mperas

Compliance, Verification and Customization of RISC-V Cores and SoCs

Getting Started with RISC-V – North America Roadshow Waltham (Boston) – Austin – Irvine – Milpitas (Bay Area) : April 2019

© Imperas Software Ltd.

Imperas Focus



 "nobody designs a chip without simulation", at Imperas we believe that:

"nobody should develop embedded software without simulation"

 Imperas develops simulators, tools, debuggers, modelling technology, and models to help embedded systems developers get their software running...

and hardware developers get their designs correct

- 10+ years, self funded, profitable, UK based, team with much EDA (simulators, verification), processors, and embedded experience
- www.imperas.com/riscv



- Compliance and verification
- Customization
- Summary



Agenda



- Compliance and verification
 - Why is compliance important for RISC-V?
 - RISC-V compliance checking
 - Verification
- Customization
- Summary

Compliance for RISC-V is Important

Q: What is meant by "compliance"?

A: The device works within the envelope of the agreed specifications

Q: Is there an easy process or path to follow to develop methodologies/tools for compliance?

A: NO – all established ISAs are single company controlled and those companies work extremely hard on proprietary solutions to ensure that all designs that go out their door work correctly – so RISC-V has to pioneer compliance collectively

Q: What happens if the RISC-V industry builds devices that are not complying with specifications?

A: Users cannot assume that tools like C compilers, operating systems, and application software will be transferable across devices and work correctly

The RISC-V Foundation's Compliance Task Group



- Jun 2017 group started
 - Compliance testing is a testing technique which is done to validate whether the system developed meets the prescribed standards or not. It is **not design verification** testing
 - compliance testing is looking for issues like missing registers, modes, instructions not for bugs in RTL implementations...
 - The tests have to be written to ensure compliance/non-compliance is observable in a test signature. The signatures are published so that the user does not have to run a reference model and can compare the results of their target runs to the reference signature
- Jan 2018 initial rv32i test suite provided by Codasip
- Jun 2018 Imperas, Embecosm worked on GitHub, made repo public
- Oct 2018 Imperas improved test coverage, added new suites, ported 32bit riscv-tests

RISC-V Compliance Suite



- It is 'work in progress'
- Two components
 - Test suites
 - Each suite focuses on a feature set of the RISC-V envelope
 - Initial focus is instructions, user mode spec, e.g. rv32i, rv32im, rv32imc, rv64i, ...
 - Awaiting RISC-V platform specifications to subset privilege spec, before starting privilege suites
 - Framework
 - Make, bash, and scripts
 - Encapsulate compiler tools, linkers, simulators, and targets (Devices Under Test)
 - Includes simulator: as example target, and to generate reference signatures
 - Run: Select suite and target
 - Runs each test, target produces signatures, compares to saved golden reference signature
- Available: www.github.com/riscv/riscv-compliance

Compliance Suite Status (rv32i)

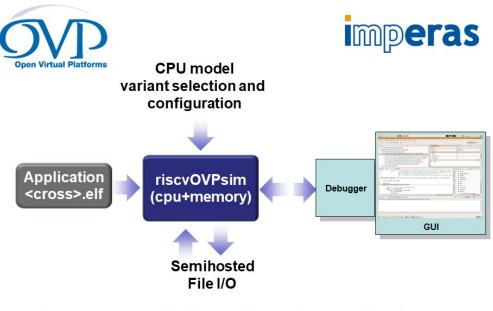
Instruction	Decode Coverage	beg 95%
lb	100%	bne 95%
lh	100%	b1+ 05%
lw	100%	bge 95% - Need more tests
lbu	100%	bltu 95%
lhu	100%	bgeu 95%
sb	100%	
sh	100%	slti sltiu 77% - Investigate (more tests?)
SW	100%	fence 0%
sll	100%	fence i O%
slli	100%	scall Os > no tests yet
srl	100%	
srli	100%	rdcycle 0% - (fence.i not in 2.3 RV32I)
sra	100%	rdcycleh 0%
srai	100%	ndtime 08
add	100%	rdtimeh pseudos+CSR, no tests yet
addi	100%	rdinstret 0% (not in 2.3 D\/221)
sub	100%	rdinstret 0% (not in 2.3 RV32I)
lui	100%	
auipc	100%	
xor	100%	Notes on rv32i test suite:
xori	1000	
or	100%	 54 hand coded directed tests (average 150 instructions each)
and	100%	https://github.com/riscv/riscv-compliance/tree/master/riscv-test-suite/rv32i/src
andi		
slt	100%	Notes on Decode Coverage
sltu	100%	 Decede Coverege: observe obserges on all bits of legal decedes
	100%	 Decode Coverage: observe changes on all bits of legal decodes
] jr	100%	Decode Coverage data from the Imperas Fault Simulation Coverage tool
mv	100%	
ALL Y	1006	 ran 478,390 simulations in 308 secs

Similar status available on RV32IM, RV32IMC, RV64I, RV64IM suites

Coverage Metrics for Compliance Tests

- Coverage metrics used in RTL design verification are *not applicable* as they are often functional and connected to specific microarchitecture (RTL)
- Imperas' Code Coverage Tool provides coverage of model source, which is useful to see how much of model is exercised by tests but *does not* show how much of specification is covered
- Imperas' Fault Simulation Coverage tool provides Instruction Decode Coverage
 - explores the decodes of the instructions and mutates legal bits
 - detects that there is a test that stimulates & observes each bit
 - tool is very fast, runs in parallel, provides other metrics including data coverage
 - very useful for users as *provides coverage of custom instructions*

riscvOVPsim simulator



Imperas riscvOVPsim Compliance Simulator

- **I**mp**eras**
- Instruction Accurate simulator using high performance Just-in-Time Code Morphing that executes RISC-V binaries and runs on Linux/Windows PC
 - Runs very fast, 1,000 MIPS
- Industrial quality for use in test development, software development, compliance testing and design verification
 - Includes capabilities to perform RISC-V compliance signature generation
 - Maintained and supported by Imperas Software (www.imperas.com)
 - Simulator restricted to single processor model and fully populated memory
 - Includes Apache 2.0 Open Source model of complete RISC-V ISA envelope model of 2.2, 2.3, 1.10, 1.11 revisions of the RISC-V Foundation specifications (not hypervisor/vectors – not stable)
- FREE. No license keys or license management
- Video: http://www.imperas.com/riscvovpsim-a-complete-risc-viss-for-bare-metal-software-development-and-specificationcompliance

Issues to be sorted by RISC-V Foundation's Compliance Working Group

- Detailed specifications needed
 - the structure of the Compliance Suite Framework (and how used)
 - what content of test should be (how written)
 - list of all the tests needed in each test suite (what is covered of specification)
- Formal process for RISC-V Foundation to allow user to claim their processor is a RISC-V processor
 - i.e. process to explore and record 'is it a RISC-V compliant design?'
 - the 'rubber stamp: RISC-V inside'

Imperas: Exploring designs



Note: these notes are Imperas exploring designs brought into Imperas, not a discussion of what our customers are doing with our tools & test suites

- Many candidate DUTs (Device-Under-Test) = customers, partners, & for fun...
 - proprietary RTL, open source RTL
 - FPGA
 - Silicon
 - Simulators
- Process goal is to load .elf file, run, write signature, compare with golden reference
- Develop encapsulation of DUT
 - so can run public GitHub Compliance Suite and Imperas internal compliance test suites
- Different levels of complexity and challenge
 - ISA Simulators relatively easy (as can read/write files)
 - RTL simulators a little harder, 2 approaches
 - Memory read/write
 - GDBserver control as if hardware, inject, extract data
 - Hardware (FPGA, silicon)
 - Pod, JTAG, GDBserver type approaches to get data in/out

What did we find...



- Many candidate DUTs (Device-Under-Test) = customers, partners, & for fun...
 - proprietary RTL, open source RTL
 - FPGA
 - Silicon
 - Simulators
- Missing registers
- Missing instructions
- Floating point mode change issues
- PMP implementation issues

. . .

