A Modern Fable: The Lost Art of Processor Verification

Verification Futures – Austin

Larry Lapides

14 September 2023

© 2023 Imperas Software Ltd.

14-Sep-23

RISC-V Is Why We Are All Worried About Processor Verification

- RISC-V is taking over the processor world, except for x86
 - Yes, that includes Arm
- RISC-V processor customization means that every RISC-V developer needs to verify the RISC-V processor
- Lost art? Processor IP vendors guard their verification methodology and details more closely than the IP itself
 - With the verification flow, someone could reverse engineer a high quality processor
 - There are few public details about x86, Arm or Apple processor verification



Agenda



- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary





RISC-V and processor verification

- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

RISC-V Freedom Enables Domain Specific Processing



- Who: RISC-V users include traditional semiconductor companies, and embedded systems companies now practicing vertical integration by developing their own SoCs
- What: RISC-V is an open instruction set architecture (ISA), it is not a processor implementation
- Where: RISC-V is growing in market segments where x86 (PCs, data centers) and Arm (mobile) architectures are not dominant
 - Small microcontrollers for SoC management, replacing proprietary cores
 - Verticals such as IoT and automotive
 - Horizontal markets such as security and AI/ML
 - Deep embedded applications
- When: RISC-V processors are now used in over 30% of SoCs
- Why: The freedom of the open ISA enables users to develop differentiated domain specific processors and processing systems

Keys to RISC-V SoC Success



- 1) Processor IP source
 - Processor IP vendor
 - Open source IP
 - Build it yourself
- 2) Processor verification
- 3) Software porting, development, bring up, test
- All 3 areas need to account for the addition of custom features to the processor (*because everyone adds custom features to the processor*)

Keys to RISC-V SoC Success



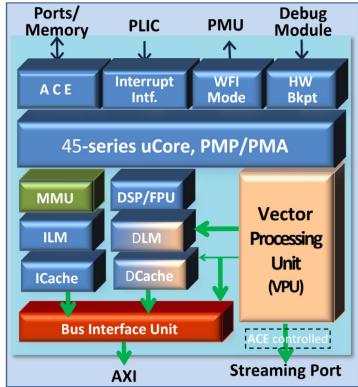
- 1) Processor IP
 - Processor IP vendor
 - Open source IP
 - Build it yourself

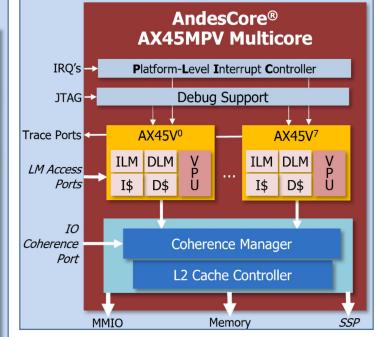
2) Processor verification

- 3) Software porting, development, bring up, test
- Users of all 3 types of processor IP need to account for the addition of custom features to the processor (*because everyone adds custom features to the processor*)
- Success in processor verification requires a high-quality model of the processor
- Success in processor verification requires innovative technologies and methodologies – *the lost art of processor verification*

RISC-V Processor Complexity

- RISC-V is a modular instruction set architecture
- Any extension (functional group of instructions, e.g. atomics, compressed, floating point, vector) can be added to the base processor
- Then add in interrupts, privilege modes, Debug mode, multi-hart (multi-core), etc. and it gets complex
- Then processor DV, tool chain development and other software development is needed

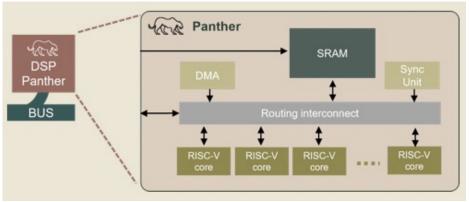




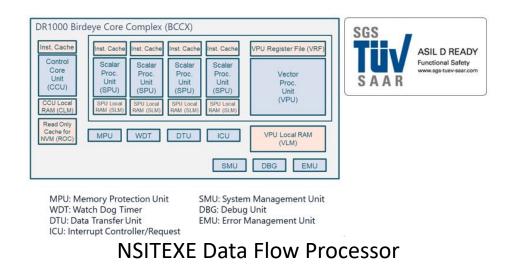
RISC-V Processing Subsystems Compound DV Issues

- Multi-processor subsystems are commonly being developed using RISC-V cores
- Application areas include DSP, AI/ML and packet processing
- This adds complexity to both the DV and software development tasks

mperas



Dolphin Design "Panther" DSP





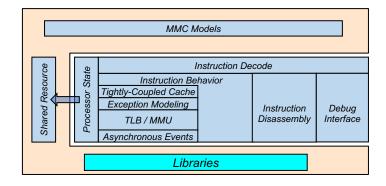


- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

RISC-V Model Requirements: Quality, Configurability, Interfaces



- Model the ISA, including all versions of the ratified spec, and stable unratified extensions
- Model other behavioral components, e.g. interrupt controllers
- Easily update and configure the model(s) for the next project
- User-extendable for custom instructions, registers, ...
- Model actual processor IP, e.g. Andes, MIPS, NSITEXE, OpenHW, SiFive, ...
- Well-defined test process for the model! including coverage metrics
- Interface to other simulators, e.g. SystemVerilog, SystemC, Imperas virtual platform simulators
- Interface to software debug tools, e.g. GDB/Eclipse, Imperas MPD
- Interface to software analysis tools including access to processor internal state, etc.
- Imperas models and simulators were built to satisfy these requirements, and matured through usage on non-RISC-V ISAs over the last 15+ years



