Open Virtual Platforms APIs Enable High-Quality, Easily Maintained Processor Models

Larry Lapides, Lee Moore, Simon Davidmann

27 June 2024

Open Virtual Platforms (OVP) APIs Enable High Quality RISC-V Models

- Use cases and requirements for RISC-V processor models
- ImperasFPM fast processor model architecture
- OVP APIs
- Case studies
- Summary

OVP APIs Enable High Quality RISC-V Models

- **Use cases and requirements for RISC-V processor models**
- ImperasFPM fast processor model architecture
- OVP APIs
- Case studies
- Summary

RISC-V Processor Model Use Cases

Compliance testing

CPU-centric software development

Architecture exploration

Many core system simulation

Toolchain development and testing

RISC-V Processor Model

Hardware accelerated verification (HAV)

Entry level verification

Regression testing

Processor Verification

Virtual Prototypes

RISC-V Processor Model Requirements are Driven by the Use Cases

This is a lot more than just an Instruction Set Simulator (ISS)

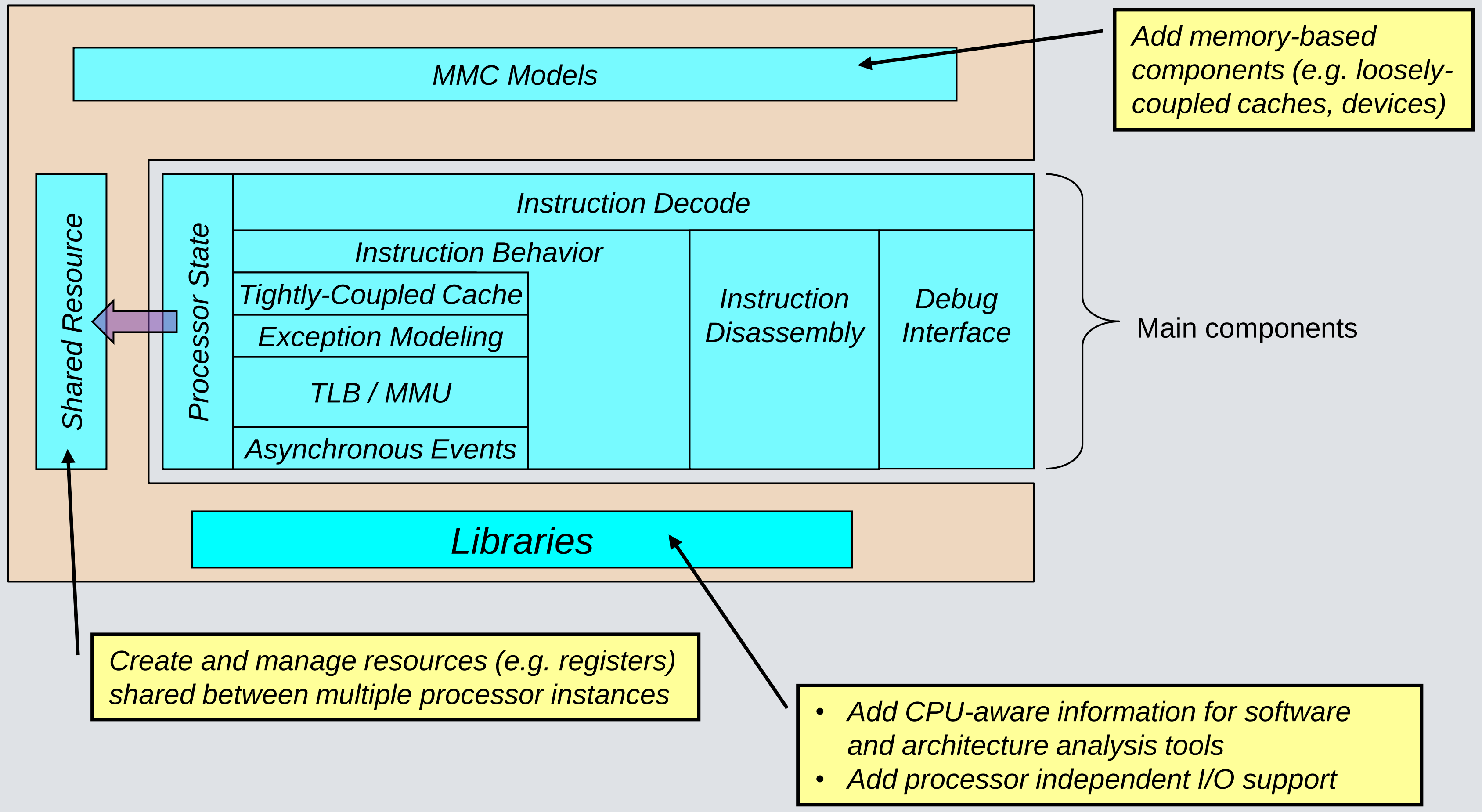
- Model the ISA, including all versions of the ratified spec, and stable unratified extensions
- Easily update and configure the model for the next project – model cannot be "one-off"
- User-extendable for custom instructions, registers, …
- Model actual processor IP, e.g. Andes, SiFive, OpenHW CORE-V, MIPS, ARC-V, …
- Well-defined test process including coverage metrics
- Interface to other simulators, e.g. SystemVerilog, SystemC, …
- Interface to software debug tools, e.g. GDB/Eclipse, Lauterbach, GHS, …
- Interface to software analysis tools including access to processor internal state, etc.
- Interface to architecture exploration tools including extensibility to timing estimation

