SYNOPSYS® 新思

Extending RISC-V into VLIW / SIMD Architectures for Application-Specific Workloads

毛海雪, Staff Applications Engineer

2025-07-18

Legal Disclosure

CONFIDENTIAL INFORMATION

The information contained in this presentation is the confidential and proprietary information of Synopsys. You are not permitted to disseminate or use any of the information provided to you without the prior written consent of an authorized officer of Synopsys.

IMPORTANT NOTICE

In the event information in this presentation reflects Synopsys' future plans, including but not limited to Synopsys' financial targets, expectations and objectives; strategies related to our products, technology and services; business and market outlook, business opportunities and strategies; and more, such information is based on current estimates, provided as of the date of this presentation and are subject to change. Synopsys undertakes no duty to, and does not intend to, update any forward-looking statement, whether as a result of new information, future events or otherwise, unless required by law.

Outline

1. ISA extensibility in the RISC-V philosophy
2. ISA extensibility for high computational performance
3. Extending RISC-V into VLIW / SIMD architectures
4. ASIP Designer: a tool-suite to design extended RISC-V architectures

ISA Extensibility: a Key Ingredient of the RISC-V Philosophy

Standard Extensions

- Proposed and developed in the RISC-V community for new areas of common interest
- Collaborative review and ratification process

1. Inception
2. Plan
3. Develop
4. Freeze
5. Public Review
6. Ratified

I	Integer
E	Reduced Integer

M	Integer Multiply & Division
A	Atomics
F	Single-Precision Floating Point
D	Double-Precision Floating Point
G	General
Q	Quad-Precision Floating Point
C	16-bit Compressed Instructions
B	B Extension
P	Packed SIMD
V	Vector
H	Hypervisor
Z…	Additional Unprivileged Extensions
Ss…	Supervisor-Level Extensions
Sh…	Hypervisor-Level Extensions
Sm…	Machine-Level Extensions

Zicsr	Control & Status Register Instructions
Zifencei	Instruction-Fetch Fence.
Zfinx	Floating Point in Integer Registers
Zaamo	Standard Atomic Memory Operations
Zabha	Byte and Halfword Atomic Memory Operations
Zacas	Atomic Compare & Swap Instructions
Zve…	Floating-Point Vector Instructions
Zfh	Half-Precision Floating Point Instructions
Zb…	Bit Manipulation
Zt	Compressed Instructions
Zpn	Packed SIMD Instructions
Zvb… Zvk…	Vector Cryptography Instructions
…	

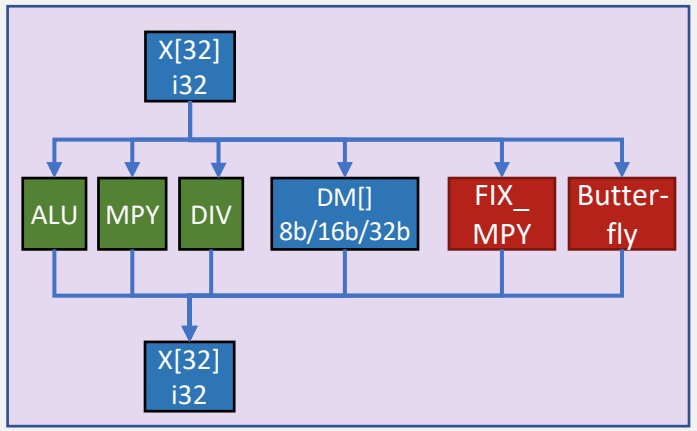
ISA Extensibility: a Key Ingredient of the RISC-V Philosophy