Verification



- Two key issues in verification of SoCs based on open ISA processors:
- **1**. Verification of the processor
- 2. Verification of the interface between the processor and the NoC/system bus
- Needed:
 - Reference model of the processor
 - Test generation
 - Processor instruction coverage measurement

Agenda

- Compliance and verification
- Customization
 - Why customize the ISA?
 - How to add custom features
- Summary



There are and will be many different RISC-V CPU developers:



- Deliver RTL IP as a business (like Arm, MIPS, just for RISC-V)
 - e.g. Andes, CloudBear, Codasip, Incore, Syntacore, ...
- Develop internal cores for SoCs
 - e.g. Nvidia, ...
- Develop for internal use but make available as open source
 - e.g. Western Digital, ...
- Develop and use open source as a business opportunity
 - e.g. SiFive, ...
- Download and use open source RTL in their products
 - e.g. Greenwaves, Google, ...

...

RISC-V CPU Adopters: (above & below the line)



New architectures (ML, IoT), arrays of processors, custom features, high performance, feature control (efficiency), large SoC design, architecture innovation, ...

Key requirement: 'freedom to innovate'

Researchers, education, open source community, freely available, no license cost or restriction, ...

Key requirement: 'free'

Many (above the line) adopters of RISC-V want to add their own custom extension instructions



- Traditional ISA choice has been hard if you want to add your own custom processor instructions to an ISA
- RISC-V as an open standard has specific regions of instruction decode space specifically allocated for users to add their own instructions
- Challenges
 - How to choose the processor IP starting point
 - How to add instructions to processor RTL
 - How to verify the complete RTL
 - How to evaluate effectiveness and performance gains of new instructions
 - How to enable software development utilizing the new instructions

Key Challenge of Optimizing Custom Instructions Requires New Methodology, Tools



- How to choose the processor IP starting point
- How to add instructions to processor RTL
- How to verify the complete RTL
- How to evaluate effectiveness and performance gains of new instructions
- How to enable software development utilizing the new instructions

Extend instruction accurate simulation tools and models to support analysis and optimization of custom instructions

Agenda – Custom Instructions



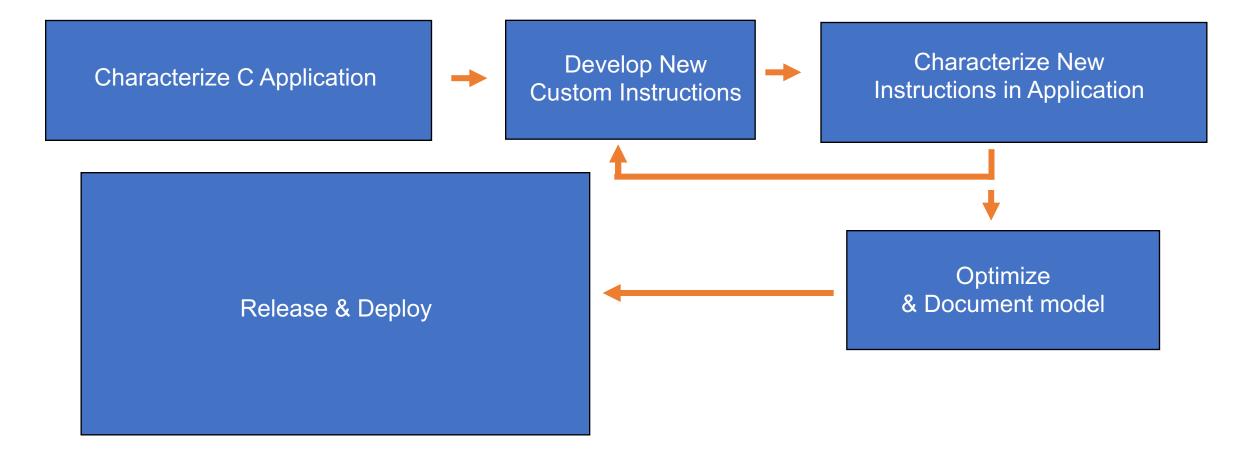
- Who is adopting RISC-V and why...
- Adding custom instructions to RISC-V processors
- Custom instruction optimization flow

Custom Instruction Optimization

- Show the flow used when designing new instructions to improve performance of applications running on a RISC-V processor
 - Allows evaluation and firming up of the instructions
 - Gets to the specification of the instructions needed to be implemented in RTL
- Introduce the technologies & tools needed for each stage
- Application software used for this walk-through is a character stream encoder, based on ChaCha20 encryption algorithm
 - Instruction Extensions to RISC-V courtesy of Cerberus Security Laboratories Ltd
 - https://cerberus-laboratories.com

Flow to add new custom instructions



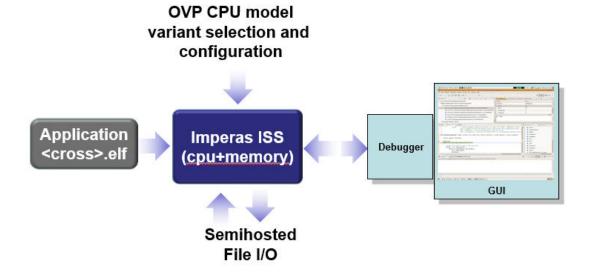


© Imperas Software Ltd.

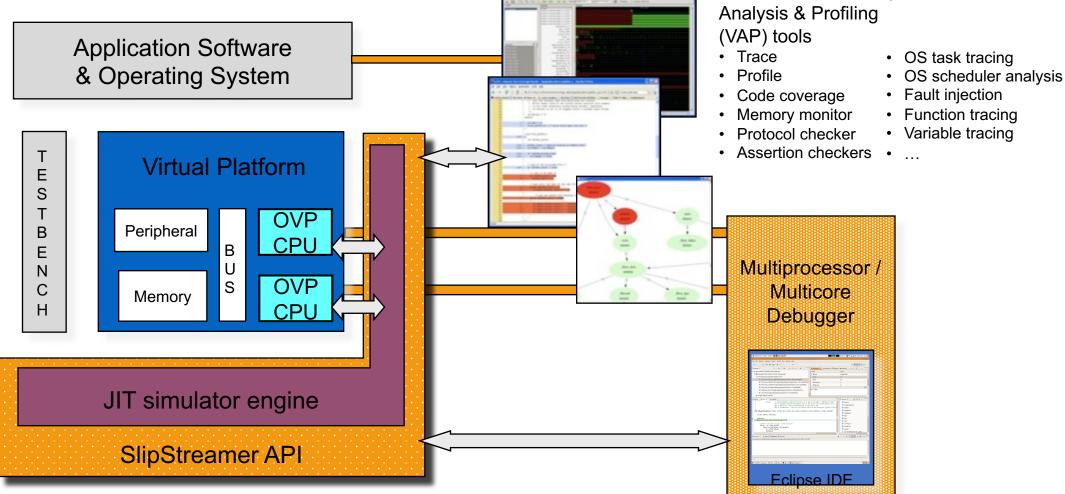
April 2019

The first thing needed is a simulator – think ISS, however ...

