imperas

Hybrid Simulation with Emulation for RISC-V Software Bring Up and Hardware-Software Co-Verification

Duncan Graham and Larry Lapides

June 2023

Agenda

- RISC-V: Who/What/Where/When/Why
- RISC-V: How … to be successful
- Processor modeling
- Virtual and Hybrid Platforms with Helium, Palladium and Protium
- Summary

Agenda

- **RISC-V: Who/What/Where/When/Why**
- RISC-V: How … to be successful
- Processor modeling
- Virtual and Hybrid Platforms with Helium, Palladium and Protium
- Summary

Imperas RISC-V Timeline

- **Q1 2017: Imperas joins the RISC-V Foundation; builds first RISC-V processor model**
- Q3 2017: Imperas starts participating in the Compliance Working Group; builds/donates tests
- Q1 2018: Imperas introduces methodology for adding/optimizing custom instructions (architecture exploration) for RISC-V cores
- **Q2 2018: First paying customer using Imperas RISC-V models and tools for software development and design verification (DV)**
- Q4 2018: First paying customer using Imperas RISC-V models and tools for architecture exploration
- **Q1 2019: First tape out of RISC-V SoC based on using Imperas model as DV reference model**
- Q2 2019: Imperas releases riscvOVPsimPlus, free instruction set simulator supporting the full RISC-V specification
- Q1 2020: Imperas starts working with the OpenHW Group and individual members on DV of Core-V cores
- Q4 2021: Imperas introduces ImperasDV RISC-V verification product line
- Q1 2022: Imperas introduces RVVI (RISC-V Verification Interface) as an open standard on GitHub for the RISC-V processor DV community
- Q4 2022: Imperas introduces functional coverage verification IP for RISC-V processor DV
- **Q2 2023: Imperas and Cadence develop OEM relationship enabling Cadence to resell Imperas RISC-V models**

RISC-V Freedom Enables Domain Specific Processing

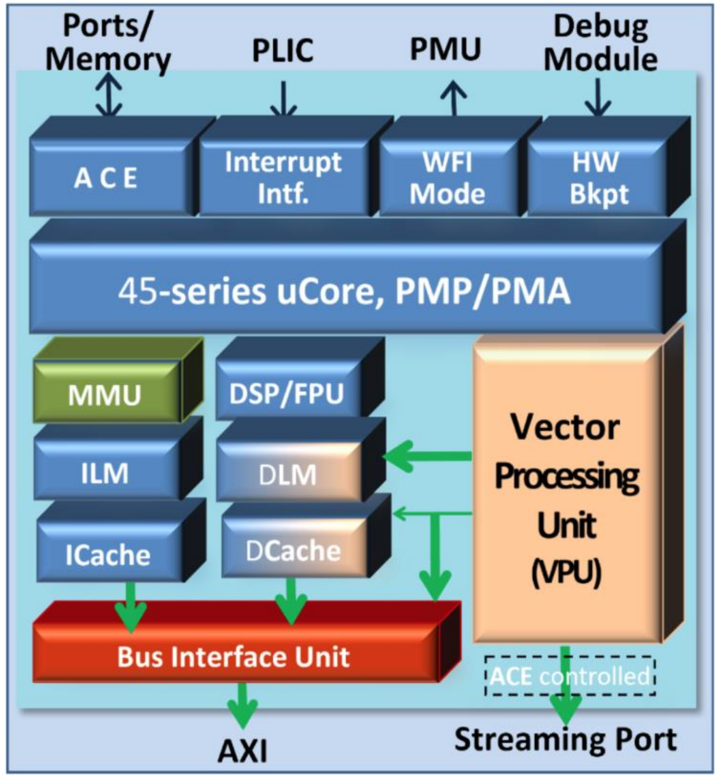
- **Who**: RISC-V users include traditional semiconductor companies, and embedded systems companies now practicing vertical integration by developing their own SoCs
- **What**: RISC-V is an open instruction set architecture (ISA), it is not a processor implementation
- **Where**: RISC-V is growing in 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

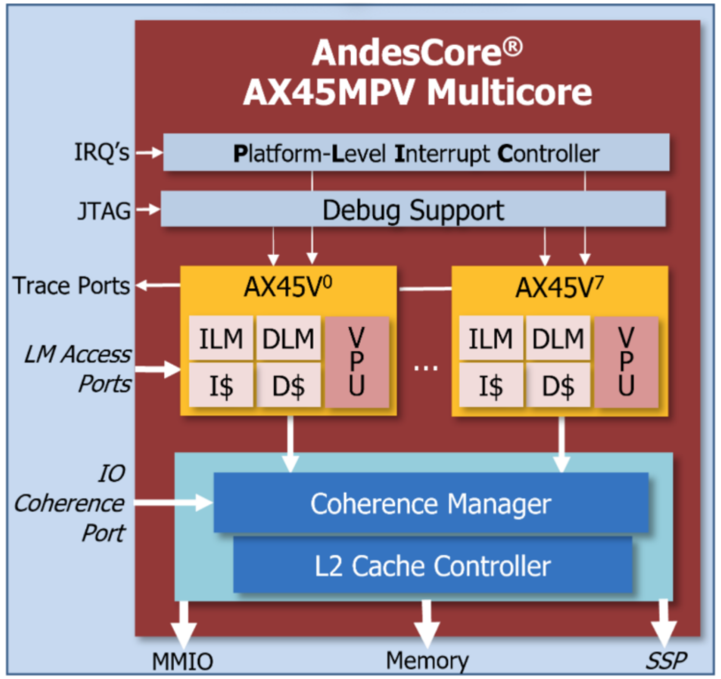
Agenda

- RISC-V: Who/What/Where/When/Why
- **RISC-V: How … to be successful**
- Processor modeling
- Virtual and Hybrid Platforms with Helium, Palladium and Protium
- Summary

RISC-V Processor Complexity

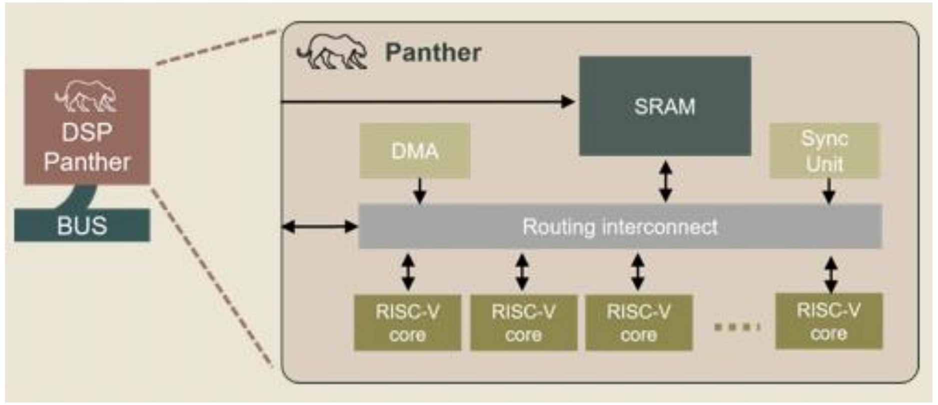
- RISC-V is a modular instruction set architecture
- Any extension (functional group of instructions, e.g. atomics, compressed, floating point, vector) can be added to the base processor
- Then add in interrupts, privilege modes, Debug mode, multi-hart (multi-core), etc. and it gets complex
- Then processor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