OVP APIs Enable High Quality RISC-V Models

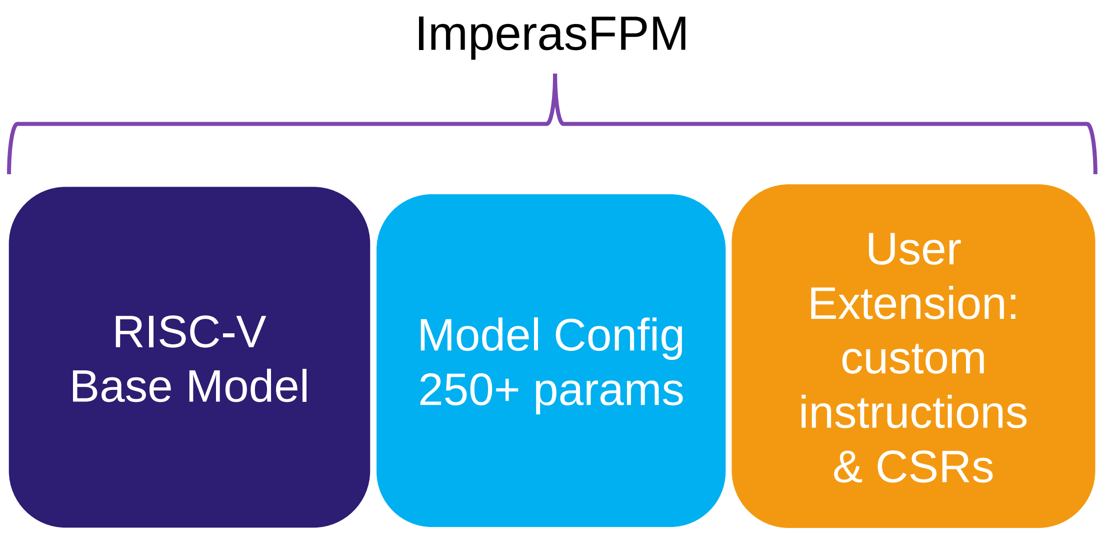
- Use cases and requirements for RISC-V processor models
- **ImperasFPM fast processor model architecture**
- OVP APIs
- Case studies
- Summary

Components of ImperasFPM RISC-V Processor Models

- Models are built in C using OVP APIs
- APIs are supported by a simulator engine
- All models have both C and SystemC/TLM2 native interfaces


MMC Models
Add memory-based components (e.g. loosely-coupled caches, devices)
Shared Resource
Processor State
Instruction Decode
Instruction Behavior
Tightly-Coupled Cache
Exception Modeling
TLB / MMU
Asynchronous Events
Instruction Disassembly
Debug Interface
Main components
Libraries
Create and manage resources (e.g. registers) shared between multiple processor instances
• Add CPU-aware information for software and architecture analysis tools
• Add processor independent I/O support