- Simulator and model need to have ability to extend to allow custom instructions and new tools
- Ease of use also important
- Need to have a business-friendly open source license



Imperas Environment for **Embedded Software Development**, Debug & Test



Software Verification,



Flow to add new custom instructions

Characterize C Application

- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling



Instruction Accurate simulation of C application

- Cross compiled C application targeting RV32IM
 - Character stream encoder, with ChaCha20 encryption algorithm
- IA simulation
 - Imperas RISC-V ISS with configurable model of RISC-V specification selecting RV32IM
- Semihosting
 - Enables bare metal application to very simply access host I/O

≻runs fast

- Over 1 billion instructions a second (standard PC) -
 - Linux and Windows supported host OS

unsign	ed int processLine(unsigned int res, unsigned int word){
	s = grl c(res, word)	
	s = qr2 c(res, word)	
re	s = qr3 c(res, word)	
re	s = qr4 c(res, word)	
	s = grl c(res, word)	
	s = qr2 c(res, word)	
	s = gr3 c(res, word)	
	s = qr4 c(res, word)	
	turn res;	
}		
int ma	in(void) {	
		a = "application/custom.data";
	LE *fp = fopen(custo	
	(fp) {	
-	unsigned int res	= 0x84772366:
	unsigned int word	
	unsigned int cnt=	
	unsigned int iter	
	while (iter++ < 16	
		Sword, sizeof(unsigned int), 1, fp)) {
		cessLine(res, word);
	}	
	rewind(fp);	CpuManagerMulti (32-Bit) v9999999 Open Virtual Platform simulator from www.IMPERAS.com. Copyright (c) 2005-2018 Imperas Software Ltd. Contains Imperas Proprietary Information.
	1 rewrite(rp),	Licensed Software, All Rights Reserved.
	fclose(fp);	Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.
	printf("RES = MAR	CpuManagerMulti started: Thu Aug 23 11:19:21 2018
1	else {	
1	printf/"Failed to	Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_c.RISCV32.elf'
}	princit raited to	Info (OR_PH) Program Headers:
1		Info (OR_PH) Type Offset VirtAddr PhysAddr FileSiz MemSiz Flags Alig Info (OR_PD) LOAD 0x00000000 0x00010000 0x000173c8 0x000173c8 R-E 1000
	turn 0;	[Info (UK_P)] LURD 0x000175c8 0x000285c8 0x0000285c8 0x000009c0 0x00000a24 RM- 1000
1	curn o,	Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
1		Info (OR_PH) Program Headers: Info (OR_PH) Type Offset VirtAddr PhysAddr FileSiz MemSiz Flags Alig
		Info (GR_PD) LGAD 0x00001000 0x00000000 0x00000000 0x00000000
15		RES = 84772366 Info
		Info
		Info CPU 'iss/cpu0' STATISTICS Info Type : riscv (RV32IM)
		Info Nominal MIPS : 100
		Info Final program counter : 0x100ac
		Info Simulated instructions: 1,289,380,976 Info Simulated MPS : 1151,2
		Info
		Info
		INFO SIMULATION TIME STATISTICS
		Info Simulated time : 12,89 seconds
		Info User time : 1.10 seconds Info System time : 0.02 seconds
		Info Elapsed time : 1.14 seconds
		Info Real time ratio : 11.31x faster

Cycle Approximate simulation C application

- Same C application
- IA simulation + timing Estimation (IA+E)
 - Includes annotated timing estimation for RV32IM processor
- Same simulation data results, different timing as now counting cycles
- Shows how long algorithm will take to execute

>Extends simulated time

• Was 12.89 secs now takes 16.59 secs



CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com. Copyright (c) 2005-2018 Imperas Software Ltd. Contains Imperas Proprietary Information. Licensed Software, All Rights Reserved. Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:27:16 2018

	(OR_OF) Target 'iss/cp (OR_PH) Program Header		ect file re	ad from 'ap	plication/t	est_c.RISCV	32.elf	•
Info Info Info Info	(OR_PH) Type (OR_PD) LOAD (OR_PD) LOAD (OR_PD) LOAD (OR_PT) Target 'iss/cp (OR_PH) Program Header	0ffset 0x00000000 0x000173c8 u0' has obj	0x000283c8	0x00010000 0x000283c8	0x000173c8 0x000009c0	0x000173c8 0x00000a24	R-E RM-	1000 1000
Info Info Info RES = Info	(OR_PH) Type (OR_PD) LOAD (CMD_CC) calling 'iss/ (CPUEST_CMD) iss/cpu0; = 84772366	Offset 0x00001000 cpu0/exTT/c cpucycles	pucycles' on: time st	retch enabl		MenSiz 0x0000000c	Flags R-E	Alig 1000
Info	CPU 'iss/cpu0' STATIST	'ICS						
Info	Type Nominal MIPS	: riscy (RV32IM)					
Info Info Info	Final program counte Simulated instructio Simulated MIPS	r : 0x100ac ns: 1,289,3						
Info Info								
Info								
			econds					
	User time	: 8.02 se	conds					
Info Info Info	System time	: 8.02 se : 0.23 se						
Info Info	System time Elapsed time		conds conds					

CpuManagerMulti finished: Thu Aug 23 11:27:25 2018

CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com. Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

Info (CPUEST_RSLT) Estimated execution time 16.59 seconds, clock cycles 1.659,204,454

Function Profile C Application

- Same C application
- IA+E simulation
- Sampled profiling with call stack analysis
- Shows proportion of time spent in each application function
 21.35% spent in processLine

Name (location)	Arcs in	Samples In	Arcs out	From/To this	Percentage (%)	
✓ Platform: iss						
➡ Processor: iss/cpu0						
		1659204277				
fread_r	598189652	596534872	1654780		3	5.95%
processLine	925852930	354236017	571616913		2	1.35%
⊅ qr4_c	150627639	150627639	0		S	.08%
≬ qr1_c	146083640	146083640	0		8	.8%
p qr2_c	137682652	137682652	0		8	.3%
≬ qr3_c	137222982	137222982	0		8	.27%
Dlibc_init_array	0	135154965	1524049412		8	.15%
_srefill_r	1654780	1924985	629795		0	.06%
_sread	629637	321116	308521		(.02%
⊅ _read_r	308521	308521	0			.02%
fseeko_r	2706	2126	580			.0%
<pre>> _vfprintf_r</pre>	1874	764	1110			.0%
_sfvwrite_r	848	752	96		0	.0%
p rewind	3267	561	2706		(.0%
<pre>b _close_r</pre>	357	357	0		0	.0%
▶ _malloc_r	323	297	26			.0%
_sseek	528	288	240		0	0.0%
⊅ _lseek_r	240	240	0		o	0.0%
_sfmoreglue	399	224	175		(.0%
fclose_r	734	204	530			.0%
h afficials a	317	199	40		· · · ·	- 04-

Flow to add new custom instructions

Characterize C Application

Instruction Accurate Simulation

• Trace / Debug

• Timing Simulation

• Function Timing / Profiling

Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to ModelAdd Timing

Add custom instructions to application

- Inline assembly using new instructions replacing C code
- 4 new instructions
- Cross Compile using standard tool
- Run on IA simulator
- Unimplemented instruction exception
 - As the instructions have not yet been added to the simulator model

```
unsigned int processLine(unsigned int input, unsigned int word) {
     unsigned int res = input;
     asm __volatile_("mv x10, %0" :: "r"(res));
     asm __volatile__("mv x11, %0" :: "r"(word));
     asm __volatile_(".word 0x00850508\n" ::: "x10");
                                                                             // OR1
     asm __volatile__(".word 0x00851508\n" ::: "x10");
                                                                             // OR2
     asm __volatile_(".word 0x00B52508\n" ::: "x10");
                                                                             // OR3
     asm __volatile_(".word 0x00853508\n" ::: "x10");
                                                                             // OR4
     asm __volatile_(".word 0x00850508\n" ::: "x10");
                                                                             // OR1
     asm __volatile_(".word 0x00851508\n" ::: "x10");
                                                                             // OR2
     asm __volatile_ (".word 0x00B52508\n" ::: "x10");
                                                                             // OR3
     asm __volatile_(".word 0x00853508\n" ::: "x10");
                                                                             // OR4
     asm __volatile_("mv %0,x10" : "=r"(res));
     return res;
         CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
int maiCopyright (c) 2005-2018 Imperas Software Ltd. Contains Imperas Proprietary Information.
          icensed Software, All Rights Reserved.
         Visit www.INPERAS.com for multicore debug, verification and analysis solutions.
     CON CpuManagerMulti started: Thu Aug 23 11:34:51 2018
     FIL
     if
         Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_oustom.RISCV32.elf
         Info (OR_PH) Program Headers:
                                          VirtAddr PhysAddr FileSiz
             (OR PH)
                    Туре
                                 Offset
                                                                      MenSiz
                                                                                Flags Align
          Info (OR PD) LOAD
                                 0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E 1000
             (OR_PD) LOAD
                                 0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 Rif-
                                                                                     1000
             (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf'
         Info
             (OR_PH) Program Headers;
                                           VirtAddr
                                                    PhysAddr
                                                             FileSiz
                                                                                Flags Align
              (OR PB) 1001
                                 0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E 1000
                              iss/cpu0' 0x00010248 00b5050b custow1: Illegal instruction - extension X (non-standard extensions present) absent or inactive
          Info
         Info
             CPU 'iss/cpu0' STATISTICS
          Info
                                 : riscv (RV321M)
               Tupe
               Nominal HIPS
         Info
                                  : 100
               Final program counter : 0x102e4
               Simulated instructions: 1,340
                                  : run too short for meaningful result
               Simulated MIP:
         Info
         Info SINULATION TIME STATISTICS
         InFo
               Simulated time
                                  : 0,00 seconds
                                  : 0.01 seconds
               liser time
          Info
               Sustem time
                                  : 0.00 seconds
          Info
               Elapsed time
                                  : 0.01 seconds
```

```
CpuManagerMulti finished: Thu Aug 23 11:34:51 2018
```

© Imperas Software Ltd.