Imperas OVP RISC-V Fast Processor Models

- Use cases
 - Architecture analysis, including (especially) custom instructions
 - Software development, debug and test
 - Processor and SoC verification
- Existing Imperas Open Virtual Platforms (OVP) Fast Processor Models of ...
 - Generic or envelope models of RV32/64 IMAFDCEVBHKPZ* M/S/U privilege modes
 - Models of processor IP vendors: Andes, MIPS, NSITEXE, OpenHW, SiFive, ...
 - Models for developers building their own processor
- Custom instructions easily added by user or by Imperas
 - New instructions are added in a side file so as not to perturb the verified model
 - Custom instructions are analyzed for effectiveness
- Models are built using Test Driven Development (TDD) methodology
 - Tests are built at the same time as features are added
 - Continuous Integration (CI) test flow used
 - > 15,000 tests for models + simulator
 - Additional testing by processor IP vendors to validate models



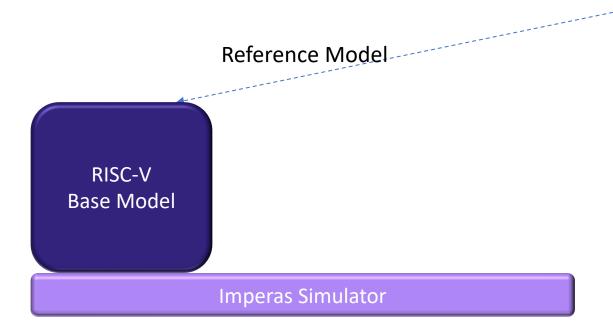
Imperas

Open Virtual Platforms

Page 12

Model + Simulator Architecture (1)

Imperas



Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs

Model + Simulator Architecture (2)

RISC-V Base Model Imperas Simulator

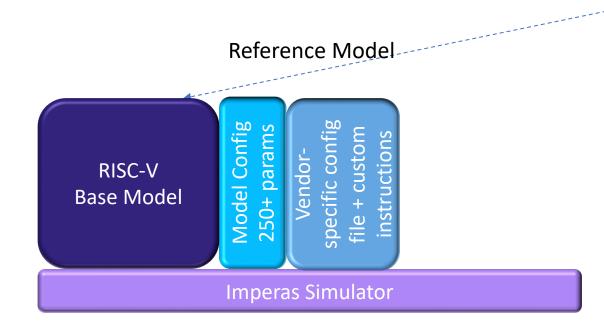
Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs

- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension

Model + Simulator Architecture (3)



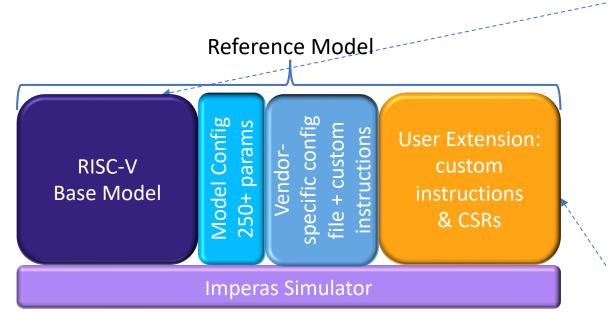


Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension
- For processor IP vendors, have pre-defined configuration plus vendor custom instructions

Model + Simulator Architecture (4)

Imperas



Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension
- For processor IP vendors, have pre-defined configuration plus vendor custom instructions

Imperas provides methodology to easily extend base model

- Custom instructions added using same APIs as in Base Model
- Separate source files and no duplication to ensure easy maintenance
- 100+ page user guide/reference manual with many examples
- User extension source can be proprietary (Apache 2.0 open source license)

Model + Simulator Architecture (5)

- RISC-V Base Model is used in all Imperas RISC-V processor models
 - By commercial users
 - By academic users
 - By users of the free ISS riscvOVPsimPlus
- RISC-V Base Model is used by > 150 organizations

Imperas develops and maintains Base Model

- Base Model implements RISC-V specification in full
- Base Model built using Test Driven Development methodology
- Built using public APIs matured over 15 different ISAs
- Simulator is separate from the model; supports the modeling APIs