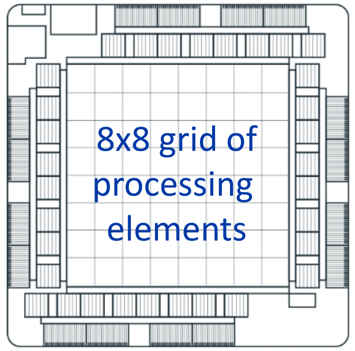


RISC-V Processing Subsystems

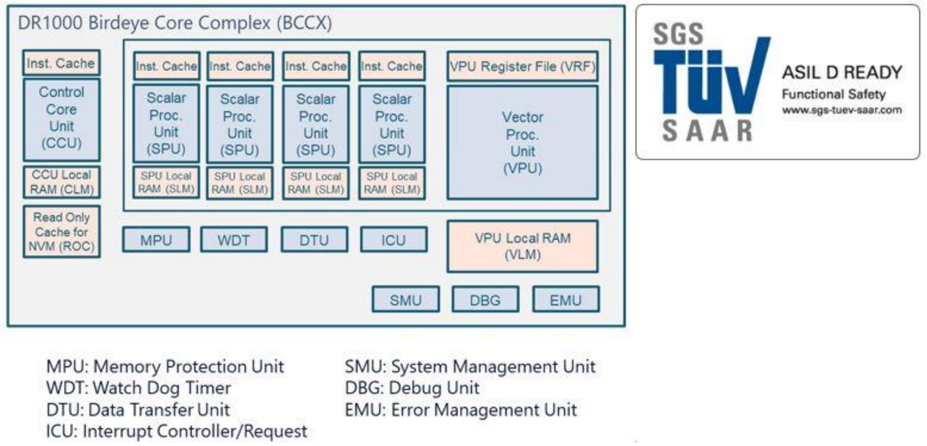
- Multi-processor subsystems are commonly being developed using RISC-V cores
- Application areas include DSP, AI/ML and packet processing
- This adds complexity to both the DV and software development tasks



Dolphin Design "Panther" DSP



Meta Training & Inference Accelerator



MPU: Memory Protection Unit	SMU: System Management Unit
WDT: Watch Dog Timer	DBG: Debug Unit
DTU: Data Transfer Unit	EMU: Error Management Unit
ICU: Interrupt Controller/Request	

