff</u>: Native Certitude integration for fast and high-quality Formal Signoff

RISC-V Core Formal Verification Overview

- FPV (Model Checking):
  - Prefetch Buffer
  - LSU – Load/Store unit
  - Pipeline
  - RISC-V AIP
- DPV (Equivalence Checking):
  - ALU/MULT/Dotp
  - Decoder
- SEQ (Equivalence Checking):
  - Clock gating verification in every functional unit
  - Designs comparison in presence of new features/timing changes
- FRV (Formal Register Verification)
  - Control and Status Registers (Zicsr)
- FSV (Formal Security Verification)
  - Secure/Non-secure data propagation



Source: https://www.semanticscholar.org/paper/Near-Threshold-RISC-V-Core-With-DSP-Extensions-for-GautschiSchiavone/47f8ce7e0f0f64d0707a13c83c32c30959aa64d5/figure/6

- RV32I base ISA, for example:
  - LOAD - LSU
  - STORE - LSU
  - BRANCH/JUMP/LUI/AUIPC - PFU
  - OP-IMM - EXU
  - OP - EXU
  - Environment call/break point
- Zicsr extension
  - CSR Write
  - CSR Read

FPV
Property Verification

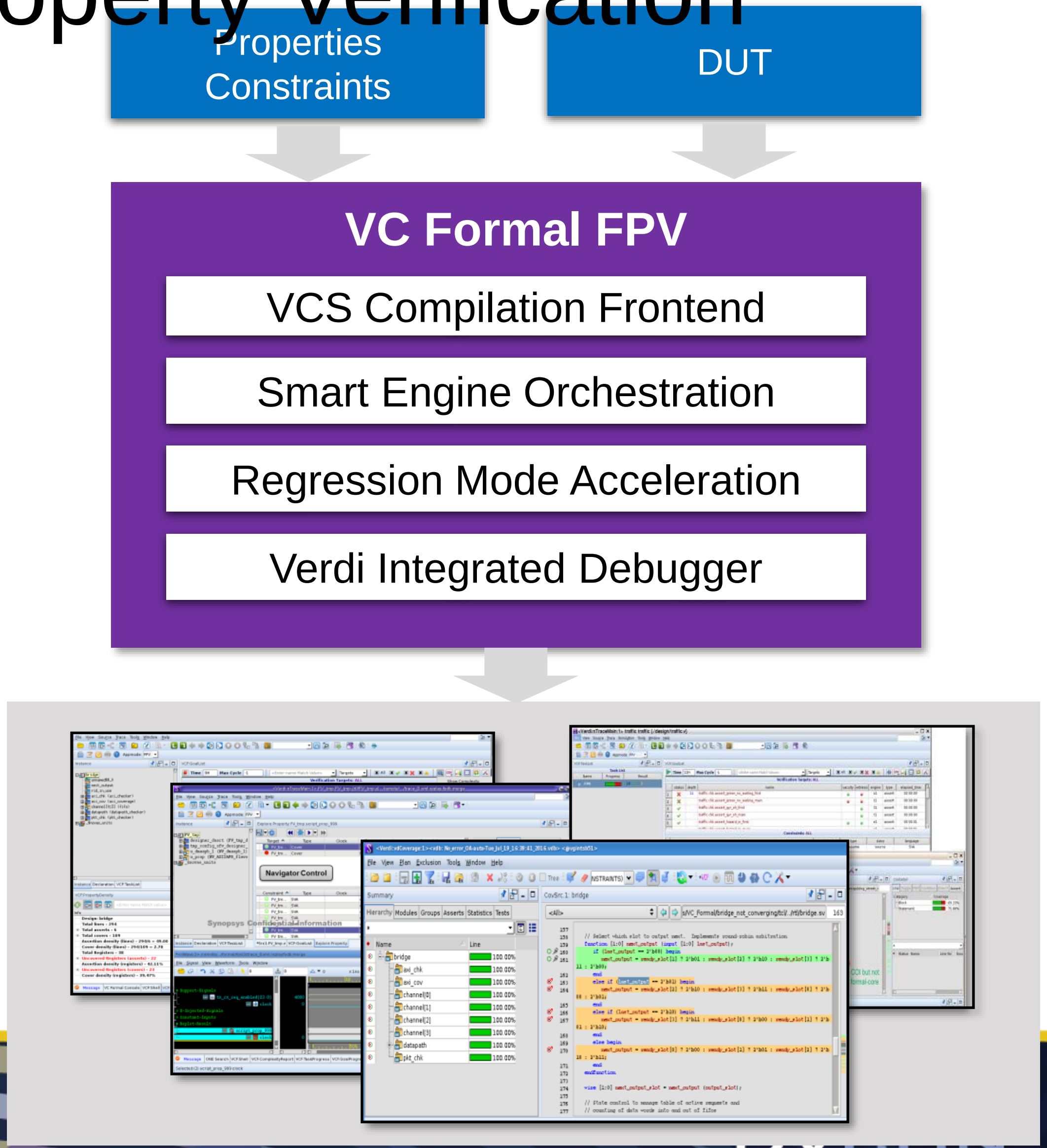
VC Formal FPV: Formal Property Verification