Add custom instructions to model // Create the RISCV decode table

- Use standard Open Virtual Platforms (OVP) instruction modeling APIs to add new instructions (and optional state) as *new extension library*
 - Easy to extend decode table, add efficient behavioral JIT code
 - Optionally can call directly into user's provided C function of behavior
- Compile and link model extension library
- Simulate IA with ISS plus standard model extended with new library

Instruction count and simulated time have reduced (IA)

static vmidDecodeTableP createDecodeTable(void)

vmidDecodeTableP table = vmidNewDecodeTable(RISCV_INSTR_BITS, RISCV_EIT_LAST);

// Emit code implementing exchange instruction // R-Type instruction in custom-0 encoding space: // opcode [6:0] = 00 010 11 static void emitChaCha20(// funct3[14:12] = 0,1,2,3 (QR1-4) vmiProcessorP processor, // funct7[31:25] = 0000000 vmiosObjectP object, // rs1[19:15] Uns32 instruction, // rs2[24:20] Uns32 rotl // rd[11:7] // handle custom instruction // extract instruction fields DECODE ENTRY(0, CHACHA20QR1, "|0000000.....000.....0001011|"); Uns32 rd = RD(instruction); DECODE ENTRY(0, CHACHA200R2, "|00000000......001.....0001011|"); Uns32 rs1 = RS1(instruction); DECODE_ENTRY(0, CHACHA20QR3, "|0000000.....010....0001011|"); Uns32 rs2 = RS2(instruction); DECODE ENTRY(0, CHACHA200R4, "|0000000.....011....0001011|"); vmiReg reg_rs1 = vmimtGetExtReg(processor, &object->rs1); return table; vmiReg reg_rs2 = vmimtGetExtReg(processor, &object->rs2); vmiReg reg_tmp = vmimtGetExtTemp(processor, &object->tmp); vmimtGetR(processor, RISCV_REG_BITS, reg_rs1, object->riscvRegs[rs1]); vmimtGetR(processor, RISCV_REG_BITS, reg_rs2, object->riscvRegs[rs2]); puManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERGS.com opyright (c) 2005-2018 Imperas Software Ltd. Contains Imperas Proprietary Information. vmimtBinopRRR(32, vmi_XOR, reg_tmp, reg_rs1, reg_rs2, 0); icensed Software, All Rights Reserved. isit www.IMPERAS.com for multicore debug, verification and analysis solutions vmimtBinopRC(32, vmi ROL, reg tmp, rotl, 0); puManagerMulti started: Thu Aug 23 11:41:32 201 vmimtSetR(processor, RISCV REG BITS, object->riscvRegs[rd], reg tmp); \$IMPERRS_VLNV/riscv.ovpworld.or nfo (OP_LPR) Processor iss/cpu Info (OR OF) Target 'iss/cpu0' has object file read from application/test_custom.RISCV32.elf nfo (OR_PH) Program Headers: Info (OR_PH) Type Info (OR_PD) LOAD Offset VirtAddr PhysAddr FileSiz MewSiz Flags Aligr 0x00000000 0x00010000 0x00017270 0x00017270 R-E 1000 0v00017270 0v00028270 0v00028270 0v000009e0 0v00000x24 RM-1000 ofo (OR PD) LOGD nfo (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf' fo (OR_PH) Program Headers: Vietoddr Phusoddr FileSiz Offset (OR_PH) Type (OR_PD) LOAD MeaSiz 0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E 100 \$IMPERAS_VLNV/riscv.ovpworld.org/semihosting/riscv32Newlib/1.0/wodel OP_PEX) Extension iss/cpu0/riscv32Newlib fo (OP_PEX) Extension iss/cpu0/exInst instructionExtensionLib info CPU 'iss/cpu0' STATISTICS : riscy (RV321M) Туре Nominal MIPS Final program counter: 0x100ac Simulated instructions: 677,012,570 Simulated MIPS : 1301.9 FO SIMULATION TIME STATISTIC Simulated time : 6.77 seconds User time : 0,50 seconds System time 0.02 seconds : 0.53 seconds : 12.81x faster Elapsed time Real time ratio puManagerMulti finished: Thu Aug 23 11:41:33 2018 © Imperas Software Ltd. April 2019



Cycle Approximate simulation including custom instructions

- IA simulation + timing annotation + custom instructions
 - Includes timing estimation for RV32IM processor
 - Need to add timing estimation for new custom instructions
- Simulate using C code application with inline assembler of custom extensions
- IA simulator + timing tool + custom extension instruction library
- See estimated improvement in throughput of application on new processor
 - > Was 16.59 secs without custom instructions
 - > Now 9.21 secs with custom instructions

test_c.c	test_custom.c	📩 customChaCha20.c	🗂 riscv32.c ន		-	
	<pre>case IC_mdu_mul : { cycles = 5; break; }</pre>	<pre>// Specify cycles</pre>	for instruction	group		^
	<pre>case IC_mdu_div : { cycles = 16; break;</pre>	// Specify cycles	for instruction	group		
	<pre>} case IC_custom : { cycles = 2; break;</pre>	// chacha20qr1-ch // Specify cycles				
	<pre>} default: { VMI_ABORT("Inval break; }</pre>	lid instructionCLassE	value %d (%s)\n"	, iClass, instrClass	Name(iClass));	
}	đ.					
}	Info (CPUEST_CMD) iss/c RES = 84772366 Info	n time instruction info pu0: cpucycles on: time stret				4
	Info Nominal MIPS Info Final program co Info Simulated instru Info Simulated MIPS	: riscv (RV32IM) : 100 unter : 0x100ac				
	Info Info Info SIMULATION TIME ST Info Simulated time Info User time Info System time Info Elapsed time Info Real time ratio	ATISTICS : 9,21 seconds : 5,30 seconds : 0,20 seconds : 5,73 seconds : 1,61x faster				
	Info CpuManagerMulti finishe	d: Thu Aug 23 11:58:35 2018				
) v99999999 Open Virtual Plat or multicore debug, verificat				
	Lefe (CREECT DOLT) East	wated execution time 9.21 sec	ands alash suslas 92	000 000		

Trace custom instructions

- Simulator has many trace features built in
- See new custom instructions in trace disassembly
- Can select when/where to turn trace on/off
 - Very efficient tracing



