SYNOPSYS®
Silicon to Software™

Flexible RISC-V Design for Deeply Embedded and Functional Safety Applications, using ASIP Designer™

Wei (Rick) Wang, Manager Field Applications Engineers, Processor Solutions, Synopsys China
Patrick Verbist, Product Marketing Manager, ASIP Tools, Synopsys Belgium

August 24-26, 2022

Synopsys ASIP Designer RISC-V Design Solutions

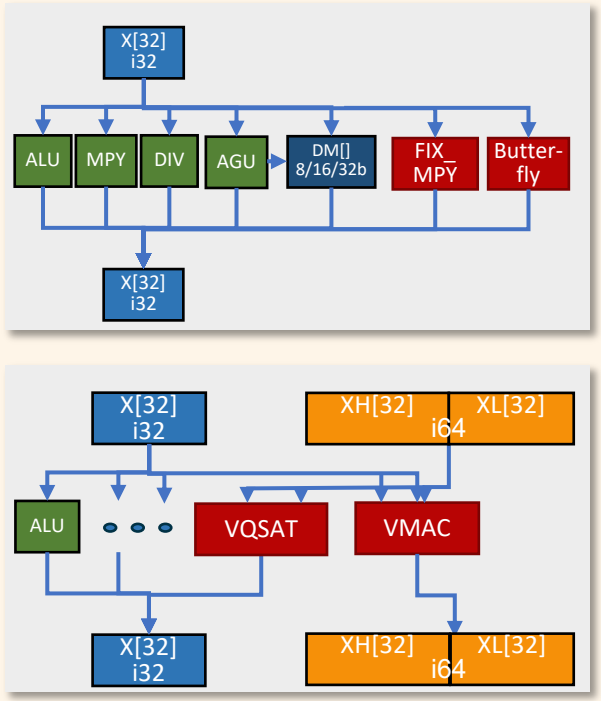
Market focus
Deeply embedded applications

- Customer designs using Synopsys' RISC-V based solutions in:
  - 5G base station
  - 5G mobile
  - Wireline
  - Security
  - Automotive
  - Mobile sensor processing



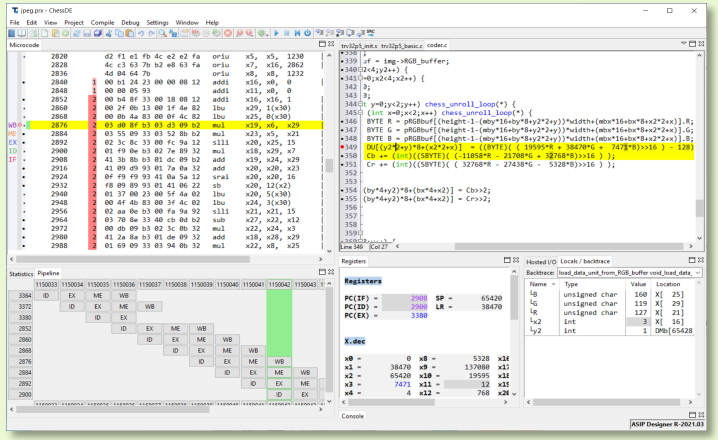

Technology focus
Flexibility

- Family of RISC-V core models, with strong support for designing ISA extensions



Technology focus
Development infrastructure

- Complete software development kit for all RISC-V models, including ISA extensions
- Tool support for exploration of ISA extensions and RTL implementation
- Interoperability with Synopsys tools and platforms for prototyping and physical design