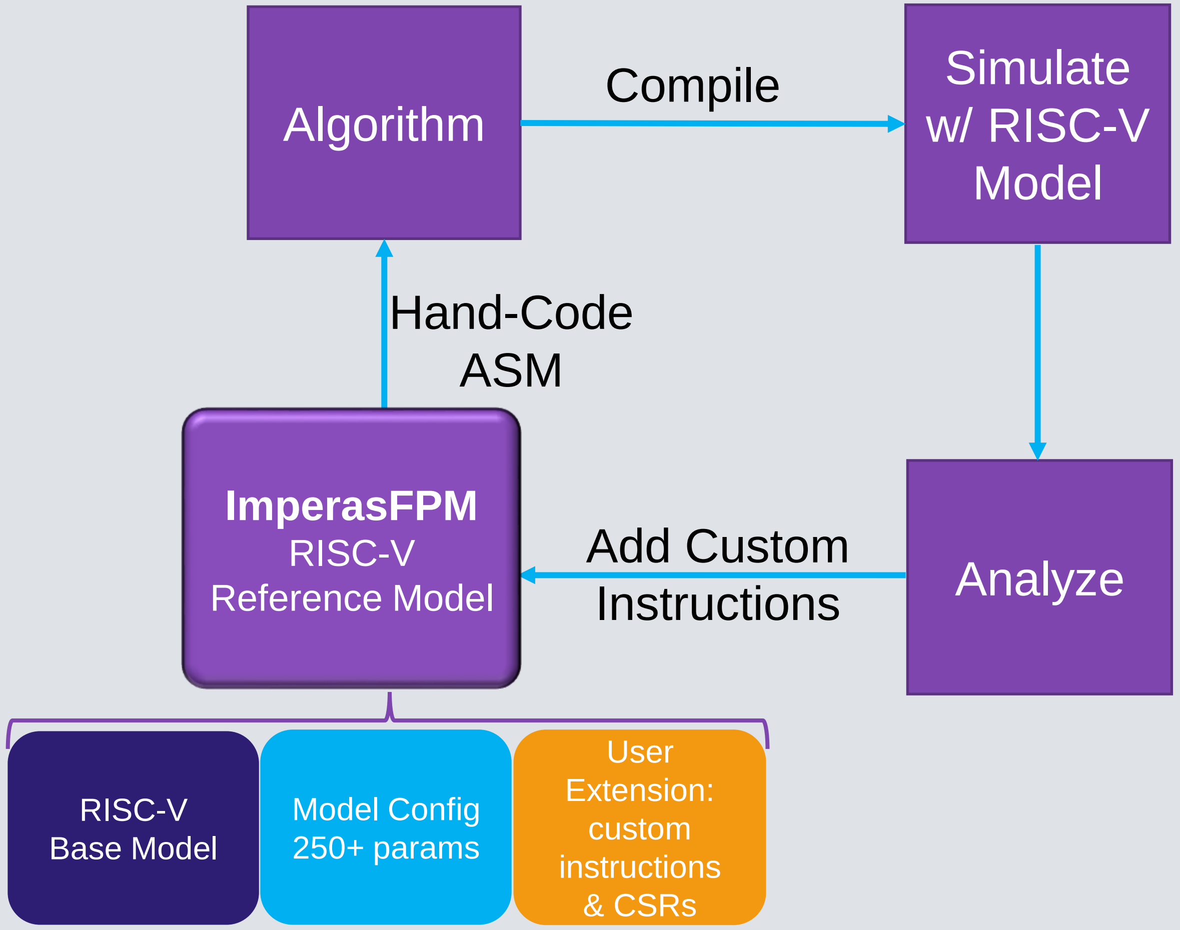
ImperasFPMs (Fast Processor Models) for RISC-V



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

Models Drive Customization

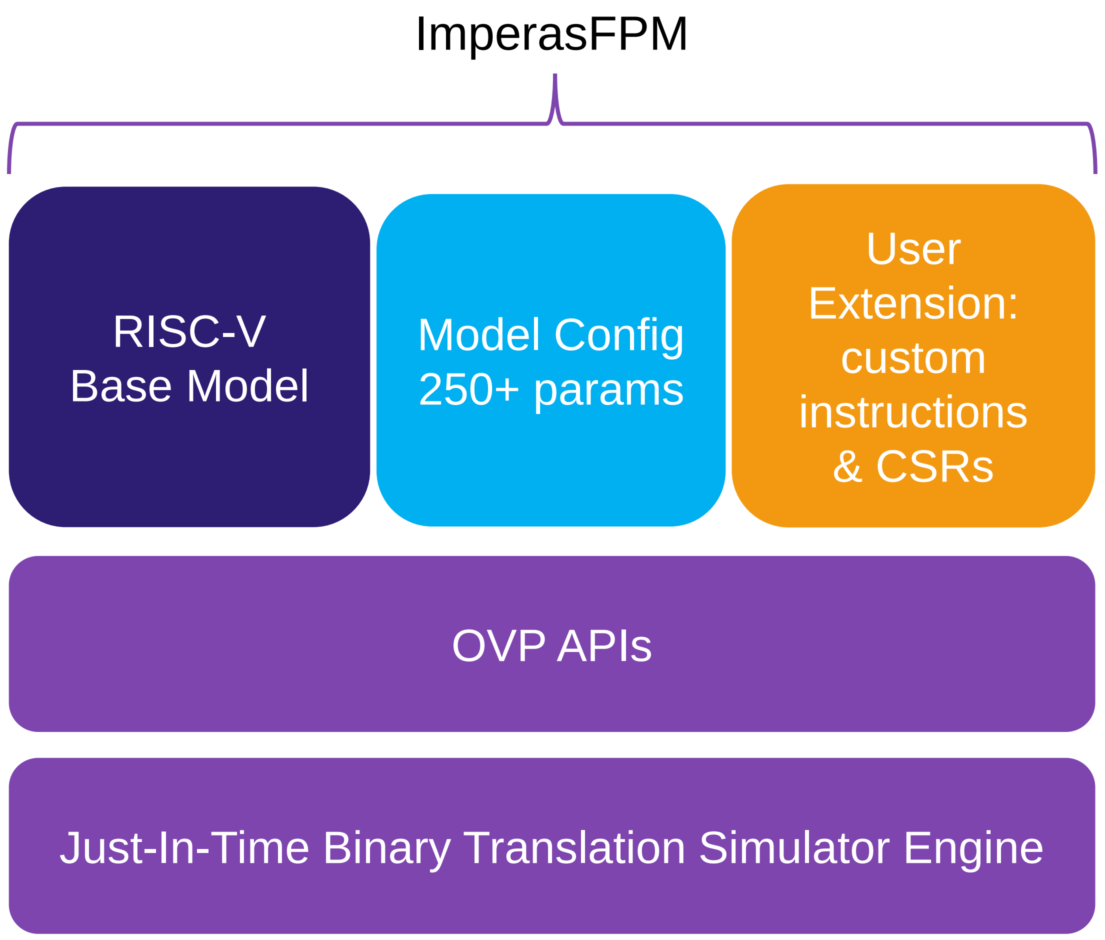
- In the RISC-V world, custom instructions are added to optimize a specific application or set of applications within a domain
  - "Domain-Specific Processors"
- Models let you explore custom instructions quickly
  - Much faster to develop/analyze custom instructions in the model than by writing RTL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