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

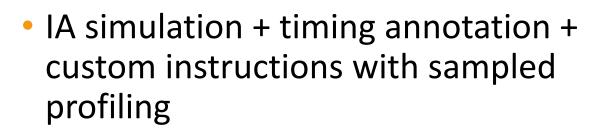
Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_custom,RISCV32,elf' Info (OR_PH) Program Headers: Info (OR_PH) Type Offset VirtAddr PhysAddr FileSiz MemSiz Flags Align Info (OR PD) LOAD 0x00000000 0x00010000 0x00010000 0x00017270 0x00017270 R-E 1000 0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RM-Info (OR_PD) LOAD 1000 Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/exception.RISCV32.elf' Info (OR_PH) Program Headers: Info (OR_PH) Type Offset VirtAddr PhysAddr FileSiz MenSiz Flags Align Info (OR_PD) LORD 0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E 1000 Info 1330: 'iss/cpu0', 0x0000000000010228(processLine+c): fca42e23 sw a0,-36(s0) nfo 1331: 'iss/cpu0', 0x00000000000000022c(processLine+10): fcb42c23 sw a1,-40(s0) a5.-36(s0) Info a5 a730c140 -> 84772366 Info 1333: 'iss/cpu0', 0x0000000000010234(processLine+18): fef42623 sw a5,-20(s0) nio 1334: 'iss/cpu0', 0x000000000010238(processLine+1c): fec42783 lw a5,-20(s0) les/cpu0', 0x00000000001023c(processLine+20); 00078513 mv a0,a5 Info 1336: 'iss/cpub', 0:000000000010240(processLine+24): fd842783 lw a5,-40(s0) Info a5 84772366 -> a750c1 Info 1337: 'iss/cpu0', 0x00000000 processLine+28): 00078593 m/ a1.a5 Info 1338: 'iss/cpu0', 0x000000000010248(pr esseLine+2c); chacha20gr1 a0,a0,a1 Info a0 84772366 -> e2262347 Info 1339: 'iss/cpu0', 0x00000000001024c(processLine+30): chacha20gr2 a0,a0,a1 Info a0 e2262347 -> 5e207451 Info 1340: 'iss/cpu0', 0x0000000000010250(processLine+34): chacha20gr3 a0.a0.a1 Info a0 6e207451 -> 10b511c9 Info 1341: 'iss/cpu0', 0x00000000010254(processLine+38): chacha20gr4 a0,a0,a1 Info a0 10b511c9 -> c2e844db Info 1342: 'iss/cpu0', 0x000000000010258(processLine+3c): chacha20gr1 a0,a0,a1 Info a0 c2e844db -> 859b65d8 Info 1343: 'iss/cpu0', 0x00000000001025c(processLine+40): chacha20gr2 a0.a0.a1 nfo a0 859b65d8 -> ba49822a Info a0 ba49822a -> 79436a1d Info 1345: 'iss/cpu0', 0x00000000000000254(processLine+48): chacha20gr4 a0,a0,a1 nfo a0 79436a1d -> 39d5aeef Info 1346: 'iss/cpu0', 0x000000000010268(processLine+4c): 00050793 m/ a5,a0 Info a5 a730c140 -> 39d5aeef Info 1347: 'iss/cpu0', 0x00000000000026c(processLine+50): fef42623 sw a5,-20(s0) Info 1348: 'iss/cpu0', 0x000000000010270(processLine+54): fec42783 lw a5,-20(s0) Info 1349; 'iss/cpu0', 0x0000000000010274(processLine+58); 00078513 m/ a0.a5 RES = 84772366 Info Info Info CPU 'iss/cpu0' STATISTICS Info Туре : riscy (RV32IM) Nominal MIPS Info : 100 Final program counter : 0x100ac Simulated instructions: 677,012,570 Simulated MIPS : 1209.0 Info Info Info

Debug custom instructions

- Imperas MPD is Eclipse based source code debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers

