# mperas

### Impact of RISC-V Adaptability on SoC Verification Methods

S. Davidmann, L. Moore, L. Lapides Imperas Software Ltd.



25-Feb-20

© 2020 Imperas Software Ltd.

### Agenda



- RISC-V verification issues
- Compliance is not verification
- Reference models and custom instructions
- Processor IP verification
- SoC level verification
- Summary





#### RISC-V verification issues

- Compliance is not verification
- Reference models and custom instructions
- Processor IP verification
- SoC level verification
- Summary

# RISC-V Presents New Challenges



- RISC-V is a new ISA an open standard ISA
  - Managed by the non-profit RISC-V Foundation (riscv.org)
  - This means any designer can build a processor implementation
    - (Feb 2020 there are almost 100 RTL designs including open source and proprietary)
- Traditionally, processor IP ...
  - comes from, and is maintained by, the ISA owner
  - is single sourced
  - comes fully verified and compliant to that specific ISA
- All users need to do is to verify using integration tests
- There is no "standard" approach and there are few available tools for processor verification
- The RISC-V industry / eco-system needs to adapt best practices for SoC verification to processor verification

# To be more specific about the RISC-V DV Problem

# mperas

- Arm processor IP
  - ~ 10<sup>15</sup> verification cycles per processor
  - Verification of interface between NoC and processor
  - 1,000s of SoC designs successfully produced
- Similar stories for ARC, MIPS, Tensilica, ...

#### RISC-V IP

- How well verified is an individual processor (from processor IP vendor, open source, self-built)?
- How to verify processor subsystems, especially for AI/ML architectures?
- How well verified is interface between NoC and processor?
- How to deal with custom instructions?

# Agenda



- RISC-V verification issues
- Compliance is not verification
- Reference models and custom instructions
- Processor IP verification
- SoC level verification
- Summary

# **Compliance Testing**



- The device works within the envelope of the agreed specifications
  - Have you read and understood the specification
- Testing of the instructions should
  - Attempt to use all registers as source and destination (not combinations)
  - Attempt to operate on all bits which compose the immediate values (1 / 0)
  - Capture a signature in memory region indicating the test result
    - Based upon a particular hardware configuration
  - Compare the signature against a known good reference
    - Static (pre defined signature extraction)
    - Dynamic (runtime generation from YAML configured reference)

# Compliance Testing (2)



- Testing of the instructions should NOT
  - Attempt to stress all possible aspects of functional verification, eg
    - All possible combinations of instruction parameters (2-in, 1-out = 32,768)
    - All possible data values
  - Attempt to expose possible micro-architectural aspects
  - Attempt to exercise behaviour which generates an exception
    - Illegal instructions (unsupported extensions)
      - (\*) Do not test for missing M instructions in context of RV32I
    - Illegal conditions (misaligned fetch, load, store)

# **Compliance Testing (3)**



- Test Qualification
  - Functional Coverage analysis
  - Mutation Fault Simulation Testing analysis (Imperas work in progress)
    - Provides Decode Coverage
      - Sees if observe changes on all bits of legal decodes

# Compliance Testing (4)



- Test Qualification
  - Functional Coverage analysis
  - Mutation Fault Simulation Testing analysis (Imperas work in progress)
    - Provides Decode Coverage
      - Sees if observe changes on all bits of legal decodes
    - Verified against RV32I test suite
      - 48 hand coded directed tests (average 150 instructions each)
      - https://github.com/riscv/riscv-compliance/tree/master/riscv-test-suite/rv32i/src
    - Decode Coverage data from the Imperas tool
      - ran 478,390 simulations in 308 secs

# **Compliance Testing (5)**



- Compliance RV32I Base Instruction Testing
  - November/12/2019 48 tests
- Compliance RV64V Vector instruction Testing (Imperas work in progress)
  - February/2020 ~6,000 tests
- RISCV-V compliance suites are still a work in progress

## Agenda



- RISC-V verification issues
- Compliance is not verification
- Reference models and custom instructions
- Processor IP verification
- SoC level verification
- Summary

# Key Issue – Reference Comparison

- One thing compliance, directed, random have in common...
  - Is a need for a reference implementation to compare with
- So why do I need a reference as part of my verification ?
  - Comparison for the observed behavior
  - Covering all possible aspects of the ISA envelope
- And it needs to represent your exact design and architecture:
  - XLEN
  - Vectors: VLEN, SLEN, ELEN, (version: 0.7.1, 0.8, 0.9 Draft, ...)
  - Bit Manipulation (version: 0.9, 0.91, 0.92, ...)
  - Custom Extensions
  - M+U (No S)
  - Hardware LSU Misalignment Support (no exception)
  - CSR: MTVEC ReadOnly

# **RISC-V Reference Choices**



- RISC-V is highly configurable
- So it can get a little .... complicated
- 60 Questions ?