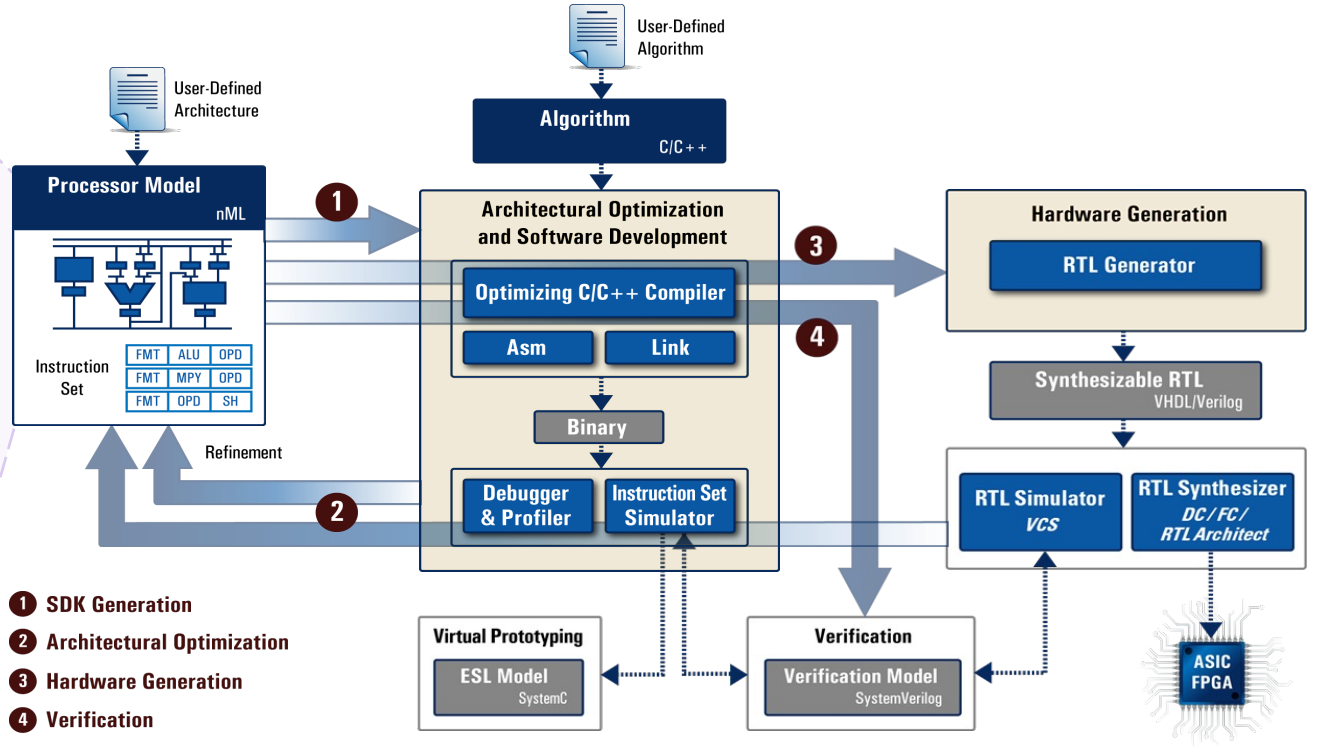
Industry Proven

Customer Designs Using RISC-V Based Design Solutions

Customer	Product Name	Application Domains	ASIP Architecture	Status (May-2022)
Undisclosed		Controller in wireline communication chip set	• RISC-V ISA	Silicon Released products
Riken	MDGrape-4A	Supercomputer	• Multicore configuration • RISC-V ISA plus application-specific SIMD	Silicon Released product
MIT-LincolnLabs		Security	• RISC-V ISA extended with security / hashing specific instructions	Silicon
Undisclosed		Gesture recognition in mobile	• Twelve-slot VLIW with RISC-V based scalar slots	In design
Undisclosed		Baseband inner-modem functions in 5G handset	• Design started from RISC-V ISA, extended and modified to custom SIMD / VLIW	Virtual prototype
Undisclosed		Baseband inner-modem functions in 5G handset	• RISC-V ISA extended, application-specific SIMD, dynamic issue	In design
Undisclosed		Matrix acceleration in 5G base station	• RISC-V ISA extended, floating-point SIMD	In design
Undisclosed		FFT acceleration in 5G base station	• RISC-V ISA extended, application-specific SIMD	In design
Undisclosed		Packet processing in 5G basestation	• RISC-V ISA extended, application-specific SIMD	In design
Undisclosed		Modulation in 5G basestation	• RISC-V ISA extended, application-specific SIMD	In design
Undisclosed		Scheduler in 5G basestation	• RISC-V extended	In design
Undisclosed		Automotive	• RISC-V ISA	Silicon
Undisclosed		Automotive	• RISC-V ISA extended	Silicon
NSI-TEXE (Denso)	DFP	Automotive	• Multicore configuration • RISC-V ISA, application-specific SIMD, multithreading	In design
Undisclosed		Security	• RISC-V ISA scaled down, application-specific SIMD	In design
Undisclosed		Deep learning	• Started from RISC-V ISA, extended and modified to custom SIMD / VLIW	Silicon Released product
Undisclosed		Unknown	• RISC-V extended with 2-slot ILP and F instructions	In design

RISC-V Design Solutions Use ASIP Designer Tool-Suite

```
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);
}
...
```



- Industry-leading tool to design your own processor
- Language-based description of ISA and microarchitecture
- Single processor model ensures that SDK and RTL are in sync
- Licensed as a tool (not IP), no royalties

Synopsys ASIP Designer RISC-V Offering

“Trv” Processor Model Family

Trv (RISC-V ISA) Models

Overview

Trv32p3xf

- word length: 32 or 64
- pipeline depth: p3 or p5
- + HW loop + post-mod + 2-way ILP [1)]
- + single-precision floating-point [1) 2)]

[1)] optional
[2)] Trv32 only

Integer models

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

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

Floating-point models

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

- Supported ISA: RV32IMF
  - Integer instructions
  - Multiply instructions
  - Float (single precision) instructions
- Micro architecture
  - FPU models based on HardFloat [Hauser]
  - Iterative divider and square-root units
- Optional extensions: Trv32p<n>fx
  - Cf. integer models