OVP APIs Enable High Quality RISC-V Models

- Use cases and requirements for RISC-V processor models
- ImperasFPM fast processor model architecture
- **OVP APIs**
- Case studies
- Summary

ImperasFPM Architecture


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

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

Why the ImperasFPM Architecture Works Well

- Performance optimization: When a specific API’s performance is improved, it impacts all the models using that API
- Maintenance: When the functionality of a specific API is fixed, it flows to all the models using that API
- Documentation: Having APIs leveraged over a range of models enables better documentation of the APIs (and of the models)
- Tool interfaces: When a new model is built, existing tools automatically work with the new model

- As a result, many fewer engineers are needed to develop and maintain the ImperasFPM models
- As a result, ImperasFPM users can easily add custom features to their models

OVP APIs Enable High Quality RISC-V Models

- Use cases and requirements for RISC-V processor models
- ImperasFPM fast processor model architecture
- OVP APIs
- **Case studies**
- Summary

ImperasFPM Use Cases

Compliance testing

CPU-centric software development

Architecture exploration

Many core system simulation

RISC-V Base Model

Model Config

Custom extensions

ImperasFPM

Toolchain development and testing

Hardware accelerated verification (HAV)

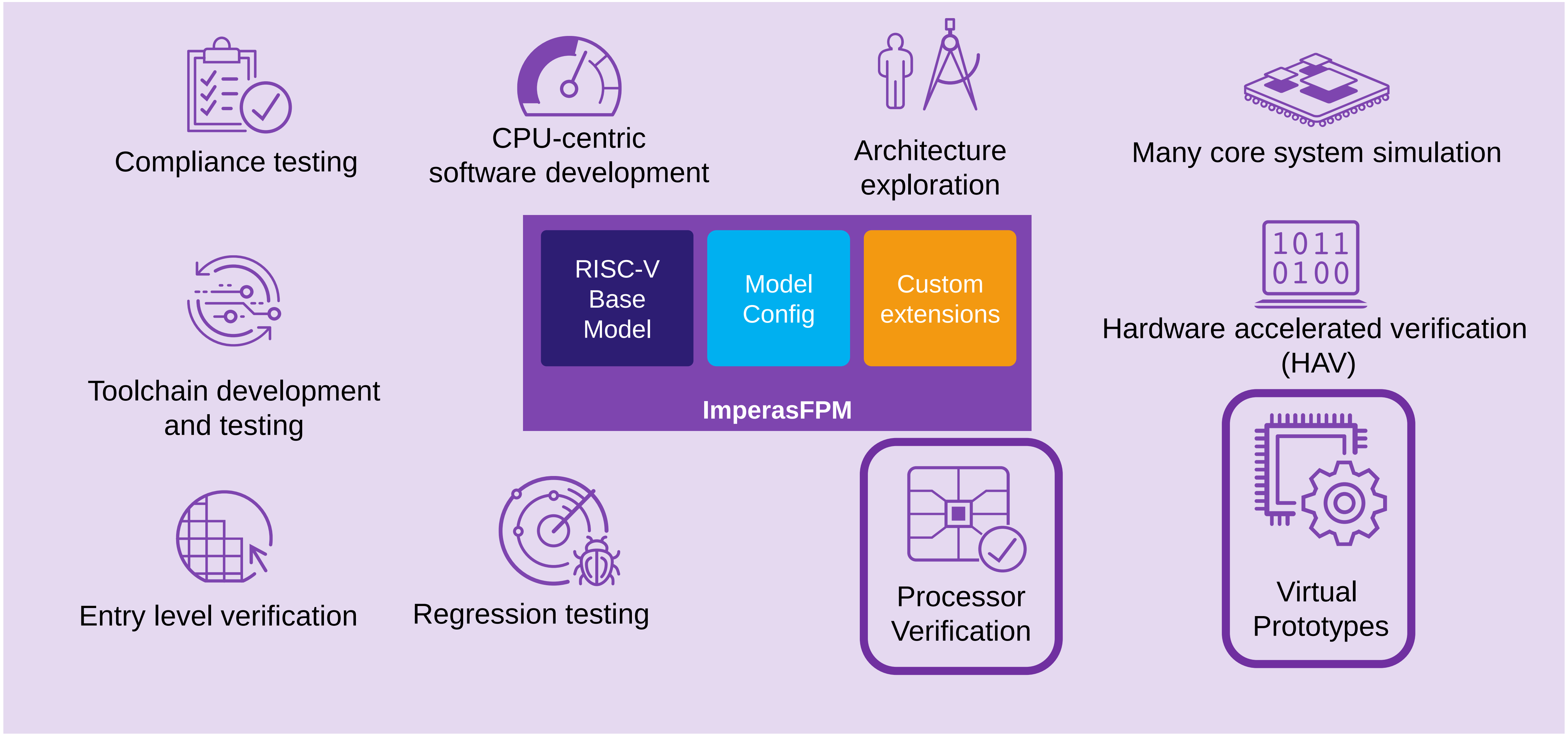
Entry level verification

Regression testing

Processor Verification

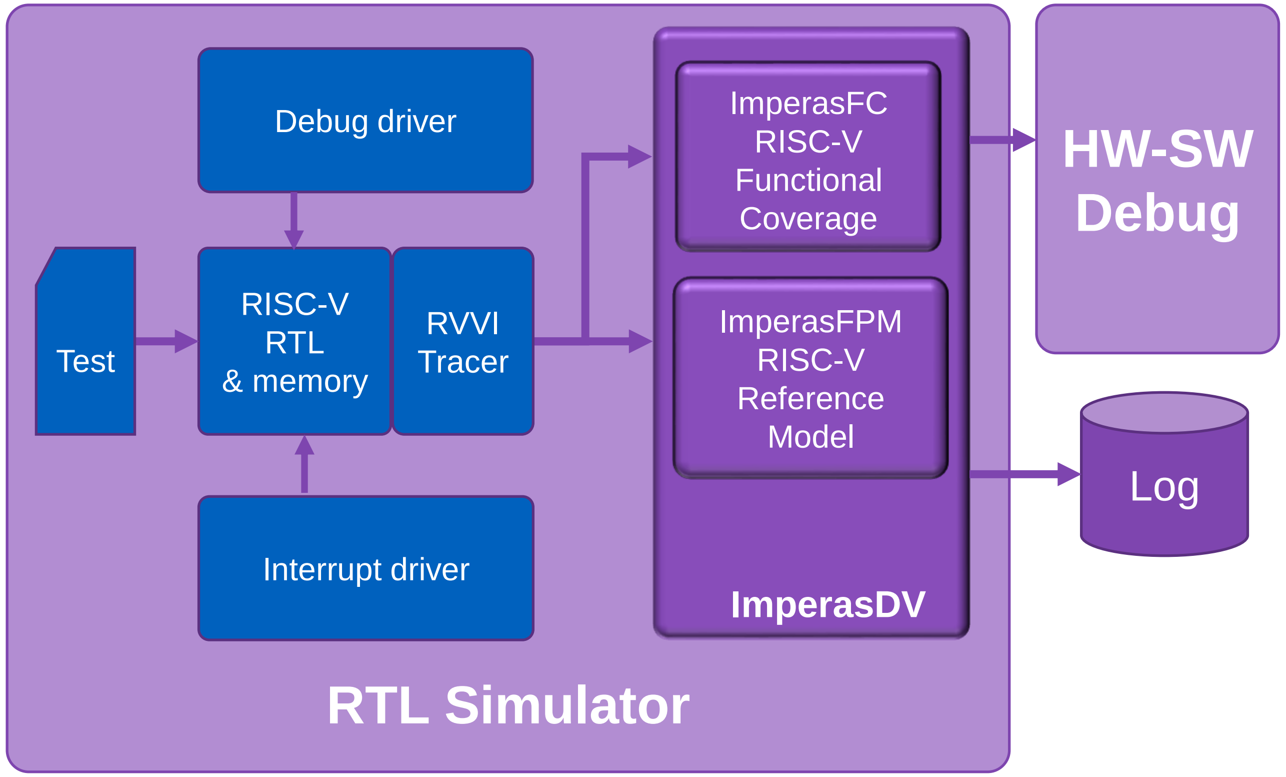
Virtual Prototypes

Using the same model for both hardware and software verification enables significant reduction in SoC bring up time