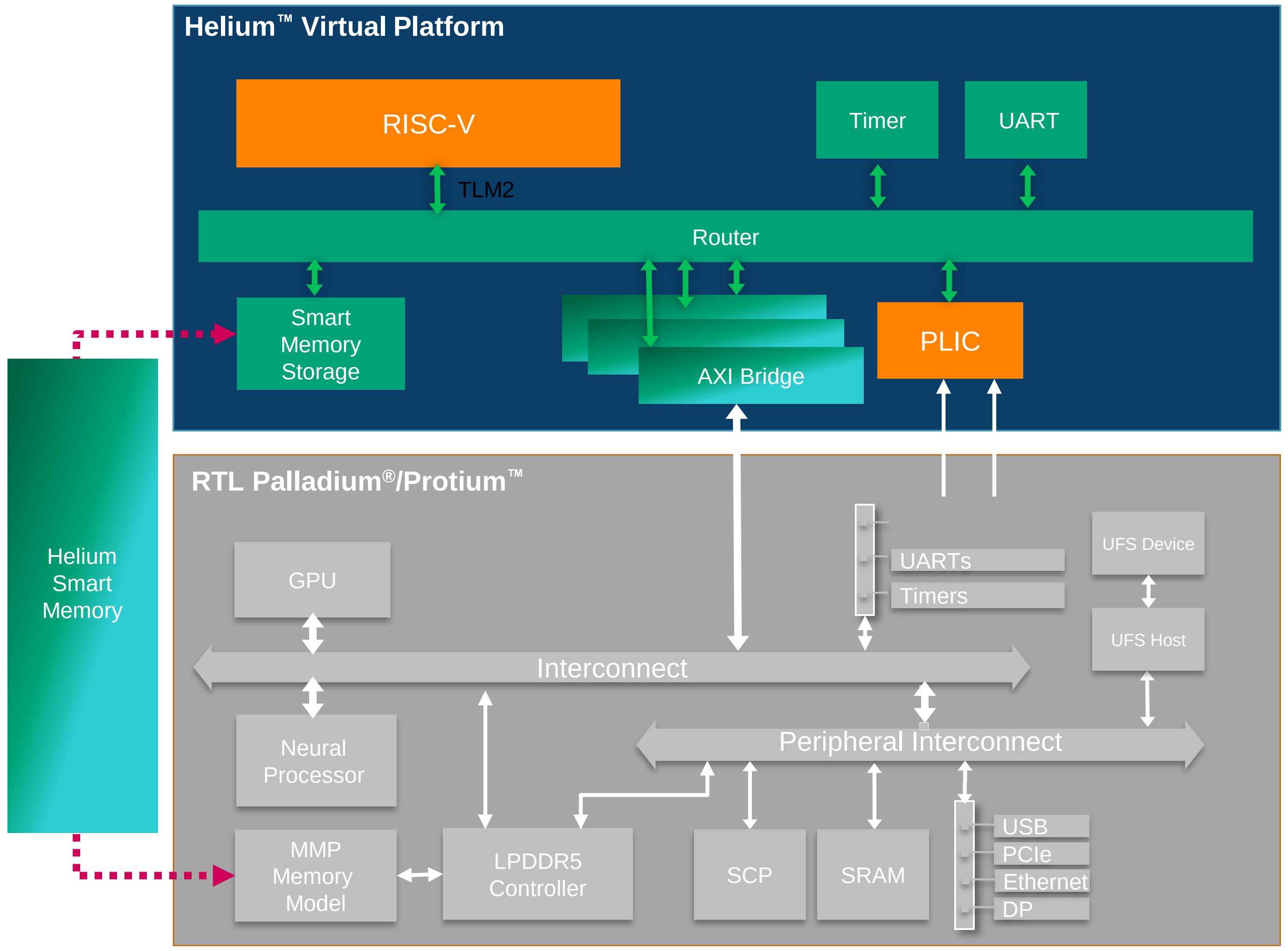
NSITEXE Data Flow Processor

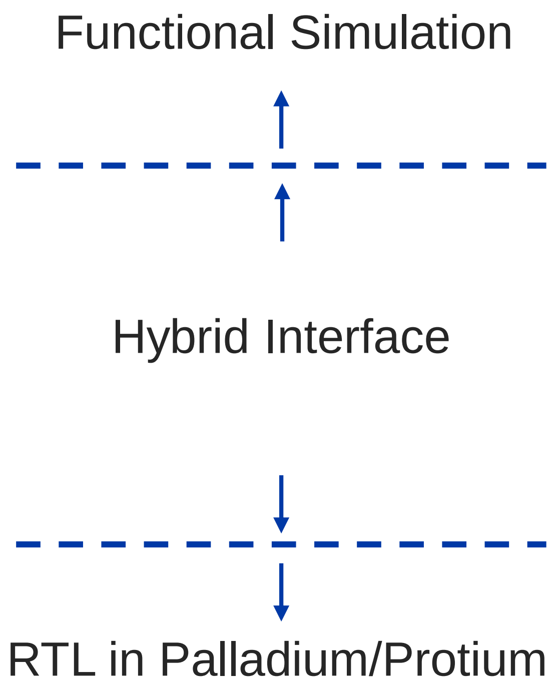
Technologies & Methodologies for Processor Verification and Software Development

- Processor verification needs RTL simulation for comprehensive processor DV
- Also needs an asynchronous-step-compare methodology, supported by verification IP and functional coverage
- Software development can use virtual platform simulation and FPGA prototypes
- These conventional techniques are not enough for the complex RISC-V processors and processing subsystems being designed today

- **SoC verification, and software development and validation for the complex RISC-V processors and processing subsystems, requires hardware-software co-verification and prototyping using a hybrid software simulation – hardware emulation methodology**

Example Hybrid Platform with Cadence Helium and Palladium





cādence®

Agenda

- RISC-V: Who/What/Where/When/Why
- RISC-V: How … to be successful
- **Processor modeling**
- Virtual and Hybrid Platforms with Helium, Palladium and Protium
- Summary

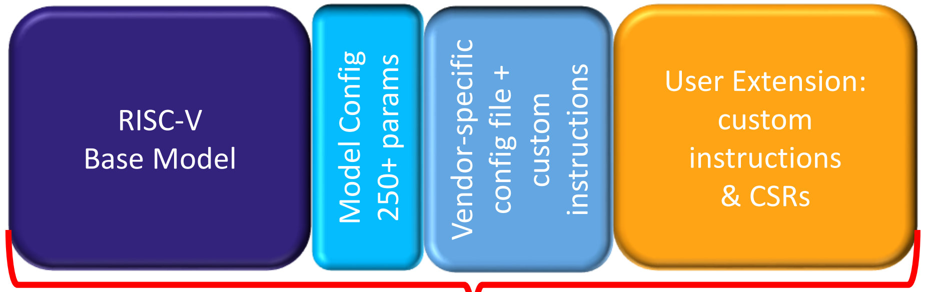
RISC-V Model Requirements

- Model the ISA, *including all versions of the ratified spec*, and stable unratified extensions
- If the model is open source, users should not have to maintain their own fork of the model
- 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, SiFive, Codasip, MIPS, NSITEXE, OpenHW, …
- Well-defined test process including coverage metrics and mutation testing
- 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.
- Interface to architecture exploration tools including extensibility to timing estimation

- Most RISC-V ISSs can meet one or two of these requirements
- Imperas models and simulators were built to satisfy these requirements, and matured through usage on non-RISC-V ISAs over the last 15+ years

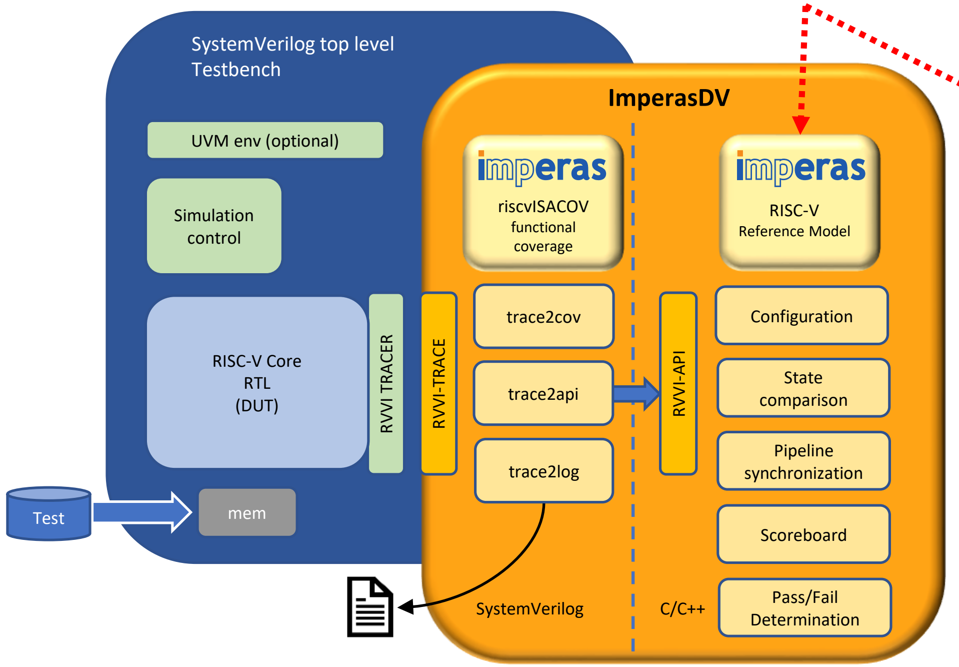
Imperas OVP RISC-V Models are used for Processor DV & SW Development and Architecture Analysis

- Base model implements RISC-V specification in full
- Fully user configurable to select which ISA extensions/versions
- Imperas provides methodology to easily extend base model