Trv (RISC-V ISA) Models

"x" Variants: Static Dual-Issue

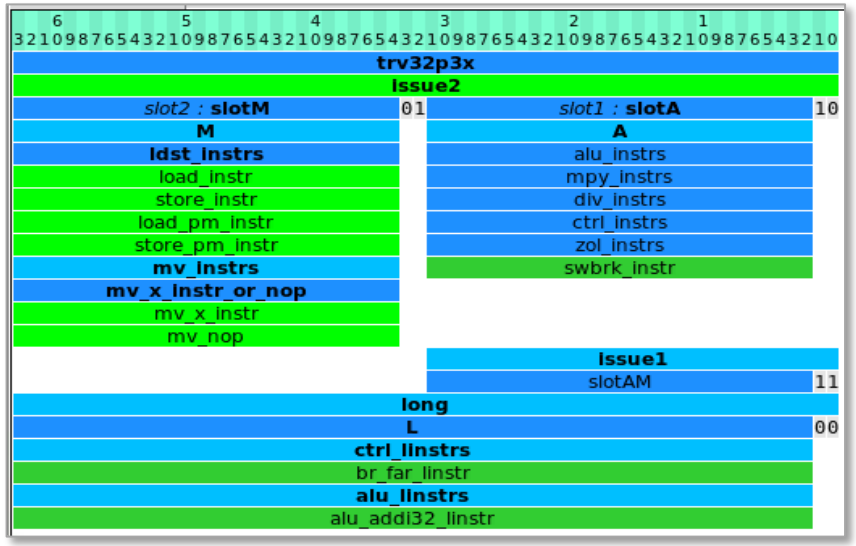
Substantial Performance Increase

Trv32p3xf

- word length: 32 or 64
- pipeline depth: p3 or p5
- + HW loop + post-mod + 2-way ILP [1)]
- + single-precision floating-point [1) 2)]

[1)] optional
[2)] Trv32 only

- Two-way static instruction-level parallelism
  - Arithmetic | Load/Store
  - 64-bit encoded
  - Potentially better utilization of arithmetic and load/store resources
  - Requires additional register ports
- Long immediate instructions
- Encoded using the format bits of the C class
  - "10" Dual-issue instructions
  - "01" Parallel part of dual-issue instructions
  - "11" Single-issue instructions
  - "00" Long immediate instructions
- Zero overhead hardware loops
- Load/stores with post-modify address modes



CoreMark matrix multiply loop ►

```
6588    36  00 00 00 33              nop
6592   324  80 60 08 31 00 04 87 92  mv    x15, x9          | mv   x16, x6
6600   324  80 00 00 31 00 00 07 36  lui   x14, 0           | nop
6608  2916  00 07 98 81 00 f5 07 b2  add   x15, x10, x15   | lh   x17, 0(x15)
6616  2916  00 28 19 0b              lh    x18, 2(x16!)
6620  2916  03 19 09 b3              mul   x19, x18, x17
6624  2916  80 00 00 31 01 37 07 32  add   x14, x14, x19   | nop
6632   324  00 e4 22 29 00 24 84 92  addi  x9,  x9,  2     | sw   x14, 4(x8!)
6640    36  00 65 03 33              add   x6,  x10, x6
6644    36  00 76 83 b3              add   x7,  x13, x7
6648     4  01 81 25 01 01 82 87 5e  zlp   x5,  24,  56    | lw   x10, 24(x2)
6656     4  80 00 01 b1 00 a0 02 12  li    x4,  10          | mv   x3,  x0
6664     4  80 00 03 31 00 00 04 12  li    x8,  0           | mv   x6,  x0
```

```
*/
void matrix_mul_matrix(ee_u32 N, MATRES *C, MATDAT *A, MATDAT *B) {
        ee_u32 i,j,k;
        for (i=0; i<N; i++) {
                for (j=0; j<N; j++) {
                        C[i*N+j]=0;
                        for(k=0;k<N;k++)
                        {
                                C[i*N+j]+=(MATRES)A[i*N+k] * (MATRES)B[k*N+j];
                        }
                }
        }
}
```

Floating Point Matrix Multiplication & Inversion

Benchmarking on Trv32p<n>f and Trv32p<n>fx

Substantial Performance Increase

Single-issue Trv32p5f

16 cycles inner loop

```
1   addi    x5,  x10, 60
1   addi    x6,  x11, 60
2   fmv.w.x f2,  x0
2   fmv.w.x f1,  x0
3   addi    x7,  x11, 4
3   flw     f10, 0(x11)
3   flw     f3,  0(x12)
3   flw     f8,  4(x12)
3   flw     f5,  0(x7)
3   fmul.s  f4,  f10, f3
3   fmul.s  f6,  f5,  f8
3   fmul.s  f7,  f10, f8
3   fmul.s  f11, f5,  f3
3   fsub.s  f9,  f4,  f6
3   fadd.s  f12, f7,  f11
3   addi    x12, x12, 64
3   addi    x11, x7,  4
3   fadd.s  f2,  f2,  f9
3   fadd.s  f1,  f1,  f12
3   bltu    x7,  x6,  -60
2   addi    x7,  x10, 4
2   fsw     f2,  0(x10)
2   addi    x11, x11, -64
2   addi    x10, x7,  4
2   addi    x12, x12, -504
2   fsw     f1,  0(x7)
2   bltu    x7,  x5,  -96
1   addi    x3,  x3,  1
1   addi    x12, x12, -64
1   addi    x11, x11, 64
1   blt     x3,  x4,  -120
```