d • ₩ % Ø * • 0 • % • Ø * % • Ø × * ∞ # % ≫	10 .8	5 Mž - 14 -	0 200 <u>1</u> 201 - 01 - 0	• ¢ + ⇔ - Quick Acce	55	4
pebug 11 e		🕬 Variables 🕄	💁 Breakpoints 🚟 Reç	gisters 🛋 Modules	6	- 6
後●日■X13.00.00mm+月 計2.31 00	~			6 4 B 👩	8	
🕫 💷 Platform Launch [Imperas - Connect to running simulator]		Name	Туре	Value		
∀ im iss		60- input	unsigned int	2222400358		
🗢 🔐 cpu0 [RV32IM riscv]		to word	unsigned int	2804990272		
▽ P ID #1 [cpu0] RV32IM riscv (Suspended : Breakpoint)		🚧 res	unsigned int	0		
processLine() at test_custom.c:5 0x10230			 Restauro de servicio de la construistica de la construist Construistica de la construistica d			
main() at test_custom.c:32 0x102e4						
🔊 mpd		-				-
		G	NAME OF ASS			
test_custom.c 🛱 🗂 customChaCha20. 📄 riscv32.c 🛛 _start() at 0x1	**1		utline 🏧 Programmen	s View 🔛 Disassembly 🛙	=	- 6
// Custom instruction test for Chacha20			Enter	location here 🗸 👔 🔞	1	
<pre>#include <stdio.h></stdio.h></pre>		000	1023c: 00078513	mv a0,a5	9	-
		- 000	1052C: 000192T2	CB, UB VIII		1.0
		000	10240 · fd842783			- 11
unsigned int processLine(unsigned int input, unsigned int word){			10240: fd842783 10244: 00078593	lw a5,-40(s0) mv a1,a5		
unsigned int res = input;		000		lw a5,-40(s0) mv a1,a5		
		000	10244: 00078593 10248: chacha20qr2 chacha20qr2	lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1		
<pre>unsigned int res = input; asm _volatite_(mv x10, 50 ""(res)); asm _volatile_("mv x11, 50 "r"(word)); asm _volatile_(".word 0x00B5050B\n" ::: "x10"); // QR1</pre>		000 000 000 000	10244: 00078593 10248: chacha20qr1 1024c: chacha20qr2 10250: chacha20qr3	lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1		-
<pre>unsigned int res = input; asm _volatile_(== vieword 0x00850508\n" == ""(cos)); asm _volatile_(== vieword 0x00850508\n" ::: "x10"); // 0R1 asm _volatile_(== vieword 0x00851508\n" ::: "x10"); // 0R2</pre>		996 996 996 996 996	10244 : 00078593 10248 : chacha20gr 1024c : chacha20gr 10250 : chacha20gr 10255 : chacha20gr 10256 : chacha20gr	<pre>lw a5,-40(s0) mv a1,a5 l a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1</pre>		
<pre>unsigned int res = input; asm _volatile_('mv x10, 50" :: "r"(word)); asm _volatile_('mv x11, 50" :: "r"(word)); asm _volatile_('.word 0x0085508\n" ::: "x10"); // 0R1 asm _volatile_('.word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_('.word 0x00852508\n" ::: "x10"); // 0R3</pre>		000 000 000 000 000	10244: 00078593 10248: chacha20qr? 10246: chacha20qr? 10250: chacha20qr? 10254: chacha20qr? 10258: chacha20qr?	<pre>lw a5,-40(s0) mv a1,a5 l a0,a0,a1 a0,a0,a1 a0,a0,a1 a0,a0,a1 a0,a0,a1 a0,a0,a1 a0,a0,a1 l a0,a0,a1</pre>		
<pre>unsigned int res = input; asm _volatile_(mv x10, 90" "r"(res]); asm _volatile_("mv x11, 90" :: "r"(word)); asm _volatile_(".word 0x00850508\n" ::: "x10"); // 0R1 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R3 asm _volatile_(".word 0x00852508\n" ::: "x10"); // 0R3 asm _volatile_(".word 0x00853508\n" ::: "x10"); // 0R4</pre>			10244: 00078593 10248: chacha20qr2 10240: chacha20qr2 10250: chacha20qr2 10254: chacha20qr2 10255: chacha20qr2 10256: chacha20qr2 10258: chacha20qr2 10258: chacha20qr2 10256: chacha20qr2	lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 1 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1		
<pre>unsigned int res = input; asm _volatile_("mv x10, %0" :: "r"(word)); asm _volatile_("mv x11, %0" :: "r"(word)); asm _volatile_(".word 0x00859508\n" ::: "x10"); // 0R1 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_(".word 0x00853508\n" ::: "x10"); // 0R4 asm _volatile_(".word 0x00853508\n" ::: "x10"); // 0R4</pre>			10244: 00078593 10248: chacha20qr? 1024c: chacha20qr? 10259: chacha20qr? 10254: chacha20qr? 10258: chacha20qr? 10258: chacha20qr? 10256: chacha20qr? 10256: chacha20qr? 10260: chacha20qr?	<pre>lw a5,-40(s0) mv a1,a5 l a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 </pre>		
<pre>unsigned int res = input; asm _volatile_(== v > 10, vol == = = vol = vol</pre>			10244: 00078593 10248: chacha20qr2 10240: chacha20qr2 10250: chacha20qr2 10254: chacha20qr2 10255: chacha20qr2 10256: chacha20qr2 10258: chacha20qr2 10258: chacha20qr2 10256: chacha20qr2	<pre>lw a5,-40(s0) mv a1,a5 l a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 3 a0,a0,a1 3 a0,a0,a1</pre>		
<pre>unsigned int res = input; asm _volatile_(mv x10, 50</pre>			10244: 00078593 10248: chacha20qr2 10246: chacha20qr2 10250: chacha20qr2 10254: chacha20qr2 10255: chacha20qr2 10256: chacha20qr2 10256: chacha20qr2 10260: chacha20qr2 10260: chacha20qr2	<pre>lw a5,-40(s0) mv a1,a5 l a0,a0,a1 a0,a0,a0 a0,a0 a0,a0 a0,a0 a0,a0 a0,a0 a0,a0</pre>		
<pre>unsigned int res = input; asm _volatile_(mv x10, %0 ::: "r"(word)); asm _volatile_("w x11, %0 ::: "r"(word)); asm _volatile_(".word 0x00850508\n" ::: "x10"); // 0R1 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_(".word 0x00853508\n" ::: "x10"); // 0R4 asm _volatile_(".word 0x00853508\n" ::: "x10"); // 0R4 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R1 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R1 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R1 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R3 asm _volatile_(".word 0x00851508\n" ::: "x10"); // 0R3</pre>	Π		10244: 00078593 10244: chacha20qr2 chacha20qr2 chacha20qr2 chacha20qr2 10254: chacha20qr2 10254: chacha20qr2 10256: chacha20qr2 10260: chacha20qr2 10260: chacha20qr2 10264: chacha20qr2 10268: 00050793 (lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 4 a0,a0,a1 mv a5,a0	emor =	
<pre>unsigned int res = input; dsm _volatile (mv x10, 50</pre>			10244: 00078593 10244: chacha20qr2 chacha20qr2 chacha20qr2 chacha20qr2 10254: chacha20qr2 10254: chacha20qr2 10256: chacha20qr2 10260: chacha20qr2 10260: chacha20qr2 10264: chacha20qr2 10268: 00050793 (lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 5 mv a5,a0 mv		
<pre>unsigned int res = input; dsm _volatile_ ("mv x10, 50" ::: "r"(vord)); asm _volatile_ ("mv x11, 50" ::: "r(vord)); asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x00853508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x00853508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x00853508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00853508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00853508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x00852508\n" ::: "x10"); // 0R3</pre>			10244: 00078593 10244: chacha20qr2 chacha20qr2 chacha20qr2 chacha20qr2 10254: chacha20qr2 10254: chacha20qr2 10256: chacha20qr2 10260: chacha20qr2 10260: chacha20qr2 10264: chacha20qr2 10268: 00050793 (lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 5 mv a5,a0 mv		
<pre>unsigned int res = input; dsm _volatile_ ("mv x10, 50" ::: "r(word)); asm _volatile_ ("mv x11, 50" ::: "r(word)); asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x00852508\n" ::: "x10"); // 0R3 climited (".word 0x00852508\n" ::: "x10"); // 0R3 climited</pre>	(● 000 ● 0000 ● 0000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000	10244: 00078593 10244: chacha20qr2 chacha20qr2 chacha20qr2 chacha20qr2 10254: chacha20qr2 10254: chacha20qr2 10256: chacha20qr2 10260: chacha20qr2 10260: chacha20qr2 10264: chacha20qr2 10268: 00050793 (lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 5 mv a5,a0 mv		
<pre>unsigned int res = input; asm _volatile_ (mv x10, %0 ::: "r(word); asm _volatile_ ("wv x11, %0 ::: "r(word); asm _volatile_ (".word 0x00850508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00851508\n" ::: "x10"); // 0R3 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R3 c B Debugger Console 13 asform Launch [Imperas - Connect to running simulator] mpd.exe (7.5) igned int), 1, fp)) { debug (cpu0) > 32 c res = processLine(res, word);</pre>	•	● 000 ● 0000 ● 0000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000	10244 : 00078593 10244 : 00078593 chacha20qri chacha20qri 10254 : chacha20qri 10258 : chacha20qri 10256 : chacha20qri 10260 : chacha20qri 10268 : ch	lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 5 mv a5,a0 mv		
<pre>unsigned int res = input; dsm _volatile ('mv x10, '00' :: ''r'(word)); asm _volatile ('mv x10, '00' :: ''x10'); // QR1 asm _volatile (''.word 0x00851508\n" ::: ''x10'); // QR2 asm _volatile (''.word 0x00851508\n" ::: ''x10'); // QR3 asm _volatile (''.word 0x00850508\n" ::: ''x10''); // QR4 asm _volatile (''.word 0x00850508\n" ::: ''x10''); // QR1 asm _volatile (''.word 0x00852508\n" ::: ''x10''); // QR2 asm _volatile (''.word 0x00852508\n" ::: ''x10''); // QR3 c</pre>	•	● 000 ● 0000 ● 0000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000	10244 : 00078593 10244 : 00078593 chacha20qri chacha20qri 10254 : chacha20qri 10258 : chacha20qri 10256 : chacha20qri 10260 : chacha20qri 10268 : ch	lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 5 mv a5,a0 mv		
<pre>unsigned int res = input; asm _volatile_ (mv x10, %0 ::: "r(word); asm _volatile_ (".word 0x00850508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x00851508\n" ::: "x10"); // 0R2 asm _volatile_ (".word 0x0085508\n" ::: "x10"); // 0R3 asm _volatile_ (".word 0x0085508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x00850508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x0085508\n" ::: "x10"); // 0R4 asm _volatile_ (".word 0x0085508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x0085508\n" ::: "x10"); // 0R1 asm _volatile_ (".word 0x0085508\n" ::: "x10"); // 0R3 asm _volatile_ (".word 0x00855508\n" ::: "x10"); // 0R3 asm _volatile_ (".w</pre>	•	● 000 ● 0000 ● 0000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000 ● 000	10244 : 00078593 10244 : 00078593 chacha20qri chacha20qri 10254 : chacha20qri 10258 : chacha20qri 10256 : chacha20qri 10260 : chacha20qri 10268 : ch	lw a5,-40(s0) mv a1,a5 1 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 2 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 3 a0,a0,a1 4 a0,a0,a1 5 mv a5,a0 mv		

Function Profile custom instructions application



Shows where slowest function is

Now much faster...

Shows benefits of using custom instructions

>processLine was 21.35% now 16.3%



Name (location)	Arcs in	Samples In	Arcs out	From/To this	Percentage (%)	Y
♥ Platform: iss		1	1			
♥ Process: 0_None		921006649				
	635365939	633628269	1737670		68.8%	
Dibc_init_array	0	150138664	770867985		16.3%	
P processLine	135494635	135494635	0		14.71%	
bsrefill_r	1737670	1066083	671587		0.12%	
♪ _read_r	340125	340125	0		0.04%	
_sread	671429	331304	340125		0.04%	
fseeko_r	3849	3269	580		0.0%	
_sfvwrite_r	784	688	96		0.0%	
sflush_r	599	559	40		0.0%	
<pre>> _vfprintf_r</pre>	1492	446	1046	1	0.0%	
P rewind	4153	304	3849		0.0%	
_malloc_r	323	297	26		0.0%	
_sseek	528	288	240		0.0%	
> _lseek_r	240	240	0		0.0%	
_sfmoreglue	399	224	175		0.0%	
_fclose_r	811	204	607		0.0%	
<pre>>sinit.part.1</pre>	146	146	0		0.0%	
<pre>> _fwalk_reent</pre>	790	106	684		0.0%	
▷ _sfp	641	96	545		0.0%	
smakebuf r	316	78	238		0.0%	