Imperas

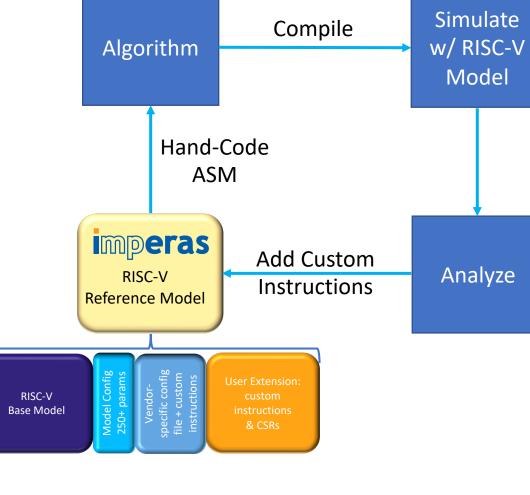
- Fully user configurable to select which ISA extensions
- Fully user configurable to select which version of each ISA extension
- For processor IP vendors, have pre-defined configuration plus vendor custom instructions

Imperas provides methodology to easily extend base model

- Custom instructions added using same APIs as in Base Model
- Separate source files and no duplication to ensure easy maintenance
- 100+ page user guide/reference manual with many examples
- User extension source can be proprietary (Apache 2.0 open source license)

Models Drive Customization

- Custom instructions are added to optimize a specific application or set of applications within a domain
- Models let you explore quickly
 - Much faster to develop than RTL
 - Better profiling information available
 - Easier to debug software
- Methodology
 - Start by characterizing the application to be optimized
 - Then add the custom instructions, evaluate, and iterate







- RISC-V and processor verification
- RISC-V processor models

RISC-V processor verification methodology

- Processor verification success
- Summary

5 Levels of RISC-V Processor DV Methodology

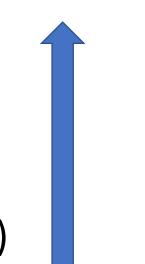
- 1) Asynchronous continuous compare
- 2) Synchronous step-and-compare
- 3) Post-simulation trace log file compare
- 4) Self-checking tests (e.g. Berkeley torture tests pre-2018)5) Hello World

Imperas

Quality

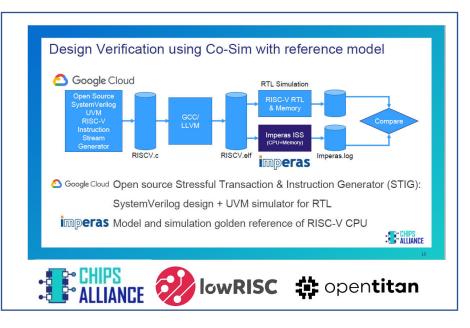
5 Levels of RISC-V Processor DV Methodology

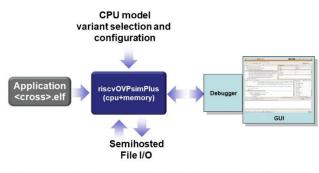
- **1)** Asynchronous continuous compare
- 2) Synchronous step-and-compare
- **3)** Post-simulation trace log file compare
- 4) Self-checking tests (e.g. Berkeley torture tests pre-2018)5) Hello World



3) Post-Simulation Trace Log File Compare (Entry Level DV)

mperas





Imperas riscvOVPsimPlus Reference Simulator

Process

- use random generator (ISG) to create tests
- during simulation of ISS write trace log file
- during simulation of RTL write trace log file
- at the end of both runs, run logs through compare program to see differences / failures
- ISS: riscvOVPsimPlus includes Trace and GDB interface
 - Free ISS: <u>https://www.ovpworld.org/riscvOVPsimPlus</u>
- ISG: riscv-dv from Google Cloud / Chips Alliance
 - Free ISG: <u>https://github.com/google/riscv-dv</u>

Post-sim Trace Compare: Pros and Cons



- Pros:
 - Availability of quality RISC-V simulators (e.g. riscvOVPsimPlus from Imperas)
 - 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

Post-sim trace compare is widely used

> Most effective as a complementary methodology to asynchronous continuous compare

riscvOVPsimPlus / riscvISATESTS: Commercial Users

Page 24



© 2023 Imperas Software Ltd.

peras

riscvOVPsimPlus / riscvISATESTS: **University & Research Lab Users**

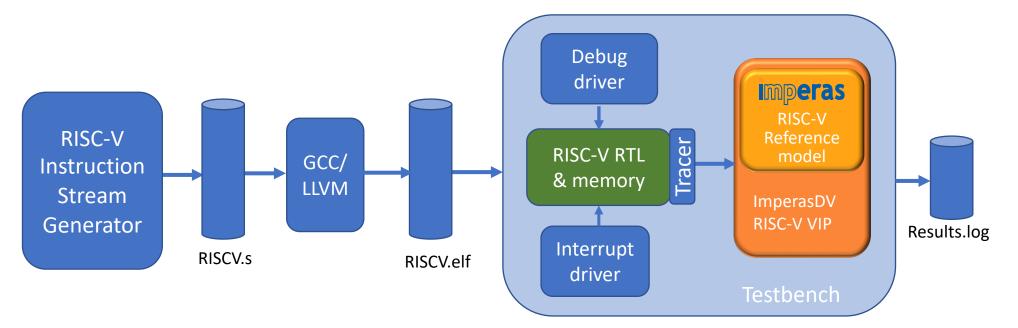




© 2023 Imperas Software Ltd.