Static dual-issue Trv32p5fx

11 cycles inner loop

```
    do      x13, 108
    slli    x3,  x13, 3
    zlp     x13, 36,  84
1   do      x13, 92
1   mv      x4,  x12
1   nop                         | nop
2   fmv.w.x f2,  x0             | mv   x5,  x11
2   fmv.w.x f1,  x0             | mv   x6,  x4
3   flw     f10, 4(x5!)
3   flw     f3,  0(x6)
3   add     x6,  x6,  x3        | flw  f8,  4(x6)
3   fmul.s  f4,  f10, f3        | flw  f5,  4(x5!)
3   fmul.s  f7,  f10, f8
3   fmul.s  f11, f5,  f3
3   fmul.s  f6,  f5,  f8
3   fadd.s  f12, f7,  f11
3   fsub.s  f9,  f4,  f6
3   fadd.s  f1,  f1,  f12
3   fadd.s  f2,  f2,  f9
2   addi    x4,  x4,  8
2   fsw     f2,  4(x10!)
2   fsw     f1,  4(x10!)
1   add     x11, x3,  x11
```

C source code

```
for (int i=0; i < dim; i++) {
      for (int j=0; j < dim; j++) {
            int k = j;
            cmplx_t r = 0;
            for (int l=0; l < dim; l++) {
                  r += *(A + l) * *(B + k);
                  k += dim;        // b next row
            }
            *D = r;
            D++;
      }
      A += dim;              // a next row
}
```

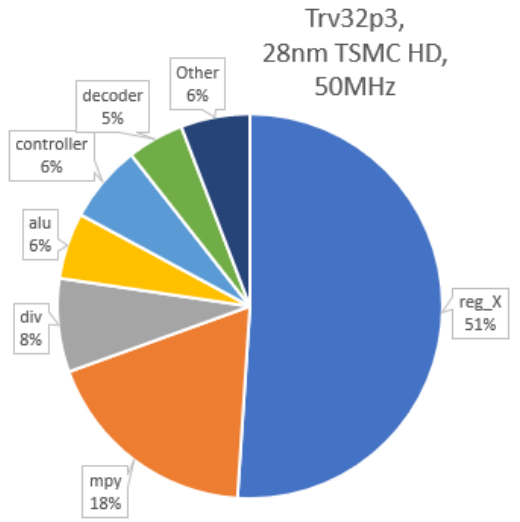
ISA	Performance		
	Optimization Objective	Cycles inv8 & mxmul8 / Code Size	
Trv32p3f (IMF)	Min. cycles >	5258 3824	10408
	Min. code size >	13440 12534	5472
Trv32p3fx (IMF + ILP + zero-ovh. loop + post-modify)	Min. cycles >	3873 2177	8408
	Min. code size >	9055 7983	5420
Trv32p5f (IMF)	Min. cycles >	5275 3828	10528
	Min. code size >	13984 13050	5576
Trv32p5fx (IMF + ILP + zero-ovh. loop + post-modify)	Min. cycles >	3942 2189	12212
	Min. code size >	10278 9010	5576