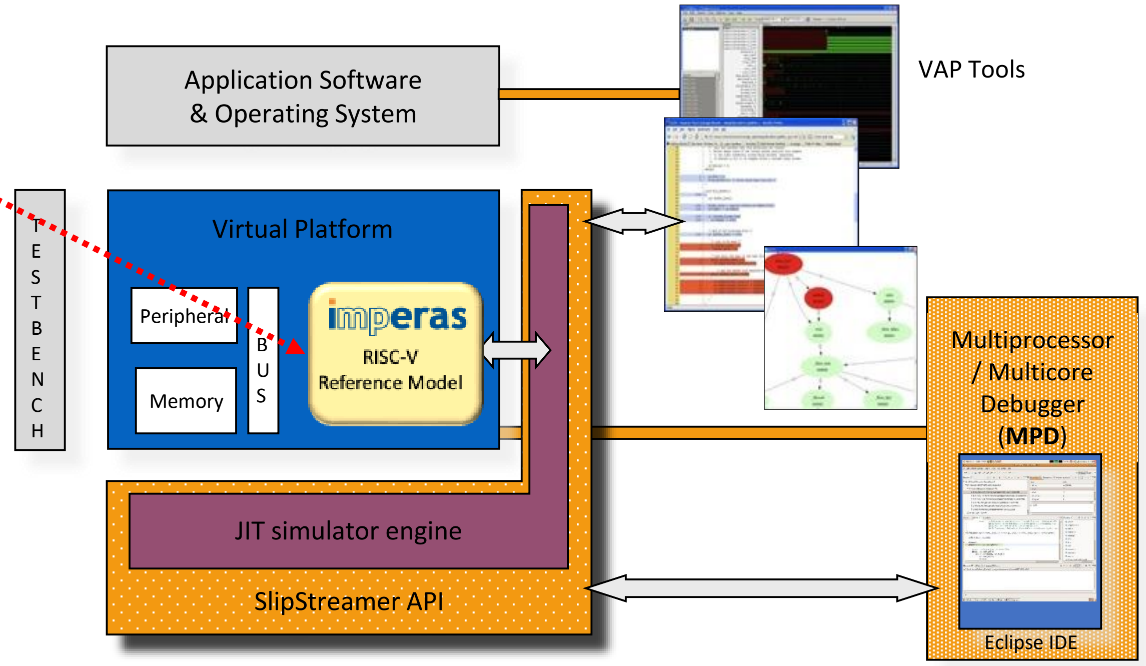


RISC-V Base Model
Model Config 250+ params
Vendor-specific config file + custom instructions
User Extension: custom instructions & CSRs