14-Sep-23

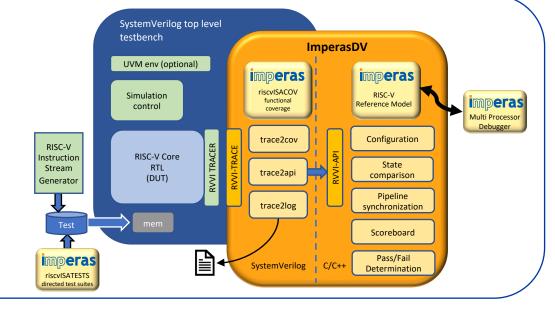
1) Async Continuous Compare (Highest Quality DV Methodology)



- RTL and reference model are run in "lock-step" in the same simulation
- Internal state of the two is continuously compared
- Asynchronous events are driven into the DUT
- Tracer informs reference model about async events
- Verification IP handles async events, scoreboarding, comparison, pass/fail

ImperasDV Components

- Reference model needed for comparison of correct behavior
- Verification IP provides ease of use, saves time and resources
- RVVI standard provides communication between test bench and reference model subsystem
- riscvISACOV: functional coverage modules
- Test suites: riscvISATESTS, directed test suites for difficult extensions
- MultiProcessor Debugger (MPD) enables RTL-reference model co-debug
- Feature selection and design choices require serious consideration due to implications of every decision
 - Every addition dramatically compounds verification complexity
 - Adds schedule, resources, quality costs == big risks
- Before 2021, no off-the-shelf toolkit/products available for DV of processors ... then came ImperasDV
- ImperasDV, with async continuous compare methodology, is needed to support features such as interrupts, privilege modes, Debug mode, multi-hart, multi-issue and OoO pipeline, ...



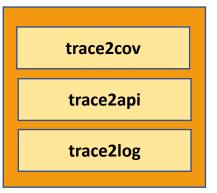
Async Continuous Compare Pros and Cons

mperas

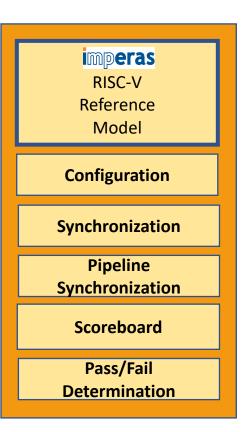
• Pros:

- Instruction by instruction comparison
- Comparison of execution flow, program data, internal state
- Errors are flagged immediately no runaway simulations
- Detects synchronous and asynchronous bugs
- Checking is done for you
- Verification IP (VIP) is reusable across different core DV projects
- Ease of use
- Training, documentation, and support
- Cons:
 - Cost of VIP licenses

ImperasDV: Verification IP



- Data prep for functional coverage
- Data movement from SystemVerilog to C verification IP
- Logging data

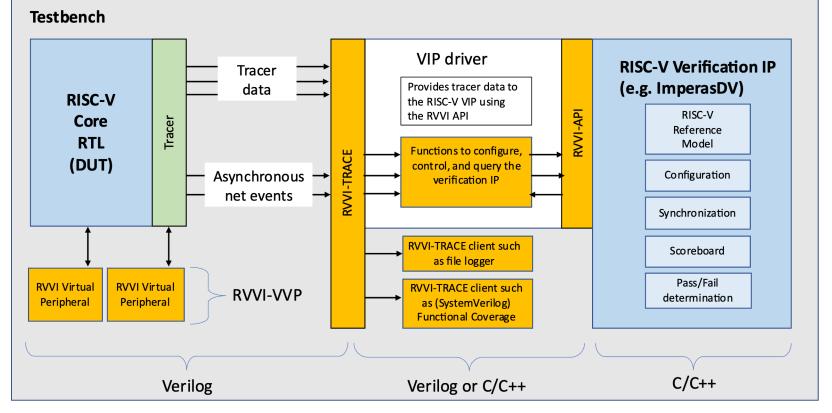


- Reference model encapsulation
- Includes DUT reference state storage
- Includes synchronization technology
 - Can run sync, async, interrupts, debug, multi-hart
- Pipeline synchronization is key for asynchronous event DV
- Includes comparison technology
 - Comparisons are done on DUT/Reference Model processor events; enables DV of multi-issue and OoO pipeline processors

RISC-V Verification Interface (RVVI)

Imperas

- Standardize communication between DUT, testbench, and RISC-V VIP
 - RVVI-TRACE: provides tracer data to RISC-V VIP
 - RVVI-API: function level interface to RISC-V VIP
 - **RVVI-VVP**: virtual peripherals
- Collaborative work has evolved over 3 years
 - Imperas, EM Micro, SiLabs, OpenHW Group