Basic Block (BB) Profile custom instructions application

- IA simulation + timing annotation + custom instructions with detailed BB profiling
- Shows where expensive instruction sequences are
- Allows understanding of instruction performance
 - Useful for Compiler teams
 - Useful for Hardware teams

bbPr	ofile_c.txt 🛛	
1	BB@0x000102ac:0x000102d0 executed 5758080 times	: ^
2		sp,-16
3	0x000102b0(gr1 c+0x4): 00812623 sw s0,	12 (sp)
4	0x000102b4(qr1 c+0x8): 01010413 addi s0,	sp,16
5	0x000102b8(qr1_c+0x12): 00b545b3 xor al	.,a0,a1
6	0x000102bc(qr1_c+0x16): 01059513 slli a0	,a1,0x10
7	0x000102c0(qr1_c+0x20): 0105d593 srli a1	,a1,0x10
8	0x000102c4(qr1_c+0x24): 00a5e533 or a0	,a1,a0
9	0x000102c8(qr1_c+0x28): 00c12403 lw s0),12(sp)
10		,sp,16
11	0x000102d0(qr1_c+0x36): 00008067 ret	
12		
13	BB@0x000102d4:0x000102f8 executed 5758080 times	
14	0x000102d4(qr2_c+0x0): ff010113 addi sp,	
15		12(sp)
16		sp,16
17	0x000102e0(qr2_c+0x12): 00b545b3 xor a1	
18	0x000102e4(qr2_c+0x16): 00c59513 slli a0	
19	0x000102e8(qr2_c+0x20): 0145d593 srli a1	
20		,a1,a0
21	0x000102f0(qr2_c+0x28): 00c12403 lw s0	
22		,sp,16
23	0x000102f8(qr2_c+0x36): 00008067 ret	
24	BB@0x000102fc:0x00010320 executed 5758080 times	
26	0x000102fc(gr3 c+0x0): ff010113 addi sp,	535320
27		12 (sp)
28	0x00010304(qr3_c+0x4): 00012023 sw so, 0x00010304(qr3_c+0x8): 01010413 addi so,	
29	0x00010308(gr3 c+0x12): 00b545b3 xor al	
30	0x0001030c(gr3 c+0x16): 00859513 slli a0	
31	0x00010310(gr3 c+0x20): 0185d593 srli a1	
20		× 1 -0 ×
length : 4	4,975 Ln : 1 Col : 1 Sel : 0 0 Windows (CR LF) U	JTF-8 INS

Flow to add new custom instructions

Characterize C Application

- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling

Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to Model

Add Timing

Imperas

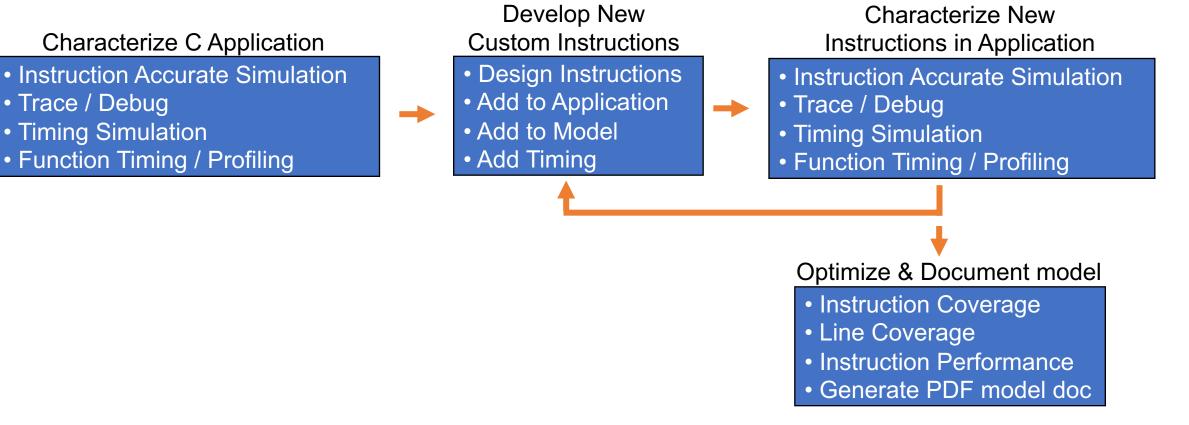
Characterize New Instructions in Application

Instruction Accurate Simulation

• Trace / Debug

- Timing Simulation
- Function Timing / Profiling

Flow to add new custom instructions



Further tools for model developers



- Model source line coverage
 - To see how completely the tests exercise the model

Name	Total Lines	Instrumente	Executed Lir	Coverage %
7 Summary	16,900	4,411	2,834	64.25%
customChaCha20.c	374	87	42	48.28%
riscvBus.c	175	46	1	2.17%
riscvCSR.c	2,573	758	549	72.43%
riscvConfigList.c	70	2	0	0.0%
riscvDebug.c	527	167	72	43.11%
riscvDecode.c	1,599	360	164	45.56%
riscvDisassemble.c	514	185	185	100.0%
riscvDoc.c	725	143	30	20.98%
riscvExceptions.c	1,408	420	317	75.48%
riscvInfo.c	83	3	0	0.0%
riscvMain.c	383	147	25	17.01%
riscvMorph.c	2,887	1,032	776	75.19%
riscvParameters.c	589	145	9	6.21%
riscvRegisterTypes.h	115	10	6	60.0%
riscvSemiHost.c	44	6	3	50.0%
riscvStructure.h	204	4	2	50.0%
riscvUtils.c	493	122	45	36.89%
riscvVM.c	2,695	770	608	78.96%
vmiMt.h	1,442	4	0	0.0%

test_c.c 📑 test_custom.c	n customChaCha20.c 🛱	Tiscv32.c	<pre>_start() at 0x100d0</pre>	
// Emit code implementing	exchange instruction			-
<pre>static void emitChaCha20{ vmiProcessorP processo vmiosObjectP object, Uns32 instruct Uns32 rotl) {</pre>				
<pre>// extract instruction Uns32 rd = RD(instruc Uns32 rs1 = RS1(instru Uns32 rs2 = RS2(instru Uns32 rs2 = RS2(instru</pre>	tion); ction);			
<pre>vmiReg reg_rs1 = vmimt vmiReg reg_rs2 = vmimt vmiReg reg_tmp = vmimt</pre>	GetExtTemp line executed			
vmimtGetR(processor, R	ISCV_REG_BIT5, reg_rs1, o ISCV_REG_BIT5, reg_rs2, o XOR, reg_tmp, reg_rs1, re OL, reg_tmp, rotl, 0);	bject->riscvR		
<pre>vmimtSetR(processor, R }</pre>	ISCV_REG_BITS, object->ri	scvRegs[rd],	reg_tmp);	
11	5 a. 1 f			 10

Further tools for model developers (2)

- Custom instruction coverage
 - To see that there are tests for new instructions

• Op	addi : addi : begz : bnez : add : thret :	Pct 19.87% 16.78% 13.05% 8.70% 4.97% 4.35% 3.73%	113575358 88327451 58868837	677012570)		
	addi : addi : beqz : bnez : add : slli : ret :	19.87% 16.78% 13.05% 8.70% 4.97% 4.35% 3.73%	134497608 113575358 88327451 58868837	677012570)		
	addi andi beqz addi tret	19.87% 16.78% 13.05% 8.70% 4.97% 4.35% 3.75%	134497608 113575358 88327451 58868837			
	lw : sw : addi : andi : beqz : bnez : add : slli : ret :	16.78% 13.05% 8.70% 4.97% 4.35% 3.75%	113575358 88327451 58868837			
	sw : addi : andi : beqz : bnez : add : slli : ret :	13.052 8.702 4.972 4.352 3.732	88327451 58868837			
	andi : begz : bnez : add : slli : ret :	4.97% 4.35% 3.73%				
	beqz : bnez : add : slli : ret :	4.35% 3.73%				
	bnez : add : slli : ret :	3.732				
	add : slli : ret :					
	slli : ret :	3.73Z :				
		3,102	: 21020892			
	141 *	2.49%				
		2.49%				
		2,48%				
		1.24%				
	bltz :	1.24%	: 8421470			
		1.24%				
		1.24%				
		1.242				
		1.242				
	lh :	0,632	: 4243601			
	suipe :	0.62%	4210858			
		0.62%				
		0.62%				
		0,00Z				
		0.007				
		0.007				
		0.00T				
		0,007				
	lui :	0,002	: 67		~	
		0,002			I	
		0.007				
		0.007				
		0,007				
	jr :	0,00Z	: 25			
		0,007				
		0.007				
		0.007				
		0,00Z	7			
	bgez :	0,002	: 6			
	=11 *	0.007	4			