FPV BENEFITS

- Verify functional correctness of design blocks through exhaustive formal analysis
- Find corner-case bugs early without simulation and reduce time to verification closure
- Enable formal signoff methodology

FPV FEATURES

- State-of-the-art ML-powered formal analysis engines and orchestration offer best performance and capacity
- Integrated Verdi GUI offers the most familiar debugging
- Deep bug hunting and advanced proof techniques Proof Assist, Proof Architect



Properties
Constraints

DUT

VC Formal FPV

VCS Compilation Frontend

Smart Engine Orchestration

Regression Mode Acceleration

Verdi Integrated Debugger

DPV

Datapath Validation

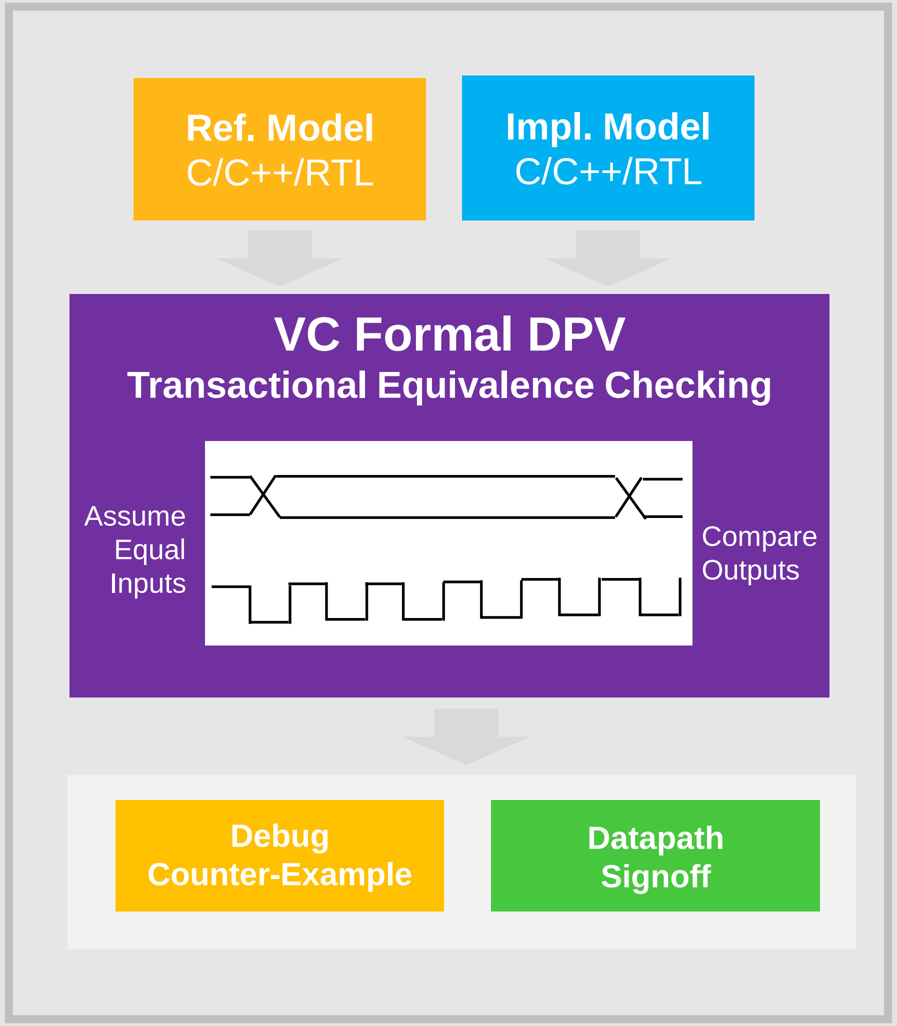
