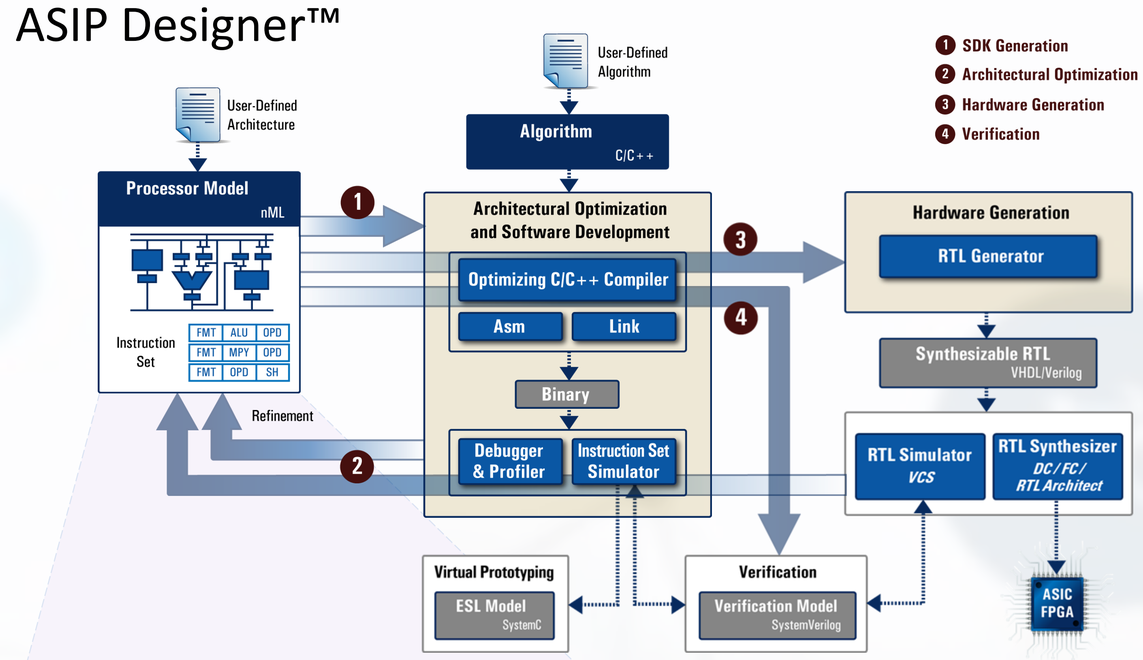
Taking the Risk out of Optimizing Your Own RISC-V Architecture Design

Gert Goossens
gert.goossens@synopsys.com

Patrick Verbist
patrick.verbist@synopsys.com

Dominik Auras
dominik.auras@synopsys.com

ASIP Designer™




```
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) wl_dead = wl = pmeX1;
    else X[rd] = wl = 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);
}
...
```

- Industry-leading tool to design your own Application-Specific Instruction-set Processor (ASIP)
- Language-based description of ISA and microarchitecture: nML
- Single processor model ensures that SDK and RTL are in sync
- Architectural exploration with Compiler-in-the-Loop™ & Synthesis-in-the-Loop™
- Licensed as a tool (not IP): product differentiation, no royalties
- Full interoperability with other Synopsys EDA tools

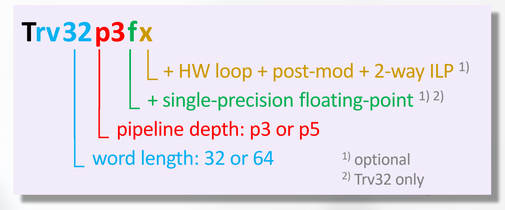
RISC-V Extensibility

- ISA customization and extensibility are drivers for the growing adoption of RISC-V
- This results in ASIPs with a RISC-V baseline
- Preserve RISC-V compatibility
  - Execute SW code and libraries
  - Reuse HW peripherals

  } designed for gen.-purpose RISC-V
- Challenges
  - Which extensions are best for the target application domain?
  - How to obtain a high-quality SW Development Kit (SDK), including an optimizing compiler?
  - How to obtain a reliable RTL implementation with excellent PPA?
  - How to verify the design?

Trv (RISC-V) Models

Shipped with ASIP Designer



Integer models: Trv<mm>p<n>

	32-bit datapath	64-bit datapath
3-stage pipe	Trv32p3 Trv32p3x	Trv64p3 Trv64p3x
5-stage pipe	Trv32p5 Trv32p5x	Trv64p5 Trv64p5x

- ISA: RV64IM, RV32IM
  - Integer and multiply instructions
- Micro architecture
  - Protected pipeline, 3 or 5 stages
  - Hardware multiplier
  - Iterative divider
- Optional extensions: Trv<mm>p<n>x
  - Two-way static ILP
  - Zero overhead hardware loops
  - Load/store with post-modify addressing

Floating-point models: Trv32p<n>f

	32-bit datapath
3-stage pipe	Trv32p3f Trv32p3fx
5-stage pipe	Trv32p5f Trv32p5fx

- ISA: RV32IMZfinx
  - Integer and multiply instructions
  - Single precision float instructions
- Micro architecture
  - Protected pipeline, 3 or 5 stages
  - FPU models based on HardFloat [Hauser]
  - Iterative divider and square-root units
- Optional extensions: Trv<mm>p<n> fx
  - Two-way static ILP
  - Zero overhead hardware loops
  - Load/store with post-modify addressing