### riscvOVPsim as the Reference Model for Compliance Testing

Debugger

CPU model variant selection and

configuration

riscvOVPsim

(cpu+memory)

Semihosted File I/O

Imperas riscvOVPsim Compliance Simulator

Imperas

GUI



- Industrial quality, free ISS / reference model for compliance testing
  - GitHub.com/riscv/riscv-compliance
  - GitHub.com/riscv/riscv-bitmanip
- Model is built using Open Virtual Platforms (OVP) APIs
- Implements full RISC-V envelope
  - Configurable for all features and version
- Includes full open source Apache 2.0 model
- Kept up to date for specification changes
- Works 'out of the box' with full tracing, debug, and many other options
- Video: <u>http://www.imperas.com/riscvovpsim-a-complete-risc-v-iss-for-bare-metal-software-development-and-specification-compliance</u>
- Has some limitations which make it not appropriate as a reference model for RTL design verification (DV)

Application

<cross>.el



# Additional Capabilities Needed for a DV Reference Model

- Support for multi-hart processors
- Support for custom instructions
- Support for injection of external / asynchronous events
- Support for step-and-compare DV flow

# OVP RISC-V Model and Imperas Simulator as Reference



http://www.imperas.com/riscv

- Support for multi-hart processors
- Support for custom instructions
- Support for injection of external / asynchronous events
- Support for step-and-compare DV flow
- Used as golden reference in RISC-V Foundations' Compliance Suite and Bit Manipulation group
- In use as reference with customers for RTL DV

# Flow to add new custom instructions



#### Page 18

#### Page 19

#### © 2020 Imperas Software Ltd.

#### 25-Feb-20

## Agenda

- RISC-V verification issues
- Compliance is not verification
- Reference models and custom instructions

#### Processor IP verification

- Step and compare methodology
- Directed tests
- Instruction stream generation
- Test generation and execution
- SoC level verification
- Summary



### DV Methodology: Step and Compare vs Trace/Signature Compare



- Short answer: bottom line is DV resources used
- With trace / signature comparison, failures are not known until after the simulation has completed; this can be a long time for a complex test, and therefore could waste simulation resources
- Step and compare enables failures to be flagged and the simulation stopped when the failure occurs

# Step and Compare Requires Encapsulation of the Reference in SystemVerilog

SystemVerilog Interface



#### SystemVerilog module

- The OVP model is a binary shared object of a RISC-V CPU model
- Encapsulated into a SystemVerilog module, using SystemVerilog DPI
  - Interfaces being: reset, step, address bus, data bus, interrupts, etc.,...
- Instanced in SystemVerilog design or testbench like any module



- Testbench loads .elf program into both memories, resets CPUs (RTL and OVP model)
- Steps CPUs (DUT and reference), extracting data, and comparing
  - There is no stored log file test log data is dynamic and requires two targets to be run and compared

# **Directed Testing**

- Test Encoded Self Checking
  - Tests are written with expected behaviour encoded
  - Tests can introspect the state and (self) diagnose faults
- Reference Comparison Checking
  - Tests are written without predicting the result
  - A reference is consulted for the correct value

// Device Under Test
int a = 4; int b = 5;
int c = a + b;
// c == ?