Processor DV Flow

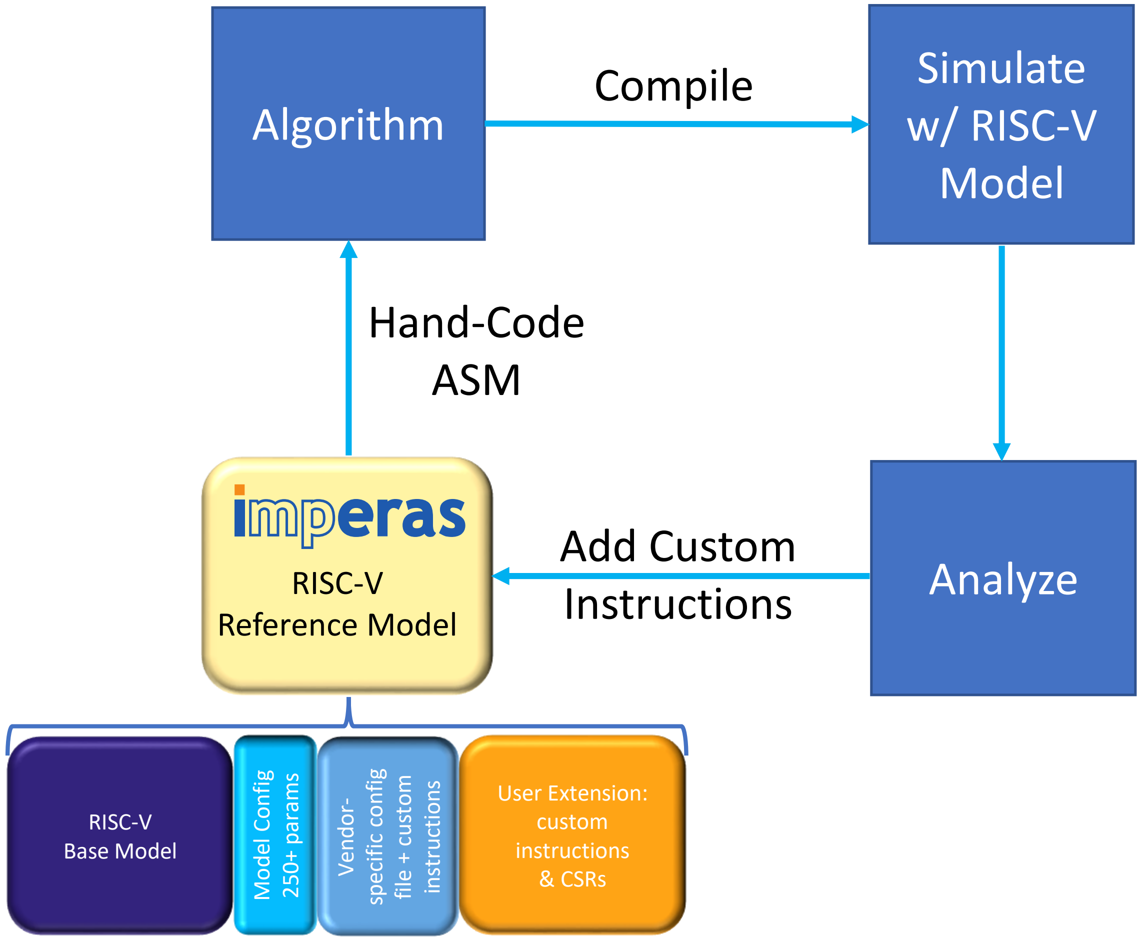


Software Simulation Environment

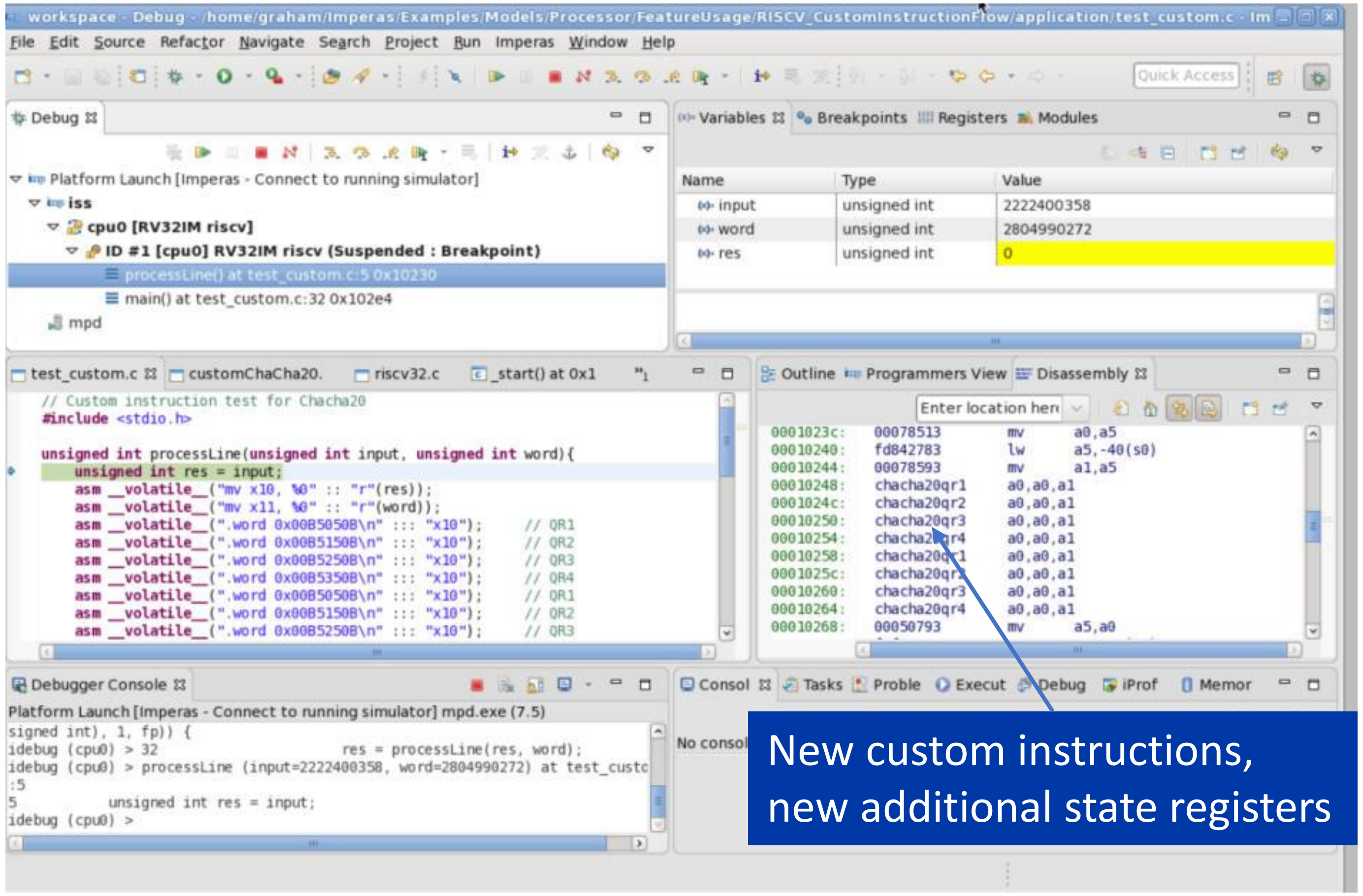


Models Drive Customization

- 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



Software Analysis Tools Automatically Work With the Custom Instructions


workspace - Debug - /home/graham/Imperas/Examples/Models/Processor/FeatureUsage/RISCV_CustomInstructionFlow/application/test_custom.c - Im
File Edit Source Refactor Navigate Search Project Run Imperas Window Help
Quick Access
Debug
Variables Breakpoints Registers Modules
test_custom.c customChaCha20. riscv32.c _start() at 0x1
Outline Programmers View Disassembly
Enter location here
Debugger Console
Consol Tasks Proble Execut Debug iProf Memor
No consol

```
Platform Launch [Imperas - Connect to running simulator]
  iss
    cpu0 [RV32IM riscv]
      ID #1 [cpu0] RV32IM riscv (Suspended : Breakpoint)
        processLine() at test_custom.c:5 0x10230
        main() at test_custom.c:32 0x102e4
  mpd
```

Name	Type	Value
input	unsigned int	2222400358
word	unsigned int	2804990272
res	unsigned int	0

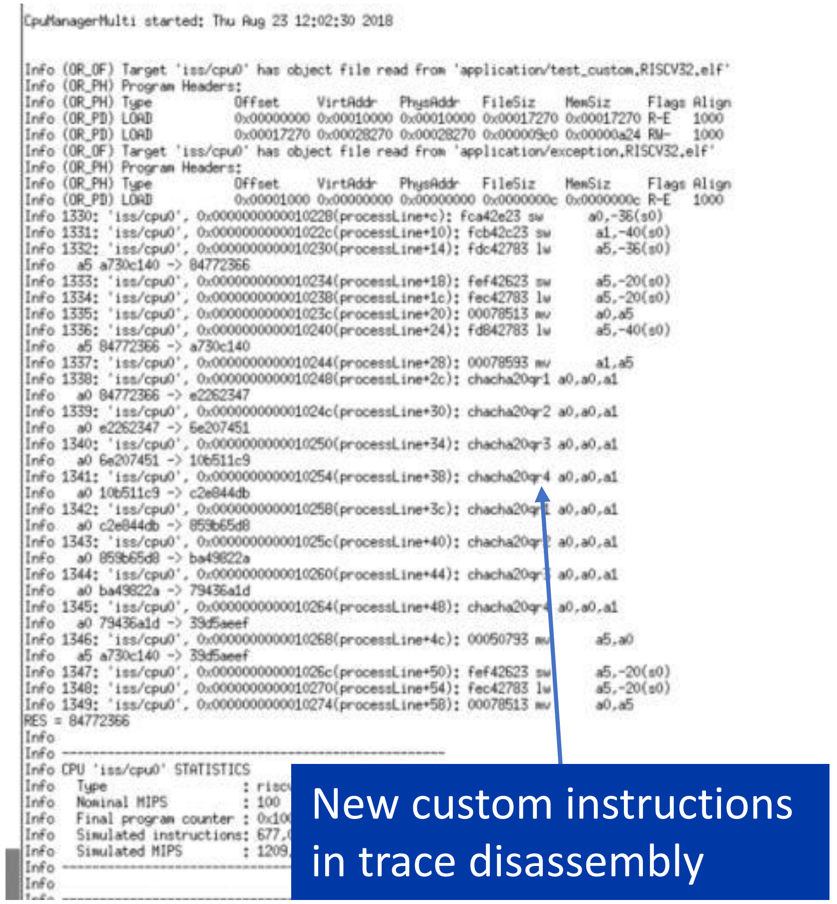
```c
// Custom instruction test for Chacha20
#include <stdio.h>

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 0x00B5050B\n" ::: "x10");      // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");      // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");      // QR3
    asm __volatile__(".word 0x00B5350B\n" ::: "x10");      // QR4
    asm __volatile__(".word 0x00B5050B\n" ::: "x10");      // QR1
    asm __volatile__(".word 0x00B5150B\n" ::: "x10");      // QR2
    asm __volatile__(".word 0x00B5250B\n" ::: "x10");      // QR3
```

```
0001023c:   00078513      mv     a0,a5
00010240:   fd842783      lw     a5,-40(s0)
00010244:   00078593      mv     a1,a5
00010248:   chacha20qr1   a0,a0,a1
0001024c:   chacha20qr2   a0,a0,a1
00010250:   chacha20qr3   a0,a0,a1
00010254:   chacha20qr4   a0,a0,a1
00010258:   chacha20qr1   a0,a0,a1
0001025c:   chacha20qr2   a0,a0,a1
00010260:   chacha20qr3   a0,a0,a1
00010264:   chacha20qr4   a0,a0,a1
00010268:   00050793      mv     a5,a0
```

```
Platform Launch [Imperas - Connect to running simulator] mpd.exe (7.5)
signed int), 1, fp)) {
idebug (cpu0) > 32                      res = processLine(res, word);
idebug (cpu0) > processLine (input=2222400358, word=2804990272) at test_custo
:5
5           unsigned int res = input;
idebug (cpu0) >
```