VC Formal DPV: Datapath Validation

DPV BENEFITS

- Exhaustively verify datapath design refinements
- Prove consistency of independently developed reference & implementation models
- Achieve datapath signoff without any testbench

DPV FEATURES

- Integrated mature HECTOR technology
- Supports ADD, SUB, MULT, DIV, SQRT operators
- Applicable to CPU, GPU, DSP, AI/ML (CNN) and other data processing designs



Ref. Model
C/C++/RTL

Impl. Model
C/C++/RTL

VC Formal DPV
Transactional Equivalence Checking

Assume Equal Inputs

Compare Outputs

Debug Counter-Example

Datapath Signoff

SEQ
Sequential Equivalence

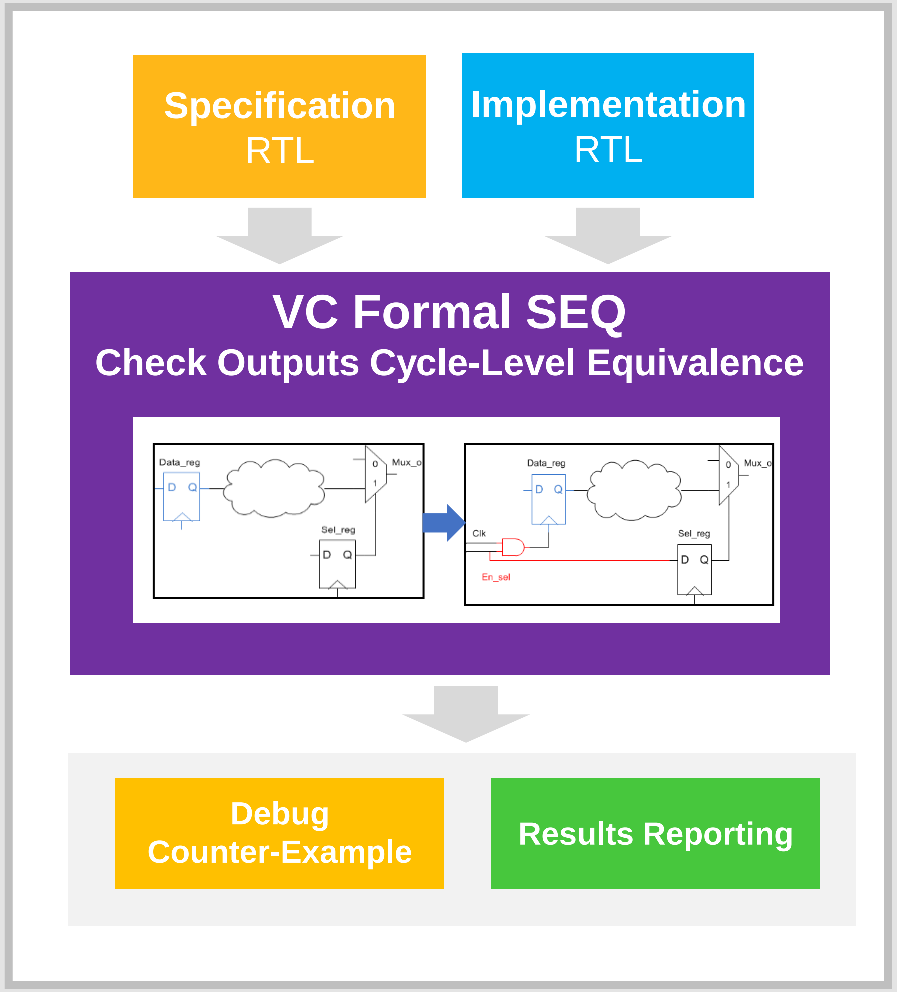
VC Formal SEQ: Sequential Equivalence Checking