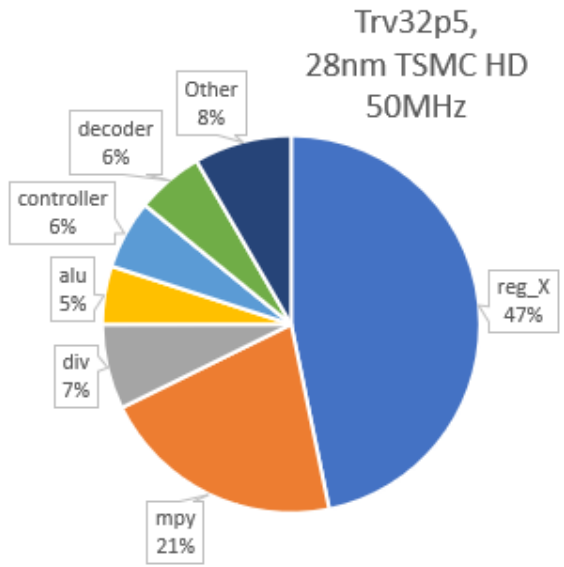
Trv (RISC-V ISA) Models

Benchmarking of Trv32p<n>x

ISA	Compiler Performance: CoreMark			Frequency & Gate Count (28nm TSMC)		
	Optimization objective	CM / MHz	Code size	Optimization objective	Freq.	Gates*
Trv32p3 (IM)	Min. cycles >	3.27	13,116	Max. freq. (HS) >	1.33 GHz	37.9 kGE
	Min. code size >	2.07	6,940	Min. area (HD) >	50MHz**	25.7 kGE
Trv32p3x (IM + ILP + zero-ovh. loop + post-modify)	Min. cycles >	3.65	9,182	Max. freq. (HS) >	1.29 GHz	42.7 kGE
	Min. code size >	2.10	5,040	Min. area (HD) >	50MHz**	32.0 kGE
Trv32p5 (IM)	Min. cycles >	3.10	12,888	Max. freq. (HS) >	1.42 GHz	31.9 kGE
	Min. code size >	2.06	6,888	Min. area (HD) >	50MHz**	28.6 kGE
Trv32p5x (IM + ILP + zero-ovh. loop + post-modify)	Min. cycles >	3.32	9,090	Max. freq. (HS) >	1.40 GHz	40.6 kGE
	Min. code size >	2.01	5,046	Min. area (HD) >	50MHz**	34.7 kGE

* 2-input drive-1 NAND gates
** Clock frequency target set, resulting in low area