// Reference									
int $Ra = 4$ ; int $Rb = 5$ ;									
int Rc = Ra + Rb;									
// Rc == 9									

assert(c == Rc) // external (@runtime or post-processed)



### Key Issue for Directed Testing: Coverage

# mperas

Name		Class Type	Coverage	Goal	% of Goal	Status Inc	luded Merge_i	nstances	Get_ins	t_coverage	1	
J /riscv_instr_pkg/	riscv_instr_cover_group		76:24%							-		
TYPE add_cg		riscv_instr_co	100.00%	100	100.00%	$\sim$		auto(1	)			
+ TYPE sub_cg		riscv_instr_co	100.00%	100	100.00%	$\sim$		auto(1	)			
TYPE addi_cg		riscy_instr_co	99.00%	100	80.00%			auto(1				
• TYPE aulpc c	g	riscy instr co	80.00%	100	80.00%			auto(1				
🕣 🗾 TYPE sra_cg	20	riscv_instr_co	100.00%	100	100.00%	· · · · · · · · · · · · · · · · · · ·		auto(1	) R			
🕁 📕 TYPE sll_cg		riscv_instr_co	100.00%	100	100.00%			auto(1	) ि			
TYPE sr1_cg		riscy_instr_co	100.00%	100	0.00%	· · · ·		auto(1	.)			
TYPE sill co		riscy instr co	100.00%	100	100.00%			auto(1	3			
🕢 🔟 TYPE srli_cg		riscv_instr_co	100.00%	100	100.00%		•	auto(1	.)			
TYPE xor_cg		deni laste es	100 000	100	100.004		,	auto/1				
TYPE or_cg	Covergioups :-				Ye			194			1997 - 19	
TYPE xori co	▼ Name					Class Type	Coverage	e Go	al	% of Goal	Status	Include
• TYPE ori_cg	🗐 🗾 /riscv_instr_pkg/riscv_in	str_cover_grou	р				76	2496				
TYPE andi_cg	😑 🗾 TYPE add_cg					riscv_instr_co	D 100	.00%	100	100.00%		
→	🕀 🗾 CVP add cg::cp	rs1				riscv instr co	o 100	.00%	100	100.00%	1	
🕣 🗾 TYPE siti_cg	🕀 🗾 CVP add cg::cp	rs2				riscv instr co	0 100	.0096	100	100.00%	6	
TYPE slou_cg	CVP add co::cp	rd				riscy instr co	D 100	.00%	100	100.00%	1	
TYPE bed_cg	CVP add cg::cp	rs1 sign				riscy instr co	D 100	.00%	100	100.00%		
TYPE blt_cg	+ CVP add cg::cp	rs2 sign				riscv instr co	D 100	.00%	100	100.00%		
TYPE bge_cg     TYPE blau ca	+ CVP add cg::cp	rd sign				riscy instr co	D 100	.00%	100	100.00%		
TYPE baeu ca	CVP add co::cp	opr harzard				riscy instr ce	0 100	.00%	100	100.00%	-	1
• TYPE Ib_cg	CROSS add com	cp sign cross				riscy instr co	D., 100	.0096	100	100.00%		
TYPE In_cg	TYPE sub ca					riscy instr co	100	0096	100	100.00%		
THE IW_Cg	TYPE addi co					riscy instr cr	n 90	5596	100	99 5546	-	
						riscy instr cr	n R(	0096	100	80.00%	-	
	TYPE autoc co					riscy instr of	n R(	00%	100	80.00%		
	TYPE sra cn					riscy instr cr	n 100	0.096	100	100.00%		
	TYPE SIL CO					riscy instr co	100	0004	100	100.00%	1	
	TYPE SIL OD					rieov_inst_o	0 100	0.0090	100	100.00%		
						ricov_instr_ci	0 100	0.00%	100	100.00%		
	THE STALLOU					nscv_msd_co		00090	100	100.00%	-	
	TYPE SIIL_CG					nscv_instr_ci	<b>D</b> 100	0096	100	100.00%	à	V.
	🛨 🛄 TYPE srli_cg					riscv_instr_ci	o 100	0.0096	100	100.00%	i.	
	🕀 🗾 TYPE xor_cg					riscv_instr_co	0 100	.00%	100	100.00%	1	$\checkmark$
	🕒 🗾 TYPE or_cg					riscv_instr_cr	o 100	.0096	100	100.00%		
	🕒 🗾 TYPE and_cg					riscv_instr_co	o 100	.00%	100	100.00%	1	
	🕀 🗾 TYPE xori_cg					riscv_instr_co	o 100	.00%	100	100.00%	1	
	🗄 🗾 TYPE ori cg					riscv instr co	D 100	.0096	100	100.00%	i	