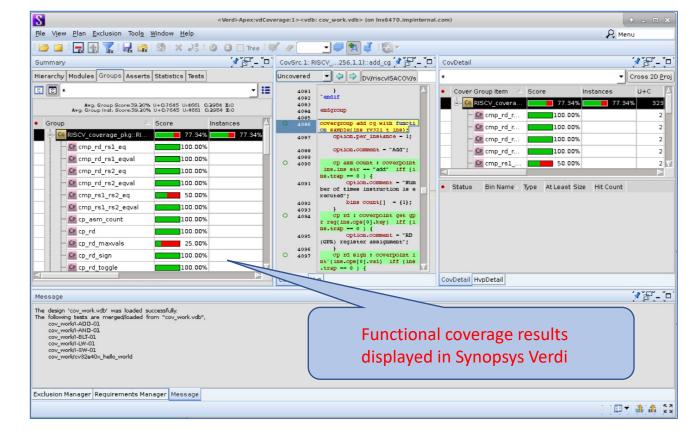


https://github.com/riscv-verification/RVVI

Functional Coverage is a Key Verification Metric Especially True for Processors

```
covergroup add_cg with function sample(ins t ins);
    option.per instance = 1;
    cp rd : coverpoint get gpr name(ins.ops[0].key, "add") {
    cp rd sign : coverpoint int'(ins.ops[0].val) {
       bins neg = \{[\$:-1]\};
       bins zero = \{0\};
        bins pos = {[1:$]};
    cp rs1 : coverpoint get gpr name(ins.ops[1].key, "add") {
    cp_rs1_sign : coverpoint int'(ins.ops[1].val) {
       bins neg = {[$:-1]};
       bins zero = {0};
        bins pos = {[1:$]};
    cp rs2 : coverpoint get gpr name(ins.ops[2].key, "add") {
    cp rs2 sign : coverpoint int'(ins.ops[2].val) {
        bins neg = {[$:-1]};
       bins zero = \{0\}:
        bins pos = {[1:$]};
   cr rs1 rs2 : cross cp rs1 sign, cp rs2 sign ;
    cmp rd rs1 eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rd eq rs1 = \{1\};
        bins rd ne rs1 = \{0\};
    cmp rd rsl eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rd equal rs1 = \{1\}:
        bins rd neval rs1 = {0};
    cmp rd rs2 eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rd eq rs2 = \{1\};
        bins rd ne rs2 = \{0\};
    cmp rd rs2 eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rd eqval rs2 = {1};
        bins rd neval rs2 = \{0\};
   cmp rs1 rs2 eq : coverpoint ins.ops[0].key == ins.ops[1].key {
        bins rs1 eq rs2 = \{1\};
        bins rs1 ne rs2 = \{0\};
   cmp rs1 rs2 eqval : coverpoint int'(ins.ops[0].val) == int'(ins.ops[1].val) {
        bins rs1 equal rs2 = \{1\};
        bins rs1 neval rs2 = \{0\};
endgroup
```

mperas



riscvISACOV (and ImperasDV) works with any SystemVerilog simulator

© 2023 Imperas Software Ltd.

RISC-V Functional Coverage



For a processor there are different types of functional coverage required:

- Standard ISA architectural features
 - unpriv. ISA items: mainly instructions, their operands, their values
 - => these are standard and the same for all RISC-V processors it is the spec...
- Customer core design & micro-architectural features
 - priv. ISA items, CSRs, interrupts, debug block, ...
 - pipeline, multi-issue, multi-hart, ...
 - Custom extensions, CSRs, instructions

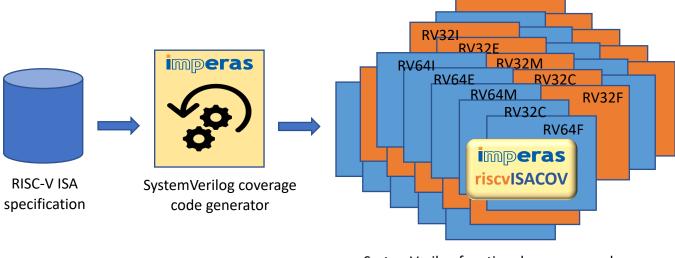
Functional Coverage of RISC-V Instructions: Scope



- There are many different instructions in the RV64 extensions:
 - Integer: 56, Maths: 13, Compressed: 30, FP-Single: 30, FP-Double: 32
 - Vector: 356, Bitmanip: 47 Krypto-scalar: 85
 - P-DSP: 318
 - For RV64 that is 967 instructions...
- Each instruction needs SystemVerilog covergroups and coverpoints
 - 10-40 lines of SystemVerilog for each instruction
- 10,000-40,000++ lines of code to be written
 - Not design or core specific

riscvISACOV Is Automatically Generated SystemVerilog Functional Coverage





SystemVerilog functional coverage code

- riscvISACOV provides functional coverage of Instructions and operands
- Roadmap includes CSRs and data hazards
- Imperas tools can automatically generate functional coverage code for custom instructions