RISC-V Processor Verification Environment Enabled by RISC-V Reference Model

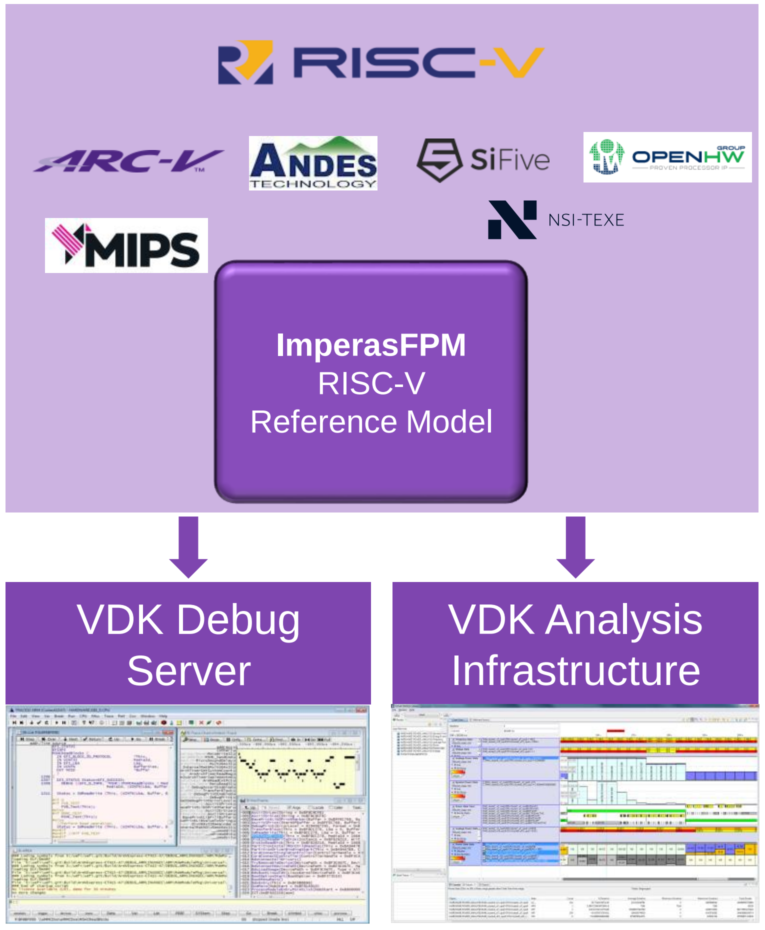
- ImperasFPM is the configurable, extendable RISC-V reference model
- Continuous comparison and checking of architectural state
- Detects synchronous and asynchronous bugs
- Complete RISC-V architectural Functional Coverage model



Virtual Prototyping for Early Software Development Enabled by RISC-V Reference Model

Virtual prototypes now a mainstream technology for embedded software development/test

- Shift left software development
- Comprehensive software development environment key for AI, automotive, other industries where software optimization and certification are required
- Support for …
  - all major RISC-V processor IP vendors
  - customers building their own RISC-V processor
- Enables Continuous Integration / Continuous Deployment (CI/CD) methodology



Virtual prototypes with ImperasFPM RISC-V models typically achieve 500-1,000 million instructions per second performance (e.g. boot Linux in ~6 sec on host PC)

Case Study: Adding CHACHA20 Encryption Instructions to Improve Performance of Character Stream Encoder

- Before adding custom instructions, character stream encoder software took ~1.3 billion instructions to execute
- Profiling showed “processLine” function took 21% of execution time
- Now add custom instructions to improve performance

- This instruction extension library will be used to develop four custom instructions
  - Each uses the same base behavior
  - Different rotation value
- When implementing new instructions, start with the decode table, including
  - The fixed fields defining the instruction class
  - The fields defining the source and result registers to be used
- In the RISC-V ISA these will be R-Type instructions in custom-1 decode space

Bits	Bit Value	Description
6 – 0	00 010 00	Custom-1 instruction class decode
11 – 7	xxxxx	Identify the result register
14 – 12	000 001 010 011 1xx	QR1 QR2 QR3 QR4 Undefined
19 – 15	xxxxx	Identify source register 1
24 – 20	xxxxx	Identify source register 2
31 – 25	0000000	Instruction decode