Custom Extensions

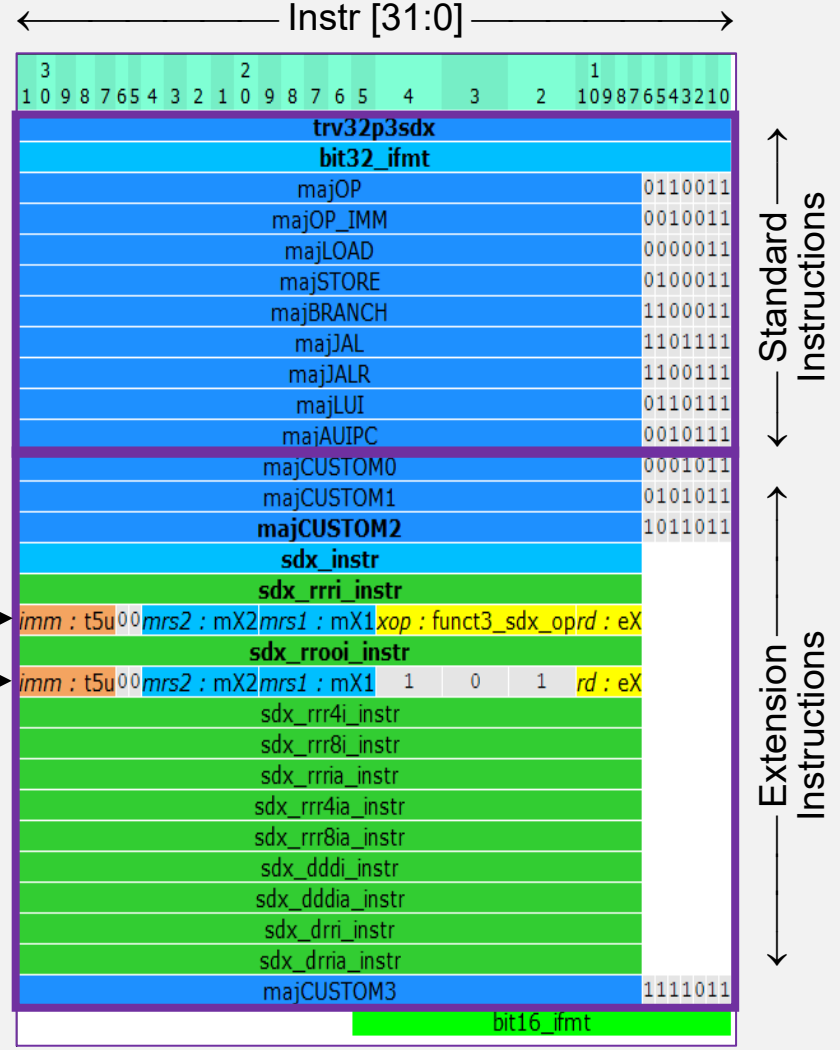
X…

- Developed independently by design teams
- Goal
  - Efficient execution of application-specific workloads
    - Accelerated execution
    - Power reduction
- How?
  - Add new functional units
    - E.g.: fusing short instruction sequences into a single operation
    - Typically operating on standard register file X
  - Encode new instructions in custom opcode spaces
    - 4 spaces of 25 bits

Example: Custom extensions for FFT



- Custom datatypes
  - Fixed-point
  - Complex (16b Re, 16b Im)
- Custom functional units
  - Complex fixed-point multiplier
  - FFT butterfly
- Custom instructions
  - Complex fixed-point multiplication & scaling
  - Absolute value
  - FFT butterfly

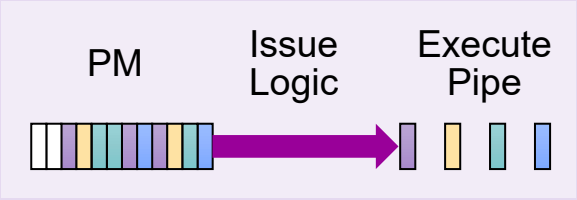


Do Custom Extensions Deliver High Computational Performance?

Custom Extensions X…

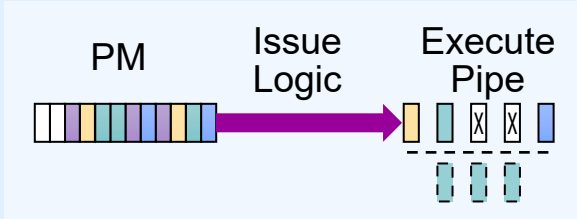
- Acceleration through fused operations

- Von Neumann or Single-Harvard
- No instruction-level parallelism (ILP)
- No or little data-level parallelism (DLP)