New custom instructions, new additional state registers

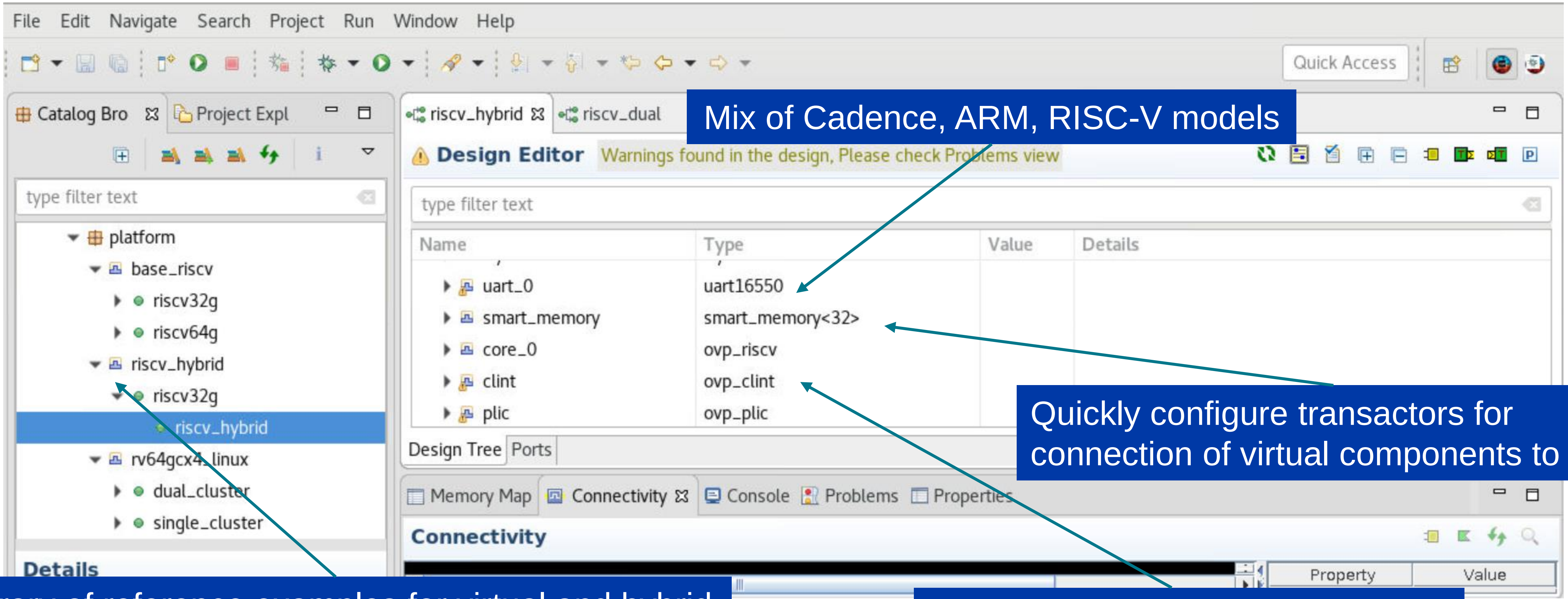


```
CpuManagerMulti started: Thu Aug 23 12:02:30 2018

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
Info (OR_PD) LOAD           0x00017270 0x00028270 0x00028270 0x000009c0 0x00000a24 RW-   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    MemSiz     Flags Align
Info (OR_PD) LOAD           0x00001000 0x00000000 0x00000000 0x0000000c 0x0000000c R-E   1000
Info 1330: 'iss/cpu0', 0x0000000000010228(processLine+c): fca42e23 sw       a0,-36(s0)
Info 1331: 'iss/cpu0', 0x000000000001022c(processLine+10): fcb42c23 sw      a1,-40(s0)
Info 1332: 'iss/cpu0', 0x0000000000010230(processLine+14): fdc42783 lw      a5,-36(s0)
Info   a5 a730c140 -> 84772366
Info 1333: 'iss/cpu0', 0x0000000000010234(processLine+18): fef42623 sw      a5,-20(s0)
Info 1334: 'iss/cpu0', 0x0000000000010238(processLine+1c): fec42783 lw      a5,-20(s0)
Info 1335: 'iss/cpu0', 0x000000000001023c(processLine+20): 00078513 mv      a0,a5
Info 1336: 'iss/cpu0', 0x0000000000010240(processLine+24): fd842783 lw      a5,-40(s0)
Info   a5 84772366 -> a730c140
Info 1337: 'iss/cpu0', 0x0000000000010244(processLine+28): 00078593 mv      a1,a5
Info 1338: 'iss/cpu0', 0x0000000000010248(processLine+2c): chacha20qr1 a0,a0,a1
Info   a0 84772366 -> e2262347
Info 1339: 'iss/cpu0', 0x000000000001024c(processLine+30): chacha20qr2 a0,a0,a1
Info   a0 e2262347 -> 6e207451
Info 1340: 'iss/cpu0', 0x0000000000010250(processLine+34): chacha20qr3 a0,a0,a1
Info   a0 6e207451 -> 10b511c9
Info 1341: 'iss/cpu0', 0x0000000000010254(processLine+38): chacha20qr4 a0,a0,a1
Info   a0 10b511c9 -> c2e844db
Info 1342: 'iss/cpu0', 0x0000000000010258(processLine+3c): chacha20qr1 a0,a0,a1
Info   a0 c2e844db -> 859b65d8
Info 1343: 'iss/cpu0', 0x000000000001025c(processLine+40): chacha20qr2 a0,a0,a1
Info   a0 859b65d8 -> ba49822a
Info 1344: 'iss/cpu0', 0x0000000000010260(processLine+44): chacha20qr3 a0,a0,a1
Info   a0 ba49822a -> 79436a1d
Info 1345: 'iss/cpu0', 0x0000000000010264(processLine+48): chacha20qr4 a0,a0,a1
Info   a0 79436a1d -> 39d5aeef
Info 1346: 'iss/cpu0', 0x0000000000010268(processLine+4c): 00050793 mv      a5,a0
Info   a5 a730c140 -> 39d5aeef
Info 1347: 'iss/cpu0', 0x000000000001026c(processLine+50): fef42623 sw      a5,-20(s0)
Info 1348: 'iss/cpu0', 0x0000000000010270(processLine+54): fec42783 lw      a5,-20(s0)
Info 1349: 'iss/cpu0', 0x0000000000010274(processLine+58): 00078513 mv      a0,a5
RES = 84772366
Info
Info -----------------------------------------------------------
Info CPU 'iss/cpu0' STATISTICS
Info   Type                  : riscv
Info   Nominal MIPS          : 100
Info   Final program counter : 0x10
Info   Simulated instructions: 677,
Info   Simulated MIPS        : 1209.
Info ------------------------------
Info
Info ------------------------------
```