- Custom instruction profile
 - See how long the simulator takes to execute each instruction
 - Use to focus speed up simulation of instructions
 - Enables improvement of speed of simulation runs

SW	0.27s	88,327,451	3.1ps/instruction (iss/cpu0
16	0,20s	113,575,358	1.8ps/instruction (iss/cpu0
WV .	0.16s	134,497,608	1.2ps/instruction (iss/cpu0
ret	0.07s	16,843,028	4.2ps/instruction (iss/cpu0
jal	0.06s	16,843,026	3.6ps/instruction (iss/cpu0
add1	0.05s	58,868,837	0.8ps/instruction (iss/cpu0
andi	0.05s	33,620,233	1.5ps/instruction (iss/cpu0
add	0.05s	25,231,546	2.0ps/instruction (iss/cpu0
auipc	0.04s	4,210,858	9.5ps/instruction (iss/cpu0
beqz	0.030	29,426,103	1.0ps/instruction (iss/cpu0
bnez	0.03s	25,248,055	1.2ps/instruction (iss/cpu0
chacha20gr3	0.03s	8,388,608	3.6ps/instruction (iss/cpu0
(JIT translation)	0.03s	2,500	12.0ns/instruction
bltu	0.02s	16,793,686	1.2ps/instruction (iss/cpu0
chacha20gr1	0.02s	8,388,608	2.4ps/instruction (iss/cpu0
lb	0.02s	4,243,601	4.7ps/instruction (iss/cpu0
XOF	0,02s	4,210,707	4.7ps/instruction (iss/cpu0
s11i	0.01s	21,020,892	0.5ps/instruction (iss/cpu0
srli	0.018	16,810,160	0.6ps/instruction (iss/cpu0
bltz	0.01s	8,421,470	1.2ps/instruction (iss/cpu0
1	0.01s	8,388,742	1.2ps/instruction (iss/cpu0
not	0.01s	4,194,307	2.4ps/instruction (iss/cpu0
(unallocated)	0.01s		
beeu	0,00s	8,421,494	0.0ps/instruction (iss/cpu0
chacha20gr2	0,00s	8,388,608	0.0ps/instruction (iss/cpu0
chacha20gr4	0,00s	8,388,608	0.0ps/instruction (iss/cpu0
sub	0,00s	4,210,835	0.0ps/instruction (iss/cpu0
beg	0,00s	16,546	0.0ps/instruction (iss/cpu0
jalr	0.00s	16,430	0.0ps/instruction (iss/cpu0
blez	0,00s	16.407	0.0ps/instruction (iss/cpu0
sb	0.00s	206	0.0ps/instruction (iss/cpu0
Ibu	0,008	97	0.0ps/instruction (iss/cpu0
sh	0,00s	82	0.0ps/instruction (iss/cpu0
lui	0,00s	67	0.0ps/instruction (iss/cpu0
or	0,00s	50	0.0ps/instruction (iss/cpu0
and	0,00s	49	0.0ps/instruction (iss/cpu0
lbu	0,00s	47	0.0ps/instruction (iss/cpu0
bne	0,00s	34	0.0ps/instruction (iss/cpu0
001	0,00s	29	0.0ps/instruction (iss/cpu0
ir	0,00s	25	0.0ps/instruction (iss/cpu0
TOTAL	1.21s	677,012,570	and a second sec

Document custom instructions

Imperas

- Imperas tools automatically generate a processor model document PDF
- Includes all base model registers and any new registers
- Provides detailed documentation of new custom instructions

Chapter 2

Instruction Extensions

RISCV processors may add various custom extensions to the basic RISC-V architecture. This processor has been extended, using an extension library, to add several instruction using the Custom0 opcode.

2.1 Custom Instructions

This model includes four Chacha20 acceleration instructions (one for each rotate distance) are added to encode the XOR and ROTATE parts of the quarter rounds.

2.1.1 chacha20qr1

31	25	24	20	19	15	14	12	11	7	6	
0000	0000000 Rs2		2	Rs1		000 (QR1)		Rd		Custom0 0001011	

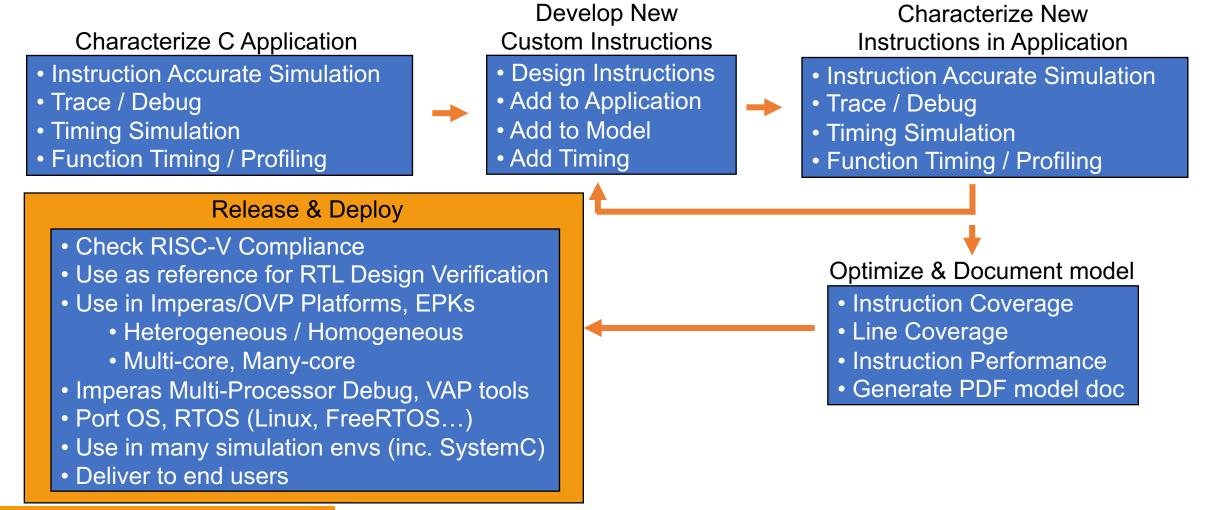
2.1.2 chacha20qr2

31	25	- 24	20	19	15	14	12	11	7	6	0
	0000000 Rs2		2	Rs1		001 (QR2)		Rd		Custom0 0001011	

2.1.3 chacha20qr3

31	- 25	24	20	19	15	14	12	11	- T.	6	0
000000		Rs2		Rs1		010 (QR3)		Rd		Custom0 0001011	

Flow to add new custom instructions



Summary



- Compliance is a new challenge for open ISAs
- Additional verification tasks related to the open ISA processor
- Custom instructions are a key value proposition of RISC-V
 - Adding custom instructions requires solving the key challenge of how to optimize those instructions
- Instruction accurate (IA) simulation environment using IA models can be extended to enable custom instruction analysis and optimization
- Flow for optimizing custom instructions in RISC-V processors is being used in real designs

Thank You



- Visit <u>www.imperas.com/riscv</u> and <u>www.OVPworld.org/riscv</u> for more information
- RISC-V Foundation Compliance Suite, includes riscvOVPsim available:
 - https://github.com/riscv/riscv-compliance

Larry Lapides VP Sales, Imperas Software larryl@imperas.com