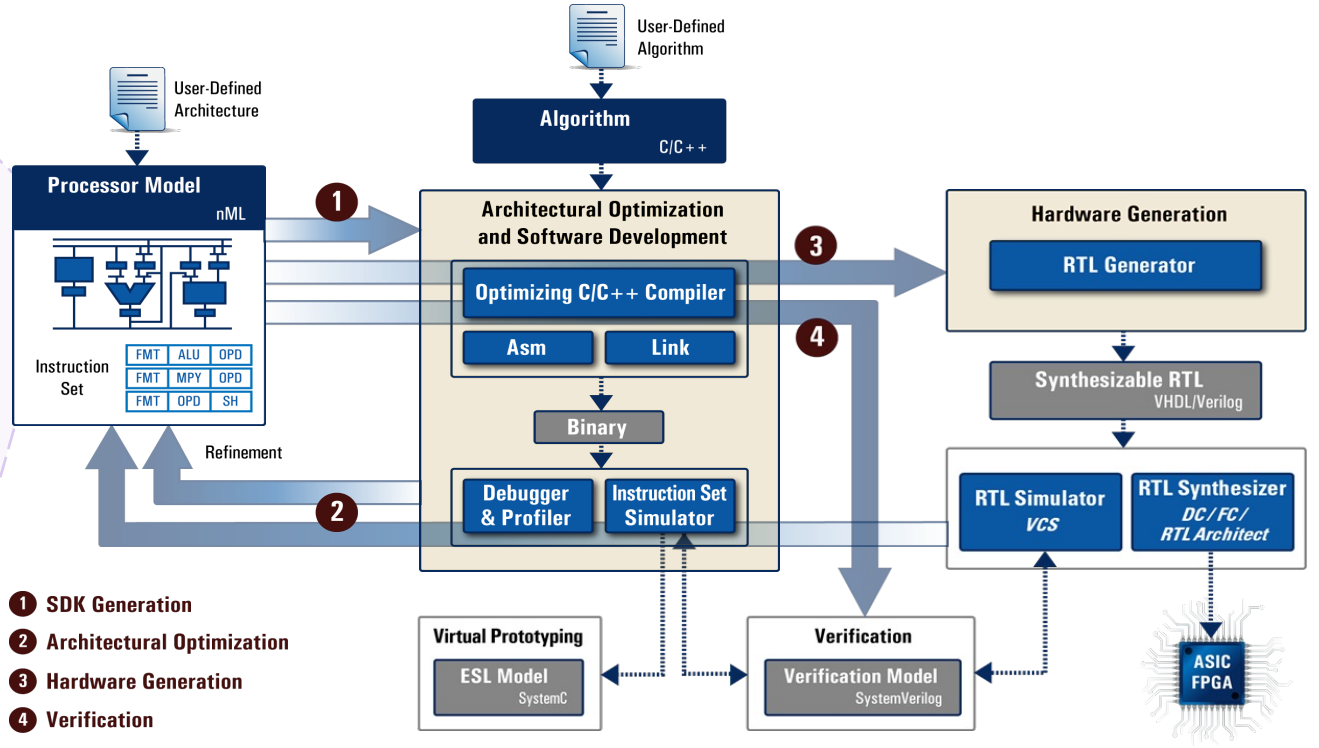


Synopsys' RISC-V Extension Support

ISA Extension of Any "Trv" Model, Using ASIP Designer Tool-Suite

RISC-V Design Solutions Use ASIP Designer Tool-Suite

```
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 descriptions of Trv family are shipped with ASIP Designer tools
- Designers can extend these nML descriptions as desired
  - By using predefined extension stubs in an nML description, within the custom-2 opcode space: “SDX” variant of Trv32p3
  - By direct editing of these nML descriptions

Trv (RISC-V ISA) Models

SDX: Simple Datapath eXtensions

Scalar Extensions

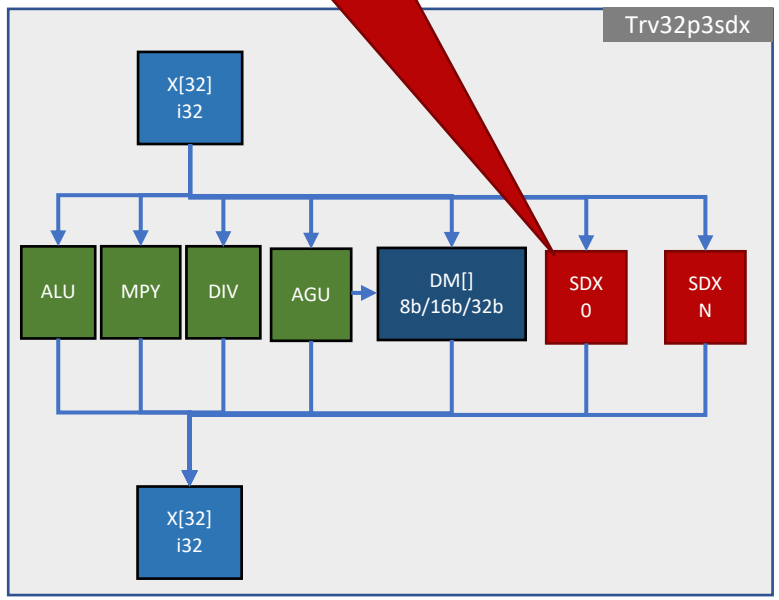
- SDX is a mechanism to add simple extension instructions to Trv (RISC-V)
  - nML model of Trv32p3 with predefined **stubs** for extension instructions
  - User codes the **behavior** of the stubs in PDG (bit-accurate C code)
  - Compiler intrinsics that target the extension instructions are provided (and can be modified)
- Benefits
  - No (deep) nML knowledge required: extensions can be created by SW engineers
  - Fast exploration of extensions with compiler-in-the-loop & synthesis-in-the-loop
- Extensions encoded in RISC-V custom-2 space
- Operand options
  - 3-register (32 and 64-bit)
  - Accumulate
  - Additional single register inputs and outputs

Behavioral model (PDG)

```
w32 sdx0(w32 a, w32 b)
{
  w32 r;
  r[15:0] = a[15:0] + b[15:0];
  r[31:16] = a[31:16] + b[31:16];
  return r;
}
```

Compiler intrinsics

```
int sdx0(int,int);
int add2(int,int);
```



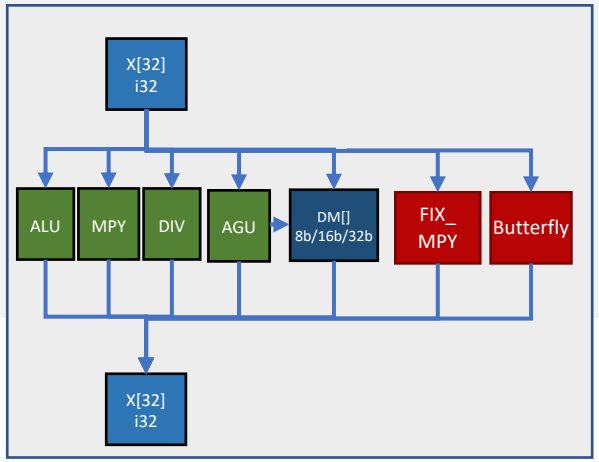
Trv (RISC-V ISA) Models

SDX Examples Provided With ASIP Designer

Performance Increase with Scalar Extensions

FFT

- SDX instructions accelerating
  - Complex fixed-point multiplication & scaling
    sdx1 *rd,rs1,rs2*
  - ABS(x) function: ***sdx2*** *rd, rs1, rs2(x0)*
  - FFT Butterfly: ***sdx5*** *rd,rs1,rs1*
- Specialization:
  - Fractional data types
  - Complex numbers (16bit/16bit -> 32bit register)
- Speedup: 280% Area increase: 31%

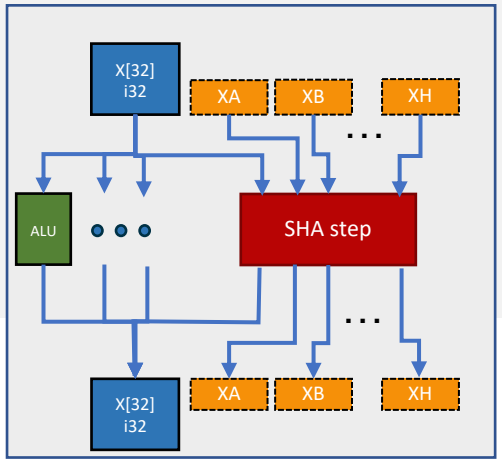


SHA256

- Computes a hash of message W using bitwise AND, OR, XOR operations, shift operations and additions
- Custom data path is ideal to implement the complex hash function in one instruction
- Additional state of the hash functions (8 state variables) require an SDX variant that supports **8 additional register reads and writes**