SEQ BENEFITS

- Exhaustively verify and signoff the design optimizations without any testbench
- Push the frontier of performance, power, and area (PPA) optimizations
- Save weeks/months simulation regression time

SEQ FEATURES

- Supports clock gating, retiming, microarchitecture optimizations
- Automatically creates equivalence mapping between specification and implementation RTL
- State-of-the-art ML powered formal engine for best performance



Specification RTL

Implementation RTL

VC Formal SEQ
Check Outputs Cycle-Level Equivalence

Debug Counter-Example

Results Reporting

FRV
101
010
Register Verification

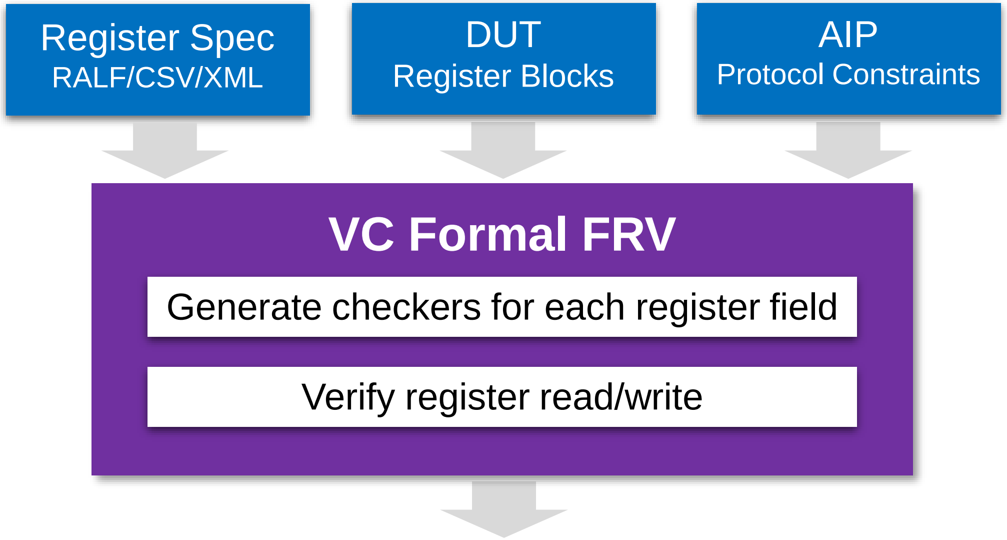
VC Formal FRV: Formal Register Verification