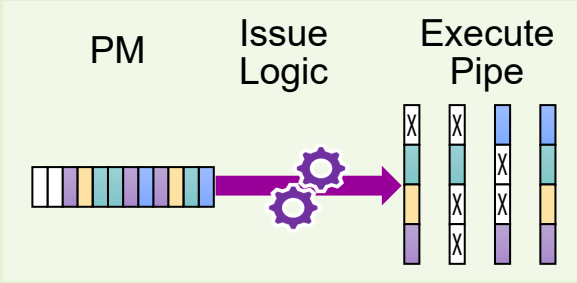


Parallel Processing Approaches in the RISC-V Community

- ILP: RISC-V with custom co-processor
  - Standard interface, e.g. CV-X-IF
  - While co-processor is executing, RISC-V core is mostly waiting

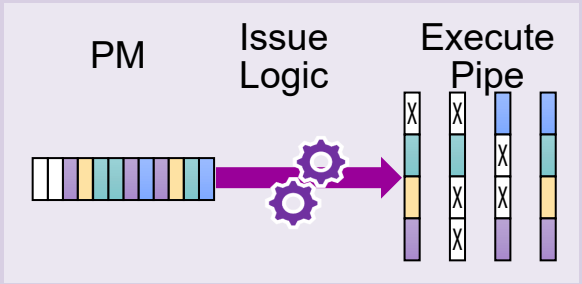
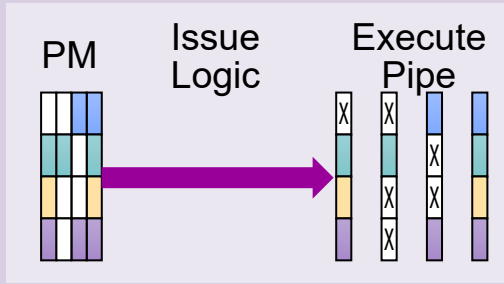


- ILP: Microarchitecture with dynamic multi-issuing (superscalar)
  - Parallel functional units
  - Parallelism is extracted at run-time
    - Seldomly exceeds 3 parallel issues
  - Complex hardware
    - Data/control flow dependency checks, resource availability checks, out-of-order execution, speculative issuing, register renaming, branch prediction…



- DLP: RISC-V vector extension (RVV)
  - General-purpose vector instructions, operating on vector register file
  - Complex hardware:
    - Over 400 instructions
    - Support for vector-length agnostic code (vector-length depends on implementation)

Microarchitectural Options for ILP

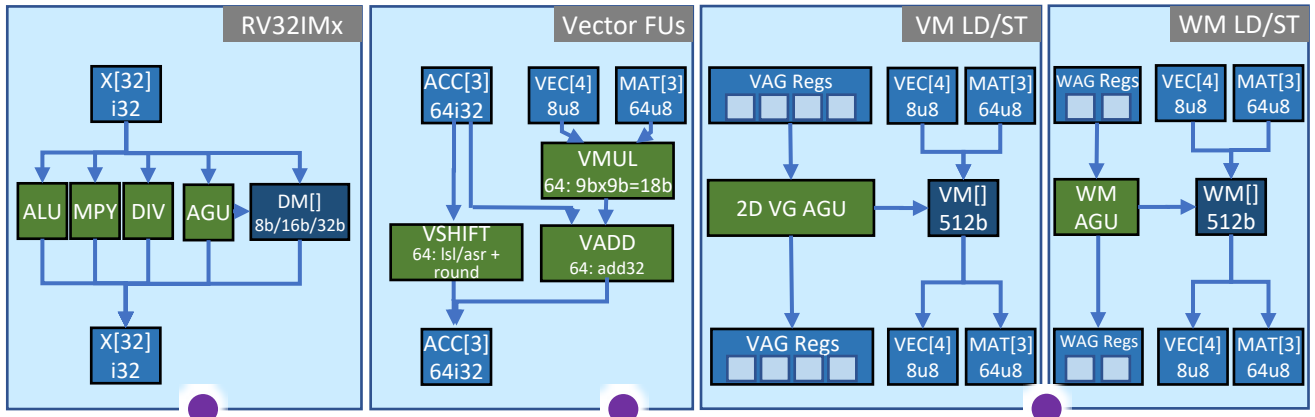
	Dynamic Multi-Issue	Static Multi-Issue
		
Use Cases	• General-purpose computing	• DSP, application-specific processors
Instruction Stream	• Single instruction stream	• VLIW or encoded instruction formats • Variable-length instructions
Parallelism Extraction	• Extracted at run-time − Complex issue logic − Seldomly exceeds 3 parallel issues − Hard to scale no. of issues	• Extracted at compile-time − Parallelizing compiler technology − 10+ parallel issues not uncommon − Easy to scale no. of issues
Control Flow Features	• Speculative issue • Dynamic branch prediction	• Delay slots • Zero-overhead loops − Software pipelining (by the compiler)