```
sdx7 rd, rs1, rs2, x24,x25,…, x32
```

- Speedup: 270% Area increase: 16%



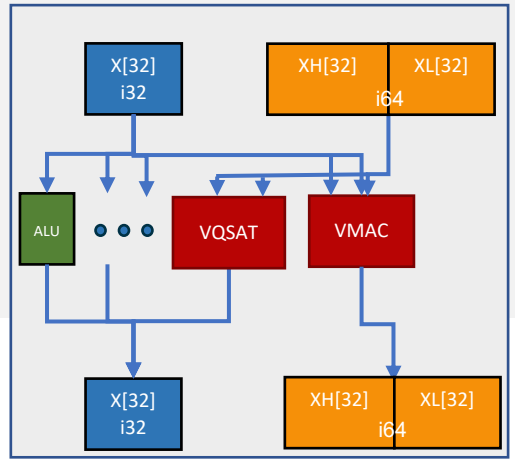
Keyword Spotting

- Based on 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: 1160% Area increase: 16%



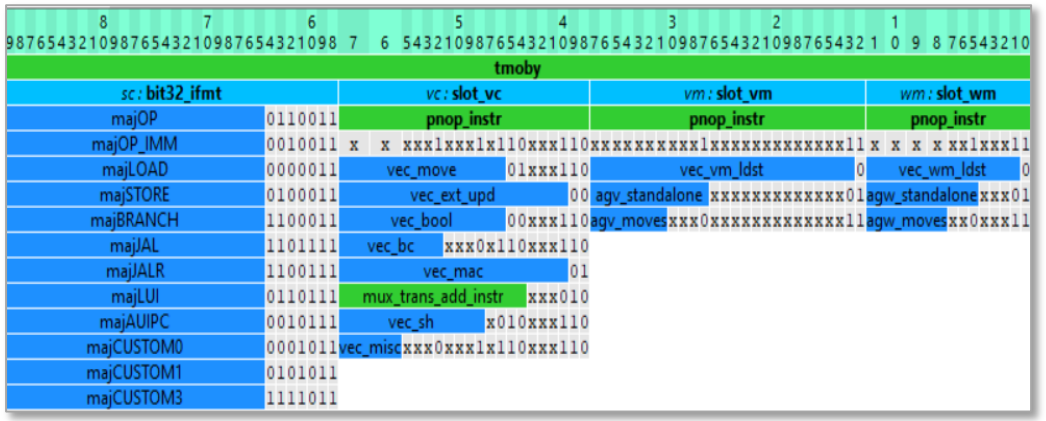
Large-Scale Trv Extensions

VLIW with Custom Vector Extensions

Tmoby: Application-Specific Processor (ASIP) for Acceleration of MobileNet v3

Application: MobileNet V3

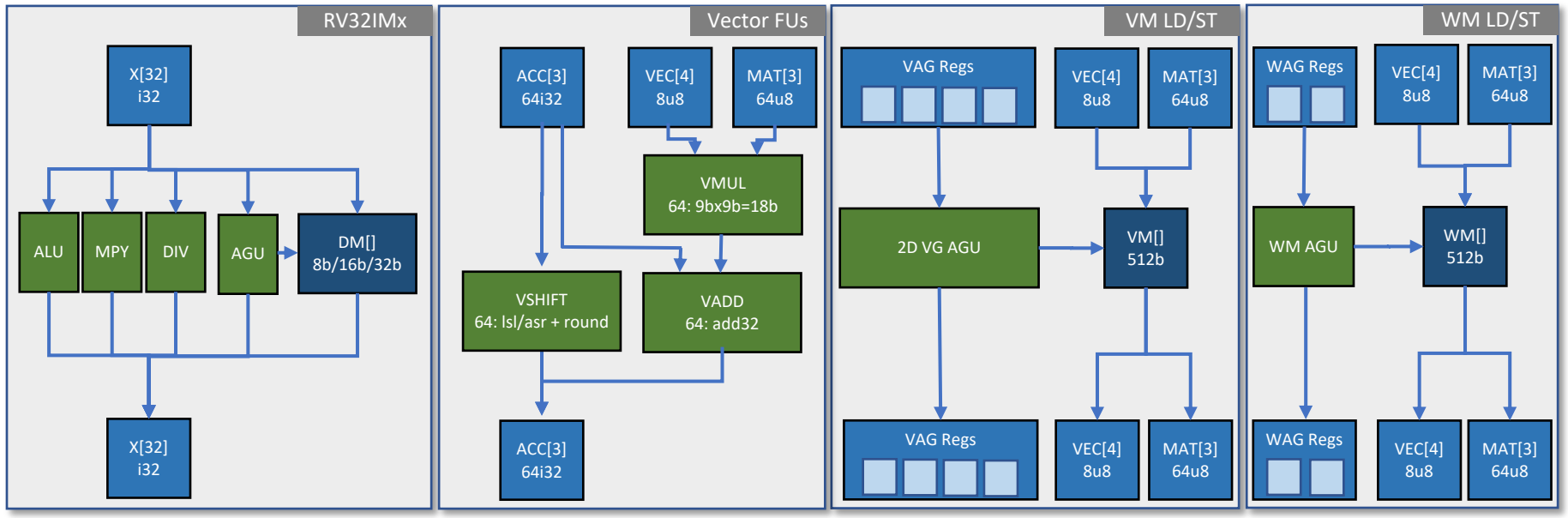
- TensorFlow code converted to C code with Tmoby-specific vector intrinsics
- Main function
  - Implementation of neural-network graph