* Covergroups									4	
Name		Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_cove	
j /riscv_instr_pkg/riscv_inst	r cover group		76.24%							
🖃 🔟 TYPE add_cg		riscy instr_co	100.00%	100	100.00%		1	auto	(1)	
CVP add_cg::cp_rs	1	riscy_instr_co	100.00%	100	100.00%		1			
CVP add_cg::cp_rs	2	riscv_instr_co	100.00%	100	100.00%		1			
CVP add_cg::cp_rd		riscv_instr_co	100.00%	100	100.00%		1			
CVP add_cg::cp_rs	1_sign	riscv_instr_co	100.00%	100	100.00%		1			
CVP add_og::cp_rs	2_sign	riscv_instr_co	100.00%	100	100.00%		1			
CVP add_cg::cp_rc	i_sign	riscv_instr_co	100,00%	100	100.00%		1			
bin auto[POSITI	NEI		/1/0	1	100.00%		1			
CVP add corres of	or barzard	risev instr. co.	100.00%	100	100.00%		1			
B) bin autoINO HA	ZARDI	"act_"and_com	10832	1	100.00%	Concession of the local division of the loca	1			
B) bin autoRAW H	AZARDI		728	1	100.00%		1			
B) bin auto[WAF	Covergroups									
B) bin auto[WAV +	Name			Clas	is Type	Coverage	Goal	% of Goal Stat	us Included	Merge instance
G Z CROSS add_cg	discu instraka/riscu instra	er aroun		production of		76.24	A CONTRACT	- Andrewski and Andrewski	and a second second second	
B] bin <auto[po< td=""><td>TYPE and co</td><td>a group</td><td></td><td>ricry</td><td>instr co</td><td>100.008</td><td>10</td><td>100.00%</td><td>1</td><td>30</td></auto[po<>	TYPE and co	a group		ricry	instr co	100.008	10	100.00%	1	30
B) bin <auto[ne< p=""></auto[ne<>	CVP add corres rel			riscu	instr co	100.000	16 10	100 00%	×	di
B) bin <auto[po]< p=""></auto[po]<>	CVP and concorres			rigen	instr co	100 000	10	100.00%		
B) bin <auto[ne< td=""><td>CVP add_cg:cp_132</td><td></td><td></td><td>riceu</td><td>instr co.</td><td>100.000</td><td>10</td><td>100.00%</td><td></td><td></td></auto[ne<>	CVP add_cg:cp_132			riceu	instr co.	100.000	10	100.00%		
B bin <auto[po< td=""><td>B) bin auto/ZEPOI</td><td></td><td></td><td>nacy</td><td></td><td>100.00</td><td>10</td><td>1 100.00%</td><td></td><td></td></auto[po<>	B) bin auto/ZEPOI			nacy		100.00	10	1 100.00%		
B) bin <auto(ne< td=""><td>B) bie suto[DA]</td><td></td><td></td><td></td><td></td><td>21</td><td>2</td><td>1 100.00%</td><td>× 1</td><td></td></auto(ne<>	B) bie suto[DA]					21	2	1 100.00%	× 1	
B) bin cauto[PO	bin auto[SD]					31	5	1 100.00%		
TYPE sub co	D bin auto[GP]					40	17	1 100.00%	Y A	
TYPE addi co	D) bin auto[CD]					40		1 100.00%		
+ TYPE lui cg	D) bin auto[70]					40	13	1 100.00%	Y I	
🗉 🗾 TYPE auipc cg	bin auto[10]					34	10	1 100.00%	Y.	
💽 🗾 TYPE sra_cg	bin auto[[1]					30	19	1 100.00%	Y I	
TYPE sll_cg	b) bin auto[12]					42	10	1 100.00%	Y.	
TYPE srl_cg	bin auto[50]					40	13	1 100.00%	Y.	
+) TYPE stai co	B) bis sub(10)					40	19	1 100.00%	V.	
auto(1	B) bin auto[A0]					48	13	1 100,00%	Y.	
auto(1	B Din auto[A1]					35	12	1 100.00%	V	
cioro(1	b) bin auto[A2]					37	1	1 100.00%	Y.	
auto(1	Di Din auto(A3)					39	10	1 100.00%	Y.	
auto(1	bin auto[44]					30	2	1 100.00%	Y.	
auto(1	bin auto[AS]					41	2	1 100.00%	Y.	
tioto(1	B) bin auto[A6]					35	8	1 100.00%	Y.	
auto(1	B bin auto(A/)					38	57	1 100.00%	× 1	
auto(1	B) bin auto[52]					42	.9	1 100.00%	×.	
auto(1	bin auto[53]					38	12	1 100.00%	×.	
	b) bin auto[54]					36	8	1 100.00%	×.	
auto(1	B) bin auto[55]					34	8	1 100.00%	V.	
auto(1	B) bin auto[56]					44	2	1 100.00%	V.	
auto(1	B bin auto[57]					32	3	1 100.00%	×.	
auto (1	B bin auto[S8]					39	19	1 100.00%		
auto(1	B) bin auto[59]					41	.4	1 100.00%	Y.	
	B bin auto[S10]					38	12	1 100.00%	×.	
	B] bin auto[S11]					44	18	1 100.00%	×.	
	B) bin auto[T3]					41	.5	1 100.00%	×.	
	B] bin auto[T4]					41	5	1 100.00%	V.	
						35	14	1 100.00%	V.	
ortingf	rom tha	mno	rac	$+ \infty$		31	.2	1 100.00%	V.	
ortinel	romune	INDE	L d S	τO	UIS	100.009	M 10	00 100.00%		
						100.009	20 10	00 100.00%		