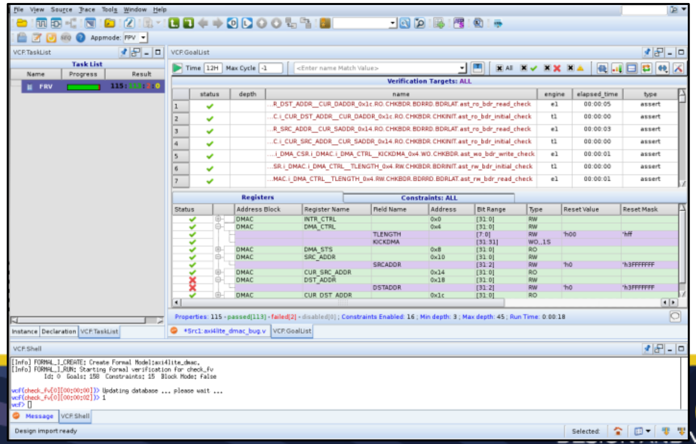
FRV BENEFITS

- Exhaustively verify the consistency of register model against specification
- Find corner-case bugs earlier in the design cycle, shorten debug time
- Save time and effort compared with manual directed simulation tests

FRV FEATURES

- Accept IP-XACT, CSV, RALF spec formats
- Verify that Control Status Registers are correctly implemented using standard or proprietary bus protocols
- Applicable at both the block and SoC level





Verification Targets: ALL

	status	depth	name	engine	elapsed_time	type
1	✓		...R_DST_ADDR__CUR_DADDR_0x1c.RO.CHKBDR.BDRRD.BDRLAT.ast_ro_bdr_read_check	e1	00:00:05	assert
2	✓		...C.i_CUR_DST_ADDR__CUR_DADDR_0x1c.RO.CHKBDR.CHKINIT.ast_ro_bdr_initial_check	t1	00:00:00	assert
3	✓		...R_SRC_ADDR__CUR_SADDR_0x14.RO.CHKBDR.BDRRD.BDRLAT.ast_ro_bdr_read_check	e1	00:00:03	assert
4	✓		...C.i_CUR_SRC_ADDR__CUR_SADDR_0x14.RO.CHKBDR.CHKINIT.ast_ro_bdr_initial_check	t1	00:00:00	assert
5	✓		....i_DMA_CSR.i_DMAC.i_DMA_CTRL__KICKDMA_0x4.WO.CHKBDR.ast_wo_bdr_write_check	e1	00:00:01	assert
6	✓		...SR.i_DMAC.i_DMA_CTRL__TLENGTH_0x4.RW.CHKBDR.BDRINIT.ast_rw_bdr_initial_check	t1	00:00:00	assert
7	✓		...MAC.i_DMA_CTRL__TLENGTH_0x4.RW.CHKBDR.BDRRD.BDRLAT.ast_rw_bdr_read_check	e1	00:00:01	assert

Registers | Constraints: ALL

Status	Address Block	Register Name	Field Name	Address	Bit Range	Type	Reset Value	Reset Mask
✓	DMAC	INTR_CTRL		0x0	[31:0]	RW		
✓	DMAC	DMA_CTRL		0x4	[31:0]	RW		
✓			TLENGTH		[7:0]	RW	'h00	'hff
✓			KICKDMA		[31:31]	WO,,1S		
✓	DMAC	DMA_STS		0x8	[31:0]	RO		
✓	DMAC	SRC_ADDR		0x10	[31:0]	RW		
✓			SRCADDR		[31:2]	RW	'h0	'h3FFFFFFF
✓	DMAC	CUR_SRC_ADDR		0x14	[31:0]	RO		
✗	DMAC	DST_ADDR		0x18	[31:0]	RW		
✗			DSTADDR		[31:2]	RW	'h0	'h3FFFFFFF
✓	DMAC	CUR_DST_ADDR		0x1c	[31:0]	RO		