Instruction Decode Table

```
//
// Create the RISCV decode table
//
static vmidDecodeTableP createDecodeTable(void) {

vmidDecodeTableP table = vmidNewDecodeTable(RISCV_INSTR_BITS, RISCV_EIT_LAST);

// handle custom instruction
DECODE_ENTRY(0, CHACHA20QR1, "|0000000..........000.....0001011|");
DECODE_ENTRY(0, CHACHA20QR2, "|0000000..........001.....0001011|");
DECODE_ENTRY(0, CHACHA20QR3, "|0000000..........010.....0001011|");
DECODE_ENTRY(0, CHACHA20QR4, "|0000000..........011.....0001011|");

return table;

}
```

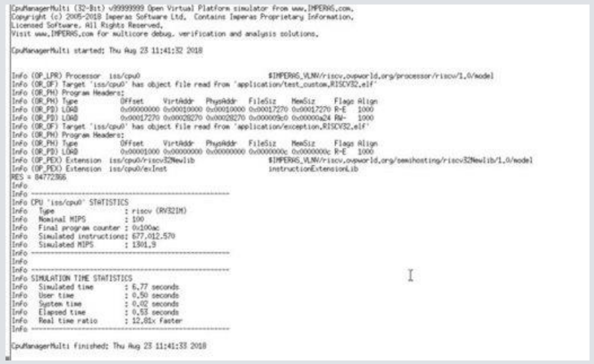
- This decode table is then constructed in the extension library constructor
- Stored in the vmiosObject structure

Instruction Behavior is Defined Using VMI Morph Time Functions

```
//
// Emit core implementing exchange instruction
//
Static void emitChaCha20(
          vmiProcessorP       processor,
          vmiosObjectP        object,
          Uns32               instruction,
          Uns32               rot1
)  {

          // extract instruction fields
          Uns32 rd  =         RD(instruction);
          Uns32 rs1 =         RS1(instruction);
          Uns32 rs2 =         RS2(instruction);

          vmiReg reg_rs1 = vmimtGetExtReg(processor, &object->rs1);
          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[rs12]);
          vmimtBinopRRR(32, vmi_XDR, reg_tmp, reg_rs1, reg_rs2, 0);
          vmimtBinopRC(32, vmi_ROL, reg_tmp, rot1, 0);

          vmimtSetR(processor, RISCV_REG_BITS, object->riscvRegs[rd], reg_tmp);
}
```

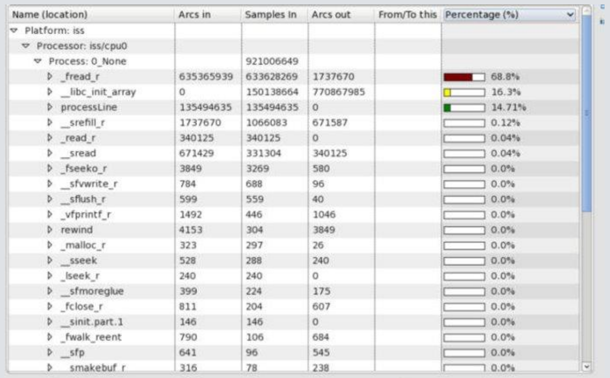
Adding Custom Instructions Reduces Number of Instructions Executed, Reduces Execution Bottleneck

Model -> API -> Simulator architecture allows tools to just work automatically



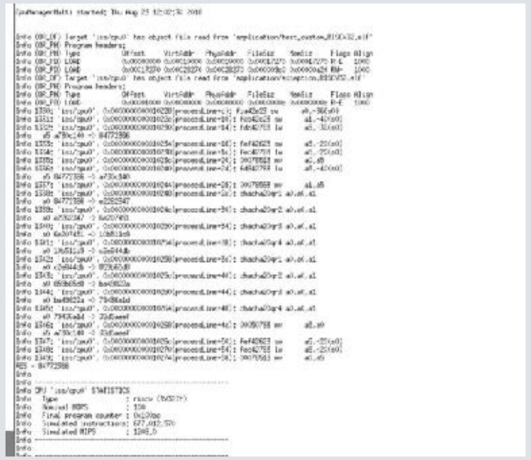
```
CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Thu Aug 23 11:41:32 2018

Info (OP_LPR) Processor  iss/cpu0                              $IMPERAS_VLNV/riscv.ovpworld.org/processor/riscv/1.0/model
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 (OP_PEX) Extension  iss/cpu0/riscv32Newlib                $IMPERAS_VLNV/riscv.ovpworld.org/semihosting/riscv32Newlib/1.0/model
Info (OP_PEX) Extension  iss/cpu0/exInst                       instructionExtensionLib
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: 677,012,570
Info   Simulated MIPS        : 1301.9
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 6.77 seconds
Info   User time             : 0.50 seconds
Info   System time           : 0.02 seconds
Info   Elapsed time          : 0.53 seconds
Info   Real time ratio       : 12.81x faster
Info ---------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:41:33 2018
```