New custom instructions in trace disassembly

Imperas Models Easily Assembled into Virtual/Hybrid Platforms in Helium



- platform
  - base_riscv
    - riscv32g
    - riscv64g
  - riscv_hybrid
    - riscv32g
      - riscv_hybrid
  - rv64gcx4_linux
    - dual_cluster
    - single_cluster

Name	Type	Value	Details
uart_0	uart16550		
smart_memory	smart_memory<32>		
core_0	ovp_riscv		
clint	ovp_clint		
plic	ovp_plic		

Mix of Cadence, ARM, RISC-V models

Quickly configure transactors for connection of virtual components to RTL

Library of reference examples for virtual and hybrid platforms with bare-metal and Linux workloads to jump-start project development

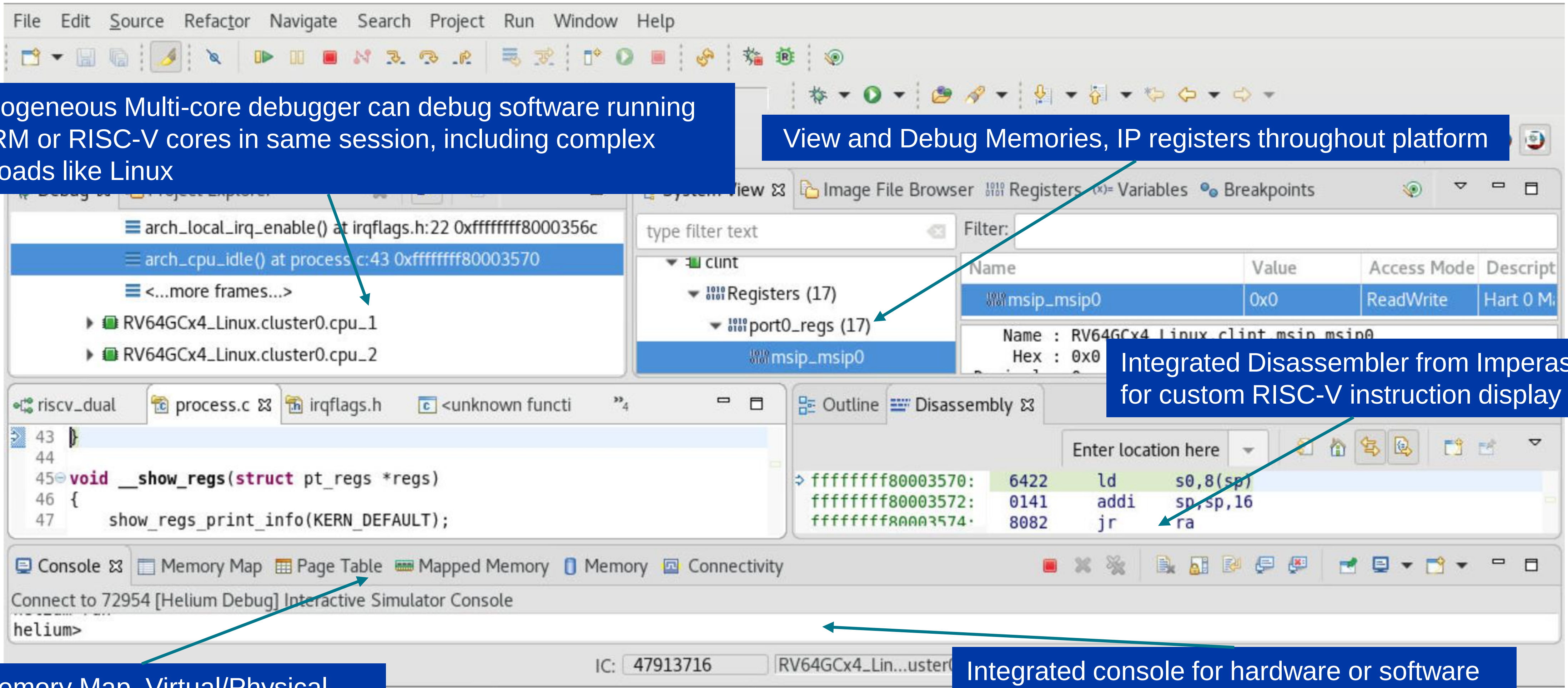
Instantiate, configure and connect Imperas RISC-V models

cādence

Heterogeneous Multi-core Helium System-level Debugger

Heterogeneous Multi-core debugger can debug software running on ARM or RISC-V cores in same session, including complex workloads like Linux

View and Debug Memories, IP registers throughout platform



```
arch_local_irq_enable() at irqflags.h:22 0xffffffff8000356c
arch_cpu_idle() at process.c:43 0xffffffff80003570
<...more frames...>
RV64GCx4_Linux.cluster0.cpu_1
RV64GCx4_Linux.cluster0.cpu_2
```

Name	Value	Access Mode	Descript
msip_msip0	0x0	ReadWrite	Hart 0 M

```
Name : RV64GCx4_Linux.clint.msip_msip0
 Hex : 0x0
```

```
}

void __show_regs(struct pt_regs *regs)
{
    show_regs_print_info(KERN_DEFAULT);
```

```
ffffffff80003570:   6422    ld      s0,8(sp)
ffffffff80003572:   0141    addi    sp,sp,16
ffffffff80003574:   8082    jr      ra
```

```
Connect to 72954 [Helium Debug] Interactive Simulator Console
helium>
```