Test Stimuli

- Instruction Stream Generator (ISG) and/or directed tests
- ISG generates test programs using constrained random approach
 - Most often obtain the ISG:
 - Commercial such as Valtrix STING
 - Open source such as Google riscv-dv
 - Require toolchains like GCC, LLVM for assemblers, linkers
 - Require functional coverage so that you know what you've tested!
- Directed tests
 - Imperas have developed a directed RISC-V test generator, instruction coverage verification IP and a mutating fault simulator (for test qualification) to provide high quality test suites
 - The generated tests suites are targeting architectural compatibility as defined in the RVIA architectural test working group coverage requirements
 - Free Imperas architectural validation test suites (50+), including RV32/64 I, M, C, F, D, B, K, V, P
 - <u>https://github.com/riscv-ovpsim/imperas-riscv-tests</u>
 - Imperas commercial directed test suites for vector extension, protected memory components
 - Can support any RISC-V vector or PMP configuration; the user selects the configuration and Imperas generates the test suite

ImperasDV: Debug with MPD

- Imperas MPD is an Eclipse based debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers
- Break at the first mismatch, debug SW and RTL concurrently

Testbench	
RISC-V Core ⋝	ImperasDV
Core	
	Imperas Platforms (mpd) Imperas Platforms (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpdexx Memor (mpd) Imperas - Connect to running simulatori mpd Memor (mpd) Imperas - Connect to running simulatori mpd Memor (mpd) Imperas - Memor (mpd) Memor (mpd) Impe
	Visit Www.JMPERAS.com for multicore debug, verification and i Imra 0x.2 < start#65 Info (MPD_SCS) Connecting Imrsp 0x400000 Info (MPD_SCS) Socket connected Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) Socket connections on port 4131: Imrsp 0x3d18 Info (MPD_SCS) (protoconnections) Imrsp 0x3d18
	imperas

Software Debug and Analysis Tools Automatically Work With the Custom Instructions

3 - 🗑 🗞 🔁 🕸 - O - 🏊 - 🎒 🛷 - 🔄 🖉 🖿 🗉 🛎	1 3. 3	.e 📭 - 🛛 🖬 🖷	92 94 + 94 + 🕫	🗘 * 🗢 🕤 🛛 Quick Ac	cess 📑	*
to Debug X 🗢 🗖		🕬 Variables 🕮 💁 Breakpoints 🚟 Registers 🛋 Modules				
後 🍽 🖬 🕺 🔍 🐼 🦄 👘 十月一日 🙁 3	t 🥎 🔻			E 🐴 🖯		4
im Platform Launch [Imperas - Connect to running simulator]		Name	Туре	Value		
⊽ km iss		69- input	unsigned int	2222400358		
🗢 🔐 cpu0 [RV32IM riscv]	to word	unsigned int	2804990272			
▽ P ID #1 [cpu0] RV32IM riscv (Suspended : Breakpoint)		w res	unsigned int	0		
processLine() at test_custom.c:5 0x10230						
main() at test_custom.c:32 0x102e4						1
🚚 mpd						
		JIG				9
test_custom.c 🛙 🗂 customChaCha20. 📑 riscv32.c 🛛 💽 _start() at	0x1 *1	🖻 🗄 📴 Out	line 🖮 Programmers	View 🔛 Disassembly 🛙		0
// Custom instruction test for Chacha20		Enter	ocation here 🗸 👔 🔞		~	
<pre>#include <stdio.h></stdio.h></pre>		00010		my a0.a5		
unsigned int processLine(unsigned int input, unsigned int word)	{	00010		lw a5,-40(s0)		
unsigned int res = input;	S		0244: 00078593	mv a1,a5		
<pre>asmvolatile("mv x10, %0" :: "r"(res)); asmvolatile("mv x11, %0" :: "r"(word));</pre>		00010	0248: chacha20qr1 024c: chacha20qr2	a0,a0,a1 a0,a0,a1		
asm _volatile_(".word 0x00B5050B\n" ::: "x10"); // QR	1	00010		a0,a0,a1		
<pre>asm _volatile_(".word 0x00B5150B\n" ::: "x10"); // 0R</pre>			254: chacha2.gr4	a0,a0,a1		
<pre>asmvolatile(".word 0x00B5250B\n" ::: "x10"); // QR</pre>		00010		a0,a0,a1		
<pre>asmvolatile_(".word 0x00B5350B\n" ::: "x10"); // QR</pre>		00010	025c: chacha20qr3 0260: chacha20qr3	a0,a0,a1 a0,a0,a1		
<pre>asmvolatile(".word 0x00850508\n" ::: "x10"); // QR asmvolatile(".word 0x00851508\n" ::: "x10"); // QR</pre>		00010		a0,a0,a1		
asmvolatile_(".word 0x00B5250B\n" ::: "x10"); // 0R		- 00010		mv a5,a0		~
G			C			D
		Dennel M Ca	Della Deskla Offic	and making Princes Of	Mamar D	
	0	Consol 23	asks 👔 Proble 🔮 Ex	ecut 🖉 Debug 🍞 iProf 🚺	Memor -	
atform Launch [Imperas - Connect to running simulator] mpd.exe (7.5)	~					
<pre>gned int), 1, fp)) { ebug (cpu0) > 32 res = processLine(res, word</pre>	· · ·	No consol		minstructio	22	
		INE	ew cusio	m instructio	ns,	
	test custo					
	test_custc					
<pre>lebug (cpu8) > 52 ebug (cpu8) > processLine (input=2222400358, word=2804990272) at unsigned int res = input; lebug (cpu8) ></pre>	test_custc	ne	w additi	onal state re	giste	r