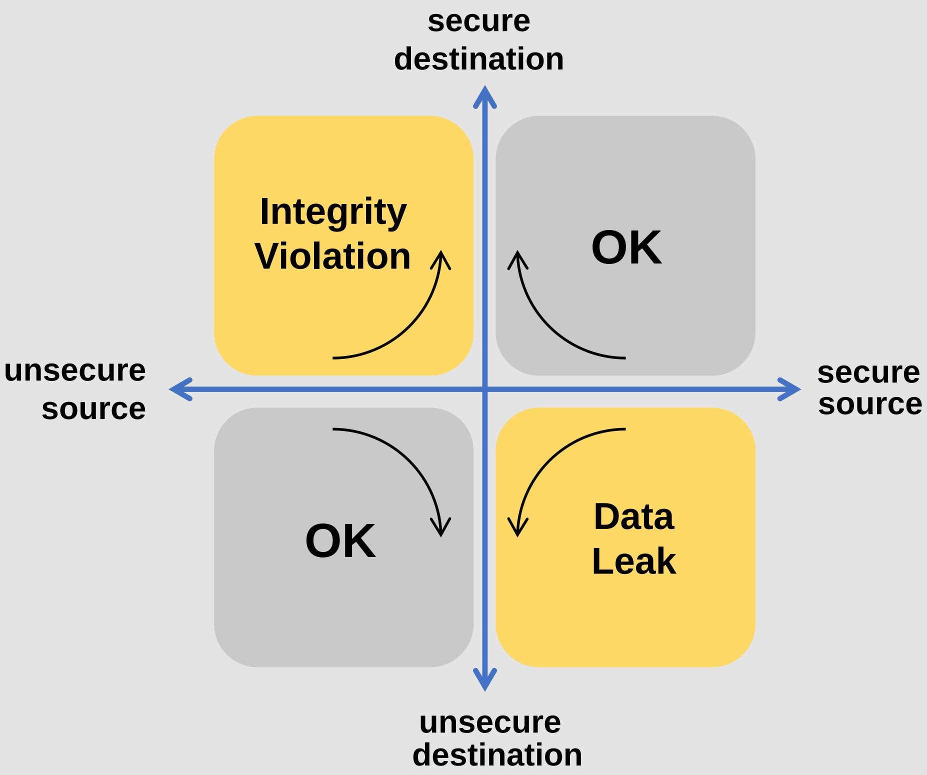
Properties: 115 - passed[113] - failed[2] - disabled[0] ; Constraints Enabled: 16 ; Min depth: 3 ; Max depth: 45 ; Run Time: 0:00:18

```
[Info] FORMAL_I_CREATE: Create Formal Model:axi4lite_dmac.
[Info] FORMAL_I_RUN: Starting formal verification for check_fv
         Id: 0  Goals: 158  Constraints: 15  Block Mode: false

vcf(check_fv[0][00:00:00])> Updating database ... please wait ...
vcf(check_fv[0][00:00:02])> 1
vcf> []
```

Design import ready

VC Formal FSV: Formal Security Verification



FSV FEATURES

- Flexible property creation & management
- ML powered engines for fast performance
- Data propagation analysis and debug with temporal flow view
- Verification of multiple scenarios in one session

FSV BENEFITS

- Ensure data security objectives are met through exhaustive formal analysis
- Ensure secure data cannot be read illegally or be written from an unsecure source
- Detect security issues that are hard to find through other techniques

Agenda

- Why is RISC-V being broadly adopted?
- Challenge: the RISC-V verification disconnect
- A RISC-V processor verification solution
- Dynamic verification
- Formal verification
- **Summary**

How to close the RISC-V Verification Disconnect?

- Need verification plan including metrics
- Need to consider multiple, complementary methodologies and technologies
  - Dynamic and formal simulation
  - Processor only or higher level of integration
  - RTL through SoC
- High quality RISC-V reference model
  - Support the full RISC-V specification
  - Support custom instructions
- Use silicon-proven processor verification tools and models

How to close the RISC-V Verification Disconnect?

- Need verification plan including metrics
- Need to consider multiple, complementary methodologies and technologies
  - Dynamic and formal simulation
  - Processor only or higher level of integration
  - RTL through SoC
- High quality RISC-V reference model
  - Support the full RISC-V specification
  - Support custom instructions
- Use silicon-proven processor verification tools and models

Thank you!

Questions?