Integrated Disassembler from Imperas for custom RISC-V instruction display

View Memory Map, Virtual/Physical Memory and Memory Mapped to RTL

Integrated console for hardware or software debugging operations

cādence

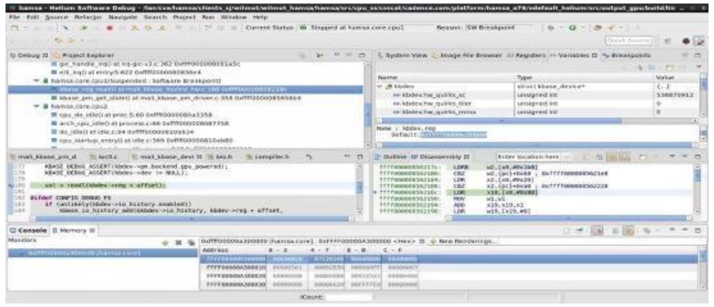
Agenda

- RISC-V: Who/What/Where/When/Why
- RISC-V: How … to be successful
- Processor modeling
- **Virtual and Hybrid Platforms with Helium, Palladium and Protium**
- Summary

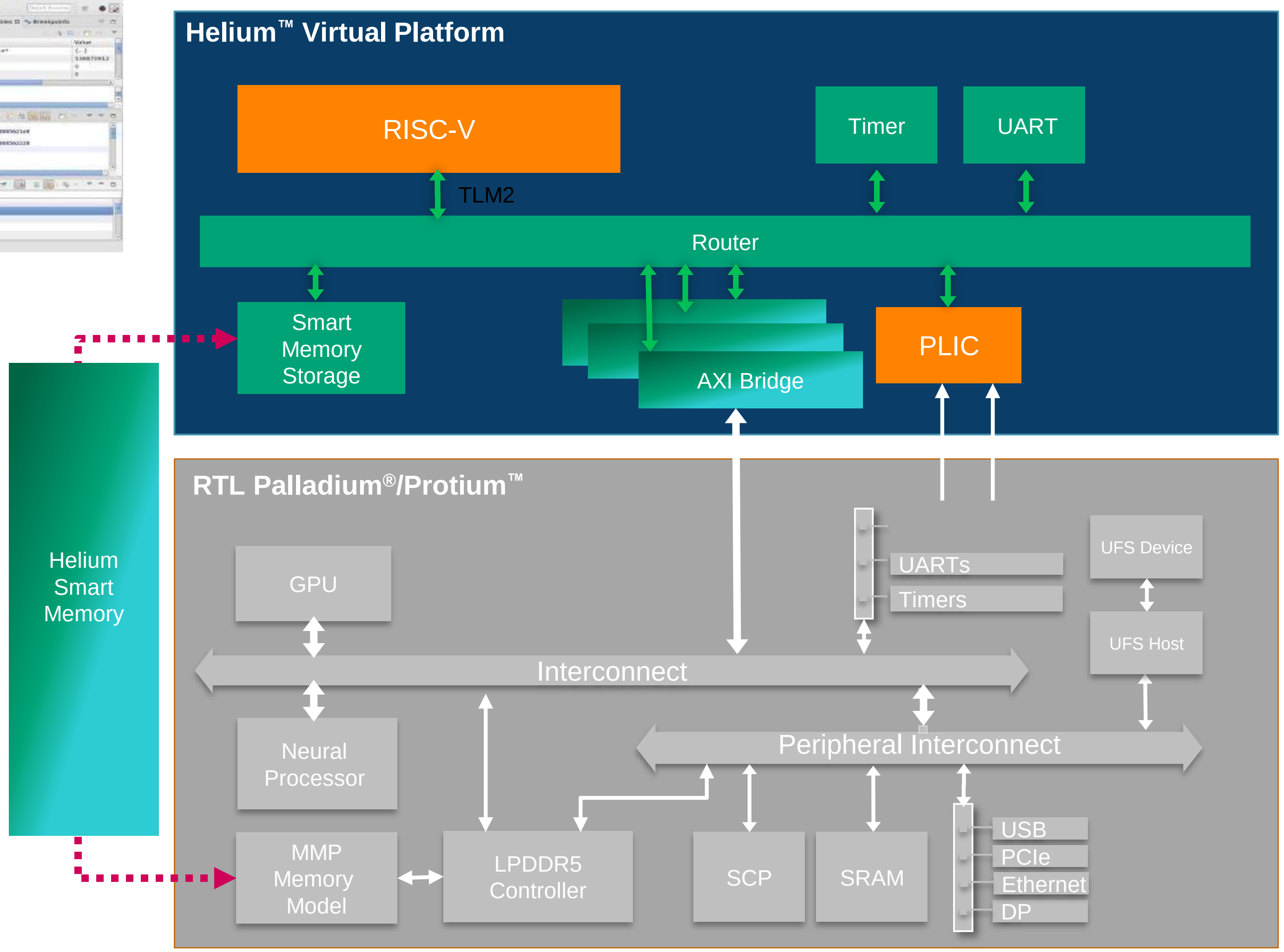
Why Hybrid Emulation-Simulation Systems?

- Hardware emulation is valuable for hardware-software co-verification, and for low level software development, porting and bring up
- Hardware emulation is expensive, typically a scarce resource in companies
- Hardware emulation is 100x slower than real time
  - Need to start execution from boot up or reset
  - What happens when the interesting events occur after billions of instructions, e.g. after Linux boot?
    - Can take minutes, or even hours, of “wasted” emulation time to get to the interesting events
- The hybrid system takes advantage of the speed of software simulation to get to the interesting events in seconds

Helium-Palladium SoC Hybrid Example



Embedded software debugger



Component Color Key

RTL	▨
TLM	🟩
Hybrid	🟩
Imperas	🟧
Adapters	◀···▶

Imperas-Cadence Joint Customer: Software Development/Optimization/Test on AI/ML SoC

- SoC has 140+ cores
  - Andes RISC-V cores with mix of scalar and vector processors
  - Processors used for a) AI/ML, b) running OS, c) SoC functions (power management, communications, …)
- Platform software simulation runs @ > 500MIPS
- Users can run full platform, or subsets for AI/ML, FW, OS, …
- Runs real software (production binaries)
- Software up and running in virtual platform one year before RTL tapeout
- Software ran within days on the first silicon
- ***Helium Hybrid with Palladium emulation used for firmware testing***

Agenda

- RISC-V: Who/What/Where/When/Why
- RISC-V: How … to be successful
- Processor modeling
- Hybrid simulation-emulation with Helium and Palladium
- **Summary**

High Quality RISC-V Models Are Required for RISC-V SoC Success

- Use cases
  - Processor and SoC verification
  - Software development, debug and test
- Imperas OVP Fast Processor Models satisfy RISC-V project requirements
- Hybrid simulation-emulation with Imperas RISC-V models is needed for complex RISC-V processors and/or RISC-V based SoCs
  - Connect RISC-V models to SoC RTL to co-develop software and hardware

imperas

Thank you

Duncan Graham

graham@imperas.com