For high-performance, application-specific workloads, we advocate extending RISC-V into Static Multi-Issue (VLIW) Architectures

Extending RISC-V into VLIW / SIMD Architectures

Example: VLIW / SIMD for Acceleration of MobileNet v3

Extended datapath



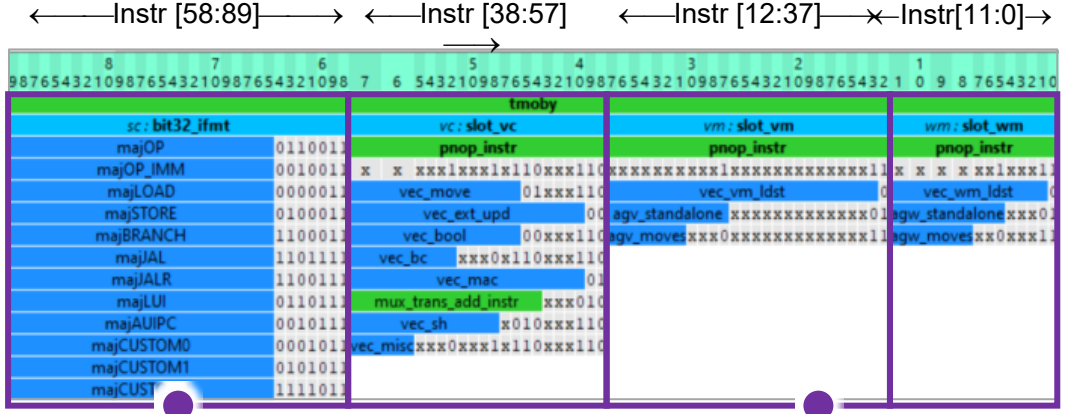
- Baseline RISC-V datapath
- Custom extensions:
  - Load/store with post-modification on AGU
  - HW do-loop

- Vector arithmetic unit
  - SIMD64: 8b x 8 lanes
  - Distributed vector register-files
  - Vector mul-acc: (9b x 9b → 32) x 8 lanes

- 2 vector load/store units
  - SIMD64: 8b x 8 lanes
  - VM = features WM = weights
  - Vector addressing with special 2D and 1D post-modification modes

Instruction encoding

- 4-slot VLIW
- 90-bit instruction word



- 32-bit scalar slot
  - Baseline RISC-V (RV32IM)
  - AGU & HW do-loop encoded in custom opcode space

- 3 vector slots
  - Vector arithmetic
  - 2 vector load/store

Speedup: 360x

RISC-V ecosystem compatibility

- **Run existing RISC-V binary code**
- **Communicate with existing RISC-V peripherals**