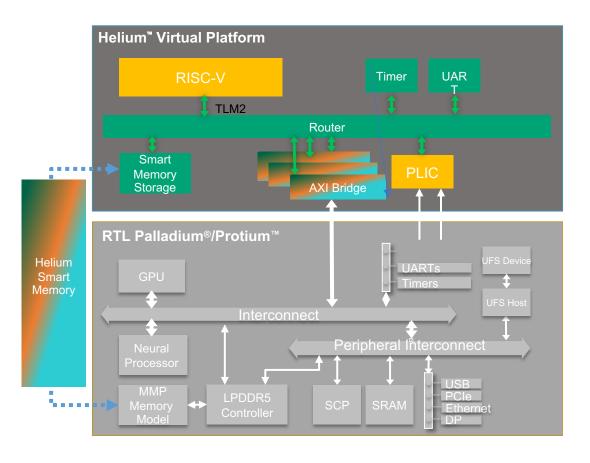


CpuManagerMulti started: Thu Aug 23 12:02:30 2018

Info (OR_OF) Target 'iss/cpu0' h	as object file re	ad from 'a	plication/t	est_custom.	RISCV3	2.elf'
Info (OR_PH) Program Headers: Info (OR_PH) Type Offs	et VirtAddr	DL	Fit-Fi-	Hardin	F1	01
	000000 0x00010000			MenSiz 0-00017270	Flags	
	017270 0x0002B270					1000
Info (OR_OF) Target 'iss/cpu0' h						
Info (OR_PH) Program Headers;	as object file re	ad from a	plication/e	xception.KI	SCV52.4	914
	at Unable	Dhumbalde	Eul-Di-	Hartin	F1	014.00
Info (OR_PH) Type Offs		PhysAddr	FileSiz	MenSiz		Align
Info (OR_PD) LORD 0x00 Info 1330: 'iss/cpu0', 0x0000000	001000 0x00000000					1000
Info 1331: 'iss/cpu0', 0x0000000 Info 1332: 'iss/cpu0', 0x0000000	000010220(process	Line*10);	CD42C25 SW	a5,-36		
Info a5 a730c140 -> 84772366	000010520/brocess	C106-14/1	0041102 16	ao,-30	(20)	
Info 1333: 'iss/cpu0', 0x0000000	00001/074/ monean	institute i	-E49697	a5,-20	(-0)	
Info 1334: 'iss/cpu0', 0x0000000				45,-20		
Info 1335: 'iss/cpu0', 0x0000000	00001023c(process	Line+20);	0079517 mi	a0, a5	(20)	
Info 1336: 'iss/cpu0', 0x0000000				a5,-40	1001	
Info a5 84772366 -> a730c140	ANALINCAN HE OCESS	2110-24/1	0042103 18	40, 40	1001	
Info 1337: '1ss/cpu0', 0x0000000	000010244(mercent	(ine+28)+ i	0079593	a1.a5		
Info 1338; 'iss/cpu0', 0x0000000						
Info a0 84772366 -> e2262347	onortorao/huncess	F106-2011	induniación a la	07707701		
Info 1339; 'iss/cpu0', 0x0000000	00001024c1 percent	(ine+20)+ .	hachs20m2	te Ge Ge		
Info a0 e2262347 -> 5e207451	opport reaction press	Line-30/+	criacinazorgi z 1	ao, ao, ar		
Info 1340; 'iss/cpu0', 0x0000000	000010250/ recorded	(ine+2d) .	hacha20m2	fe Ge Ge		
Info a0 6e207451 -> 106511c9	onorther oness	stile-34/1	and under over a l	do 1 do 1 dr		
Info 1341: 'iss/cpu0', 0x0000000	000010254(process	iner38) .	hacha20gr4	In. 0a. 0a		
Info a0 10b511c9 -> c2e844db	onorace and the press	6.110-207+	and a state of the	00700704		
Info 1342: '1ss/cpu0', 0x0000000	00001025B(process	(ine+3c):	hacha20gr I	1a. 0a. 0a		
Info a0 c2e844db -> 859b65d8	erer and the second			and and and and		
Info 1343: 'iss/cpu0', 0x0000000	00001025c (process	ine+40):	hacha20gr	1s.0s.0s		
Info a0 859b65d8 -> ba49822a			a local data of a local			
Info 1344: 'iss/cpu0', 0x0000000	000010260(process	ine+44); ;	hacha20gr	a0.a0.a1		
Info a0 ba49822a -> 79436a1d						
Info 1345: 'iss/cpu0', 0x0000000	000010264(process	Line+48); /	thacha20gr4	1a.0a.0a		
Info a0 79436a1d -> 39d5aeef						
Info 1346: 'iss/cpu0', 0x0000000	000010268(process	Line+4c); I	00050793 mu	a5.a0		
Info a5 a730c140 -> 39d5aeef	1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.	110000000000000				
Info 1347: 'iss/cpu0', 0x0000000	00001026c(process	Line+50); !	fef42623 sw	a5,-20	(s0)	
Info 1348: 'iss/cpu0', 0x0000000	000010270(process	Line+54): 1	fec42783 1w	a5,-20	(a0)	
Info 1349: 'iss/cpu0', 0x0000000				a0,a5		
RES = 84772366						
Info						
Info	*******					
Info CPU 'iss/cpu0' STATISTICS						
	iecs .			•		
Info Nominal HIPS : 1	Men New		tom	Inst		TIO
Info Final program counter : 0	1100			1130		
Info Simulated instructions: 6						
	^{209,} in tr	200	disas	com	hh	
Info				SELL		
Info			a1343			V