#### Automated instruction coverage reporting from the Imperas tools

© 2020 Imperas Software Ltd.

# **Instruction Stream Generation**



- Instruction stream generation (ISG) generates random streams of instructions
- Generator given guidance to target specific instruction types and values
  - Many constraints required to get legal instruction sequences
- No predicted results, relies upon reference
- This is just constrained random generation repurposed to processor DV
- Constrained random generation is a well-established part of SoC DV flows

# Google RISC-V Instruction Stream Generation



- High quality SystemVerilog UVM DV infrastructure
- Open source (Apache 2.0)
- Drives a RISC-V core through corner cases and pushes it to the limit
- Requires reference and DUT to generate instruction trace disassembly
- Traces compared as post-process (neutral CSV format)
- Can compare values and program flow
  - dependant upon target capability
- Provides coverage for test quality, and to aid guidance

Open Source SystemVerilog UVM RISC-V Instruction Stream Generator

https://github.com/google/riscv-dv



- Google: open source riscv-dv instruction stream generator
- Metrics : SystemVerilog design + UVM simulator for RTL
  - Now working with Cadence, Mentor, Synopsys RTL simulators
- Imperas: model and simulation golden reference of RISC-V CPU
- Imperas have added Vector and Bitmanip extension instructions to the Functional Coverage

<sup>(</sup>not yet publicly released)

# Case Study: IowRISC Ibex



- Ibex is a small 32 bit RISC-V CPU core (RV32IMC/EMC) with a two stage pipeline, previously known as zero-risky (PULP)
- https://github.com/lowRISC/ibex

#### **Case study : Ibex core verification**





# Bugs Found Using ISG Approach





🙆 Google Cloud

#### © 2020 Imperas Software Ltd.

# Valtrix Builds Executable Test Benches

- Imperas working with Valtrix
- riscvOVPsim as reference model
- Alternative/complementary approach to Instruction Stream Generator



## Agenda



- RISC-V verification issues
- Compliance is not verification
- Reference models and custom instructions
- Processor IP verification
- SoC level verification
- Summary

# **SoC Level Verification**



- What about ...
- Verification of processing elements with multiple RISC-V cores, as is common in AI/ML SoCs?
  - This flow is still evolving / being invented
- Verification of the interface between the processor or processing element and the NoC
  - Simple answer is that NoC verification IP is used
  - This takes some effort, which is taken for granted with traditional processors

## Agenda



- RISC-V verification issues
- Compliance is not verification
- Reference models and custom instructions
- Processor IP verification
- SoC level verification
- Summary

# Summary



- DV is a critical issue for RISC-V processor IP and SoCs
- Compliance testing is a subset of DV
- Reference models are needed, and are now available
- Directed testing, instruction stream generation and test generation/execution are being used for processor IP DV
- Step and compare methodology provides the most efficient DV flows
- More work is needed

# mperas

# Thank you

Larry Lapides LarryL@imperas.com

25-Feb-20