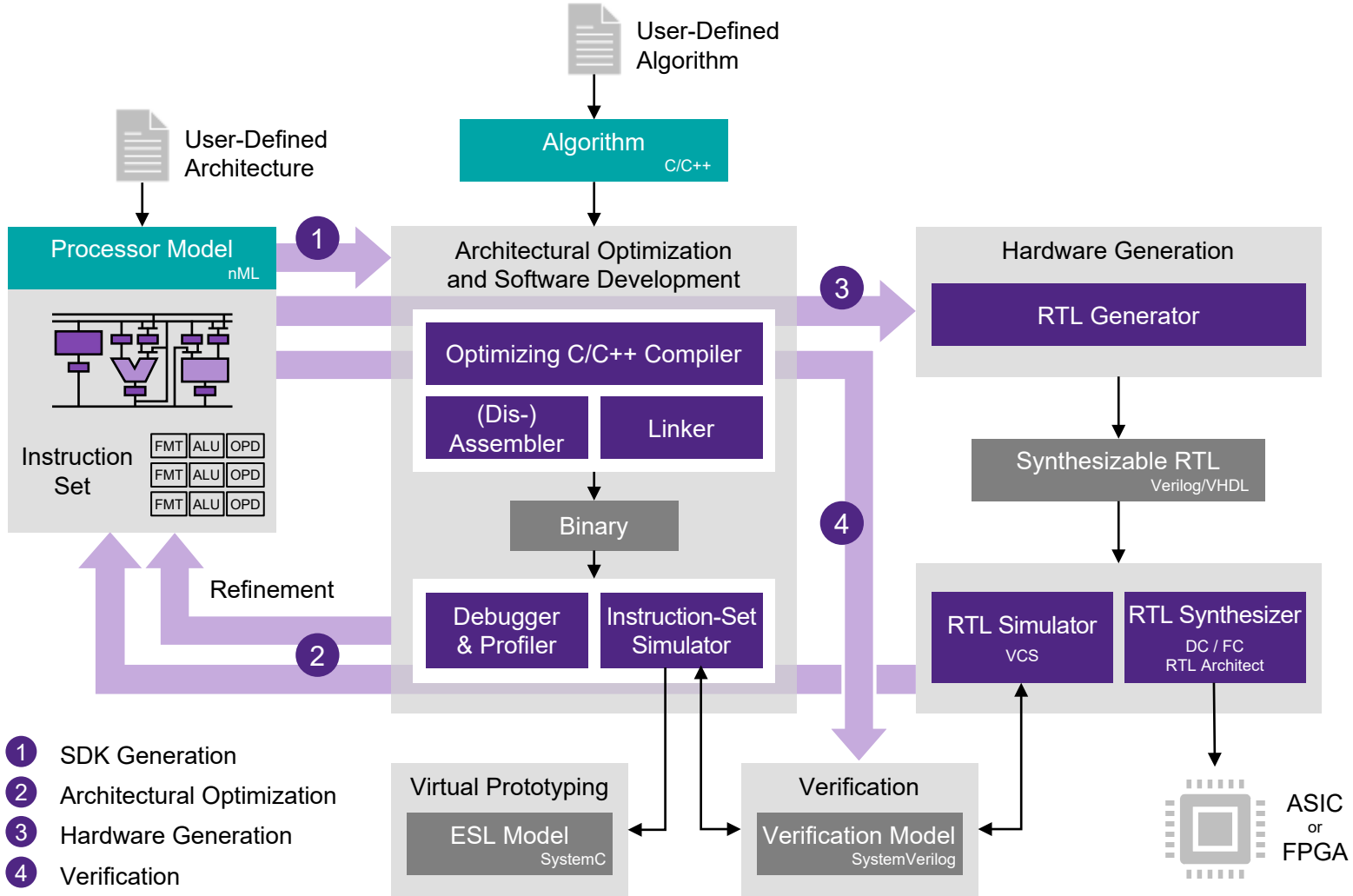
Acceleration of application-specific workloads

Extending RISC-V into VLIW / SIMD Architectures

How To Design It?