Hybrid Simulation-Emulation for HW-SW Co-Verification

- It takes more than just the RTL simulator for comprehensive processor verification
- An additional tool is the hardware emulator, e.g. Cadence Palladium
- The interface to Palladium is via the Cadence Helium SystemC simulator



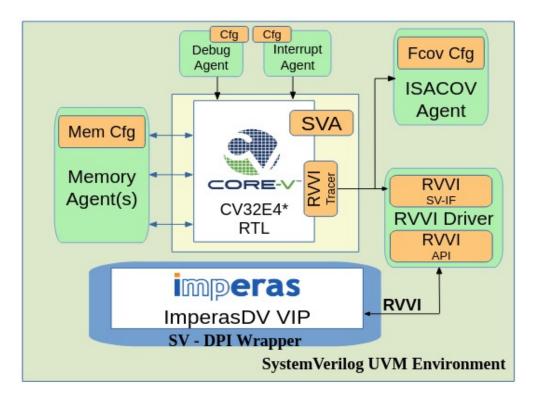
Agenda



- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

RISC-V Processor DV Results

- OpenHW Core-V-Verif
 - OpenHW users now on 3rd generation of Core-V-Verif DV flow
 - CV32E40P successfully taped out
 - CV32E40Pv2, CV32E40S, CV32E40X, CV32E20 using this flow today; all expected to complete DV this year
 - OpenHW "Wally" core
- Other successful DV projects using Imperas include Codasip, MIPS, Nagra, NSITEXE, Nvidia Networking, Skyworks, ...



Case Study: Wally RISC-V Core

- Configurable core:
 - RV32I, RV32E, RV64I, RV64E
 - A, C, F, D, M extensions, privileged modes, CSRs
 - MMU/TLB virtual memory, caches
- Developed at Harvey Mudd College / Oklahoma State University
 - Focus: high quality core for processor architecture education
 - Now in OpenHW as CORE-V Wally (<u>https://github.com/openhwgroup/cvw</u>)
- Status in January 2023 before starting to use RVVI + ImperasDV for verification:
 - Passing all RISC-V International compliance tests, Imperas compatibility tests
 - Using Compliance Level post-simulation signature file compare
 - Boots Linux

Page 41



Wally + RVVI – Status (July 2023)

RVVI Tracer + testbench integration: 3 days of effort

Results:

- 20+ bugs found in simulation almost immediately using ImperasDV and riscv-dv
- Reached Linux prompt with continuous checking: 2 days of simulation
- One bug found just *after* the Linux command prompt (!)
- Functional coverage achieved by booting Linux: covergroups 37% (bins 3%)
- Future work:
 - Boot Linux with co-sim using hardware assisted verification
 - Achieve 100% functional coverage using constrained-random tests

popenhwaroup / cvw (Public ● Issues 6 11 Pull requests Discussions Actions Projects ① Security Insights Releases / CVW v0.9

Arty A7 board support, ImperasDV Linux boot Latest

• 150 commits to main since this release 🛛 CVW_v0.9 - e43de9c 🥝 **Figure 3 Figure 3**

Imperas

Additional support for Arty A7 FPGA board.

Lock step simulation with ImperasDV completes Linux boot from ground zero to 542M instructions.

Fixes bug 203 and linux/ImperasDV mismatch at 571M instructions #304

Solution openhygroup:main from ross144:main [] last week

<> Code



Agenda



- RISC-V and processor verification
- RISC-V processor models
- RISC-V processor verification methodology
- Processor verification success
- Summary

Processor Verification: The Key to the RISC-V Castle

- High quality reference models
- Asynchronous continuous compare methodology
- Verification IP
- Verification standards (RVVI)
- Verification metrics (functional coverage)



mperas

Thank you

Larry Lapides LarryL@imperas.com