Designing RISC-V Extensions

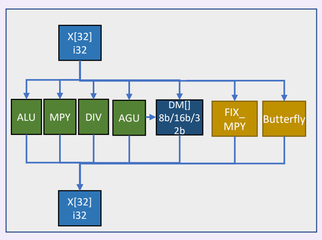
- nML models of Trv family are shipped with ASIP Designer tools
- Designers can extend these nML models as desired
- Leverage Compiler-in-the-Loop & Synthesis-in-the-Loop methodologies

- Option 1: Simple Datapath eXtensions (SDX)
  - Acceleration through specialization of scalar RISC-V core
  - Trv32p3sdx is an nML model of RISC-V with predefined extension stubs in the custom-2 opcode space
  - User adds behavior of the stubs in bit-accurate C code

- Option 2: Large-scale Trv extensions
  - Acceleration through instruction-level parallelism, SIMD, and/or specialization
  - By direct editing of any Trv model
  - Typically results in VLIW architecture with a RISC-V scalar issue slot

Example: FFT

- SDX instructions accelerating
  - Complex fixed-point multiply & scale — ***sdx1*** `rd,rs1,rs2`
  - ABS(x) function — ***sdx2*** `rd,rs1,rs2(x0)`
  - FFT Butterfly — ***sdx5*** `rd,rs1,rs1`
- Specialization:
  - Fractional data types
  - Complex numbers: 16/16 → 32-bit register

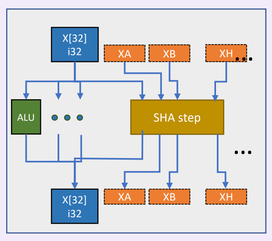


- Speedup: 3.8x
- Area increase: 31%

Example: SHA256

- Compute a hash of a message, using bitwise AND/OR/XOR, shift, add
- Custom instruction for complex hash
- Hash function has 8 state variables: use SDX' instruction extension stub supporting 8 additional register reads and writes

sdx7 `rd,rs1,rs2, x24,x25,…,x32`

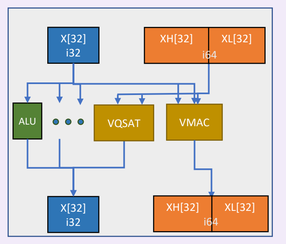


- Speedup: 3.7x
- Area increase: 16%

Example: Keyword Spotting

- Small-sized Neural Network (3.3M MACs)
- SDX architecture feature: packed SIMD
  - 32-bit register contains vector of 4x 8-bit values
  - Use of register pairs, enabling 64-bit access

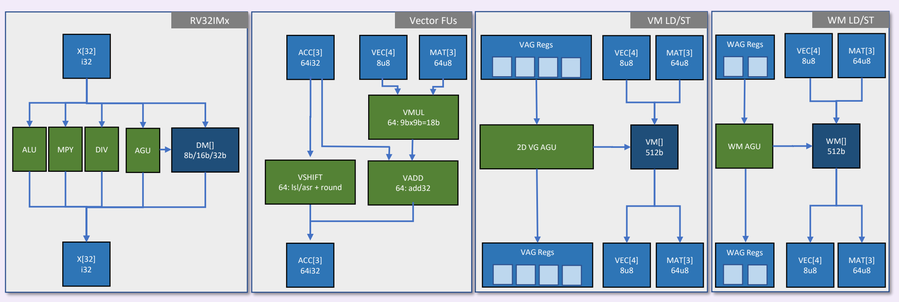
sdx4a_dr `dd,rs1,rs2,mode // vmac`
sdx0_dd `rd,ds1,ds2 // vqsat`

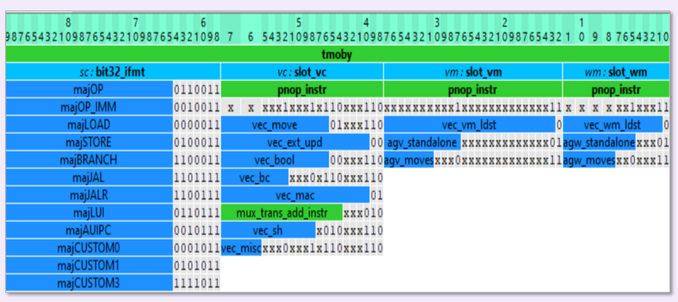


- Speedup: 12.6x
- Area increase: 16%

Example: "Tmoby" ASIP for acceleration of MobileNet v3

- Acceleration of SW kernels
  Conv_2D (pointwise), Depthwise_Conv_2D, Add, Average_Pool_2D, Softmax
- VLIW extension of Trv32p3: 90-bit instruction word, 4-way ILP
  - Scalar slot: Trv32p3
  - Vector arithmetic slot: SIMD64, MAC 8x8→32
  - 2 vector load/store slots: VM = features, WM = weights Vector addressing





- Speedup: 360x

Take-Aways

- Designing your own RISC-V architecture with application-specific extensions yields product differentiation and superior PPA, while maintaining flexibility and eco-system compatibility
- ASIP Designer is the industry-leading processor design tool, taking the risk out of your RISC-V design optimization

More info

- www.synopsys.com/asip
- asip_info@synopsys.com
- Contact authors: see above