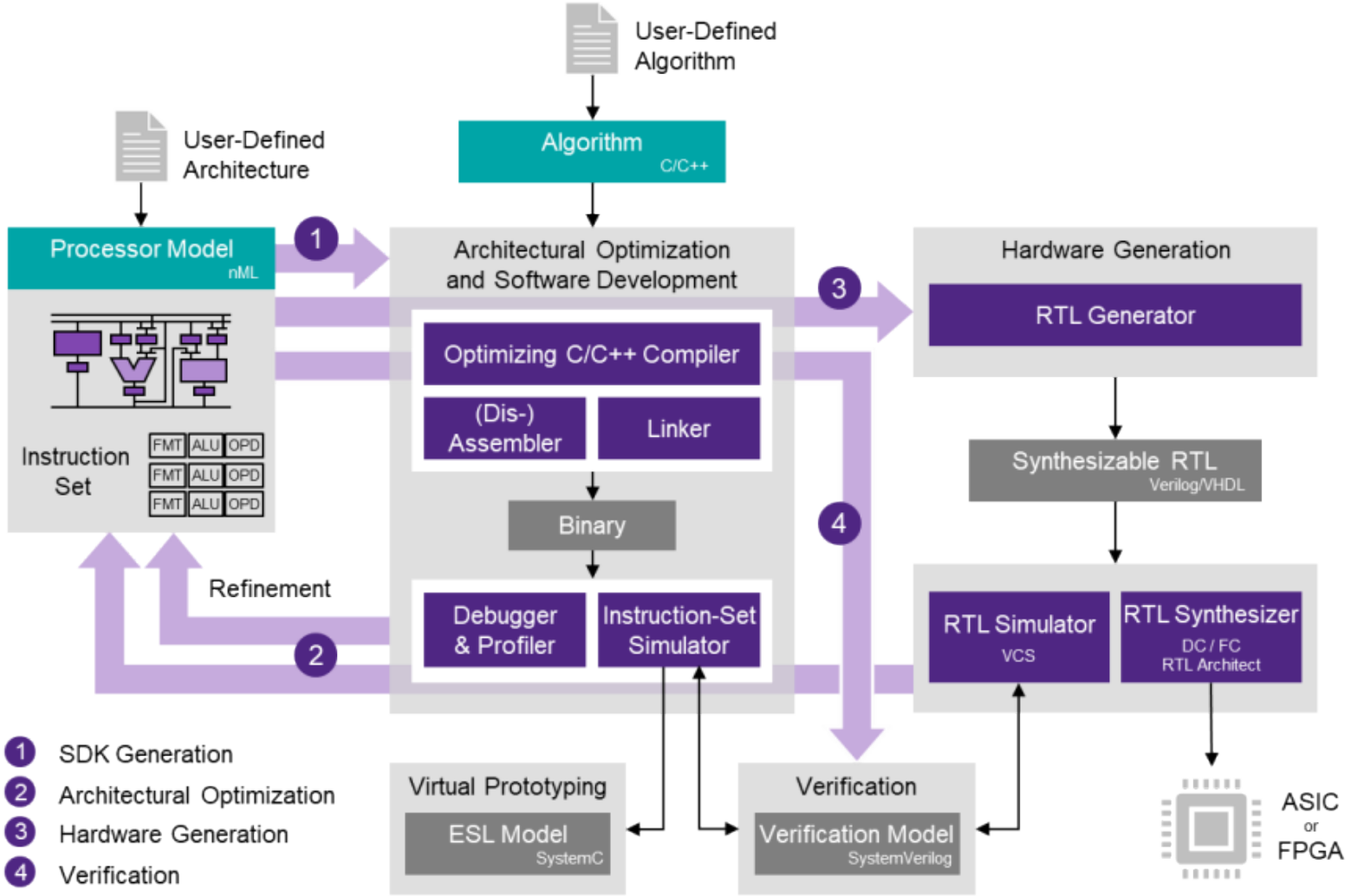
ASIP Designer™

- Leading tool-suite to design and program Application-Specific Processors (ASIPs)
- Modeling of instruction-set and microarchitecture: nML language
- Automatic generation of software development kit, including an efficient C/C++ compiler
- Algorithm-driven architectural exploration: **“Compiler-in-the-Loop”**
- Automatic generation of synthesizable RTL **“Synthesis-in-the-Loop”**
- Design verification



Extending RISC-V into VLIW / SIMD Architectures

How To Design It?