Simulation performance increase from 1.1 to 1.3 billion instructions per second



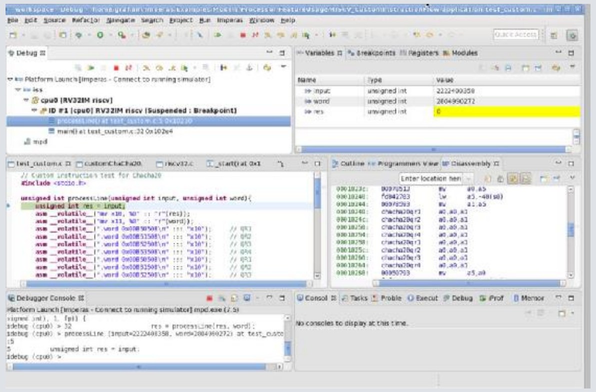
Name (location)	Arcs in	Samples In	Arcs out	From/To this	Percentage (%)
Platform: iss					
Processor: iss/cpu0					
Process: 0_None		921006649			
_fread_r	635365939	633628269	1737670		68.8%
__libc_init_array	0	150138664	770867985		16.3%
processLine	135494635	135494635	0		14.71%
__srefill_r	1737670	1066083	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%
_vfprintf_r	1492	446	1046		0.0%
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%
__sinit.part.1	146	146	0		0.0%
_fwalk_reent	790	106	684		0.0%
__sfp	641	96	545		0.0%
__smakebuf_r	316	78	238		0.0%

Key “processLine” function now takes <15% of execution time



```
CpuManagerMulti started: Thu Aug 23 12:02:32 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', 0x0000000000010230(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 -> [illegible]
Info 1339: 'iss/cpu0', 0x000000000001024c(processLine+30): chacha20qr2 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1340: 'iss/cpu0', 0x0000000000010250(processLine+34): chacha20qr3 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1341: 'iss/cpu0', 0x0000000000010254(processLine+38): chacha20qr4 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1342: 'iss/cpu0', 0x0000000000010258(processLine+3c): chacha20qr1 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1343: 'iss/cpu0', 0x000000000001025c(processLine+40): chacha20qr2 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1344: 'iss/cpu0', 0x0000000000010260(processLine+44): chacha20qr3 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1345: 'iss/cpu0', 0x0000000000010264(processLine+48): chacha20qr4 a0,a0,a1
Info   a0 [illegible] -> [illegible]
Info 1346: 'iss/cpu0', 0x0000000000010268(processLine+4c): 00050793 mv     a5,a0
Info   a5 a730c140 -> [illegible]
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 (RV32IM)
Info   Nominal MIPS          : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 677,012,570
Info   Simulated MIPS        : 1208.5
Info ---------------------------------------------------
Info
Info ---------------------------------------------------
```

CHACHA20 instructions in instruction trace



```
// 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 0x0080050B\n" ::: "x10");    // QR1
    asm __volatile__(".word 0x0080150B\n" ::: "x10");    // QR2
    asm __volatile__(".word 0x0080250B\n" ::: "x10");    // QR3
    asm __volatile__(".word 0x0080350B\n" ::: "x10");    // QR4
    asm __volatile__(".word 0x0080050B\n" ::: "x10");    // QR1
    asm __volatile__(".word 0x0080150B\n" ::: "x10");    // QR2
    asm __volatile__(".word 0x0080250B\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
```

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

```
Platform Launch [Imperas - Connect to running simulator] mpd.exe (2.5)
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) >
```

No consoles to display at this time.

CHACHA20 instructions in debugger disassembly view

OVP APIs Enable High Quality RISC-V Models

- Use cases and requirements for RISC-V processor models
- ImperasFPM fast processor model architecture
- OVP APIs
- Case studies
- **Summary**

OVP APIs Enable High Quality RISC-V Processor Models

- RISC-V processor models are at the heart of RISC-V processor development
- An API-based processor model architecture can more easily meets all the use case requirements
- An API-based processor model architecture can provide multiple benefits including quality, flexibility, extensibility, high performance, ease of documentation
- The OVP APIs used in the ImperasFPM RISC-V processor models provide all these benefits, while reducing the resources required for processor development and maintenance
- These benefits have been proven with results from hundreds of projects, including ~30 tape outs enabled by processor verification using ImperasFPM golden reference models

THANK YOU

Our Technology
Your Innovation™

Larry Lapides
Larry.Lapides@synopsys.com