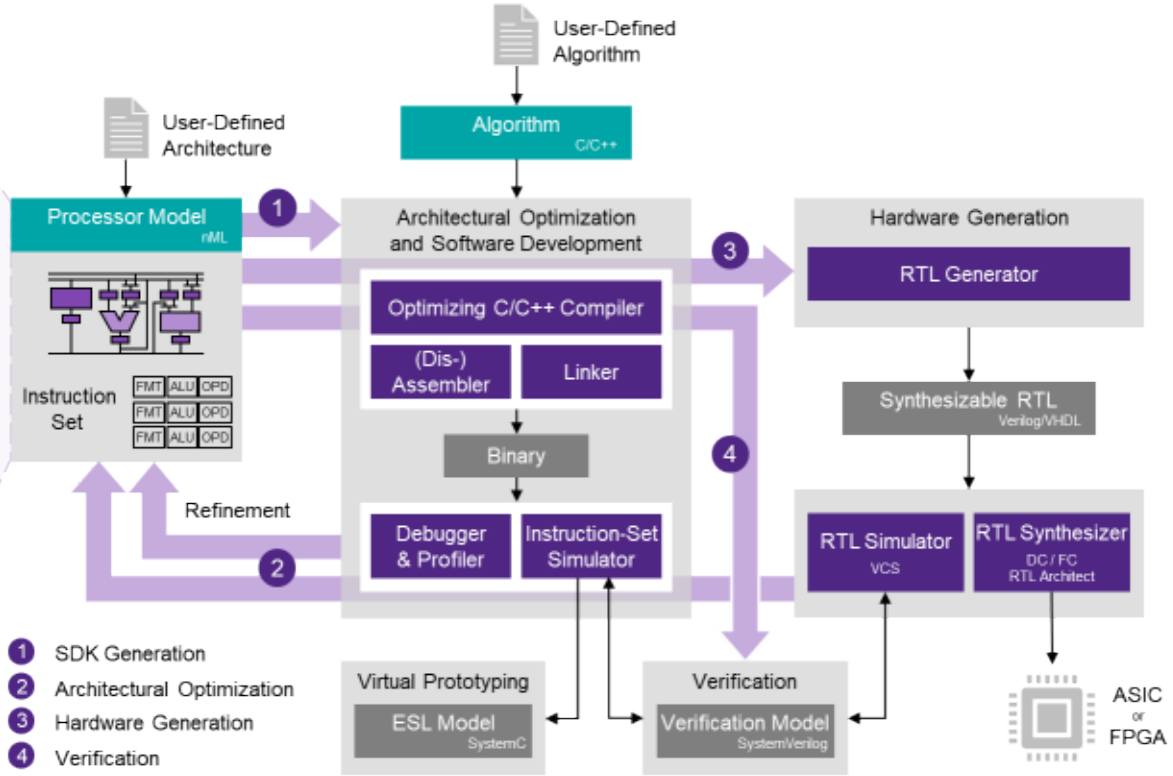


Extending RISC-V into VLIW / SIMD Architectures

How To Design?

```
start trv32p5x;
opn trv32p5x(bit32_ifmt | bit16_ifmt);
opn bit32_ifmt(majOP | majOP_IMM | majLOAD | ... | majCUSTOM3);
opn majOP(alu_rrr_ar_instr | mpy_rrr_instr | div_instr);
opn alu_rrr_ar_instr(op: majOP_fn10, rd: eX, rs1: eX, rs2: eX){
  action{
  stage ID:
     pidX1 = r1 = X[rs1];
     pidX2 = r2 = X[rs2];
  stage EX:
     aluA = pidX1;
     aluB = pidX2;
     switch(op){
        case add:  aluR = add (aluA,aluB) @alu;
        case sub:  aluR = sub (aluA,aluB) @alu;
        case slt:  aluR = slt (aluA,aluB) @alu;
        case sltu: aluR = sltu(aluA,aluB) @alu;
        case xor:  aluR = bxor(aluA,aluB) @alu;
        ...
        case sra:  aluR = sra (aluA,aluB) @alu;
     }
  stage EX:
     pexX1 = texX1 = aluR;
  stage ME:
     pmeX1 = tmeX1 = pexX1;
  stage WB:
    if (rd: x0) w1_dead = w1 = pmeX1;
    else X[rd] = w1 = pmeX1;
  }
  syntax : "neg "  rd "," rs2 op<<sub>>  rs1<<x0>>
         | "snez " rd "," rs2 op<<sltu>> rs1<<x0>>
         | "sltz " rd "," rs1 op<<slt>>  rs2<<x0>>
         | "sgtz " rd "," rs2 op<<slt>>  rs1<<x0>>
         | op " "  rd "," rs1 "," PADOP2 rs2;
  image  : op[9..3]::rs2::rs1::op[2..0]::rd, class(alu_rrr);
}
...
```

- nML models of RISC-V are shipped with ASIP Designer
  - Variants: integer/float, 32/64-bit, 3/5 pipe stages, HW do-loop, AGU
- Designers can extend these nML models as desired



Conclusions

- ISA extensibility is a key ingredient of the RISC-V philosophy
- For high computational performance, the RISC-V community has been exploring:
  - Scalar RISC-V with co-processors
  - Micro-architectures with dynamic multi-issuing
  - Vector extension standard (RVV)
- The DSP community has proven the computational performance of static multi-issuing
  - VLIW / SIMD
- We advocate extending RISC-V into static multi-issue architectures
  - High performance, while preserving compatibility with the RISC-V ecosystem
- ASIP Designer is the ideal tool-suite to design such architectures

SYNOPSYS® 新思

Thank you