  - Memory copies
  - Kernels
- Kernels
  - Conv_2D (pointwise)
  - Depthwise_Conv_2D
  - Add
  - Average_Pool_2D
  - Softmax



- 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
  - 360x cycle count decrease



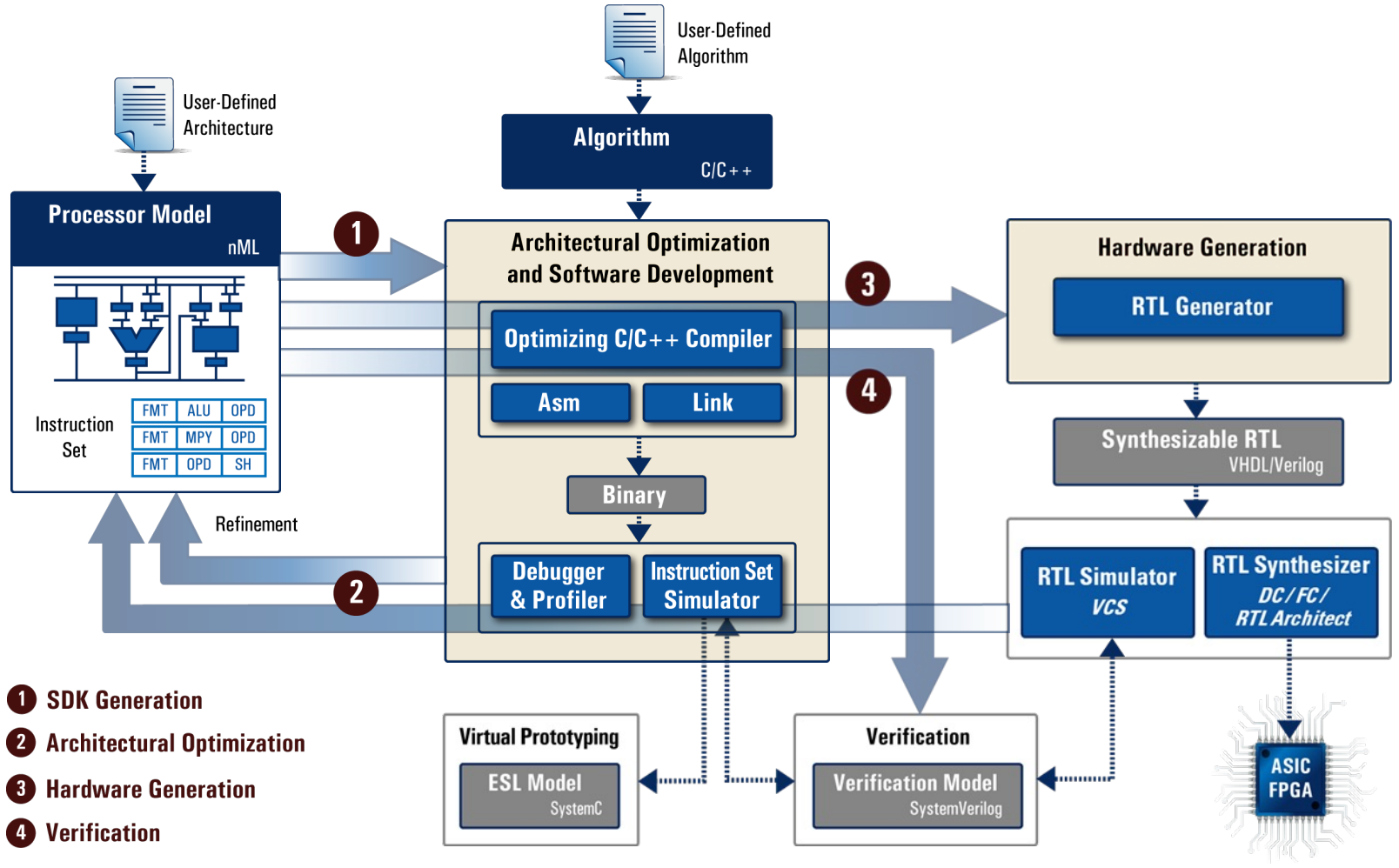
Synopsys' RISC-V Development Infrastructure

Using ASIP Designer Tool-Suite

ASIP Designer

Tool Flow

- Industry-leading tool to design your own processor/DSP
- Language-based description of ISA and microarchitecture
- Single processor model ensures that SDK and RTL are in sync
- Licensed as a tool (not IP), no royalties

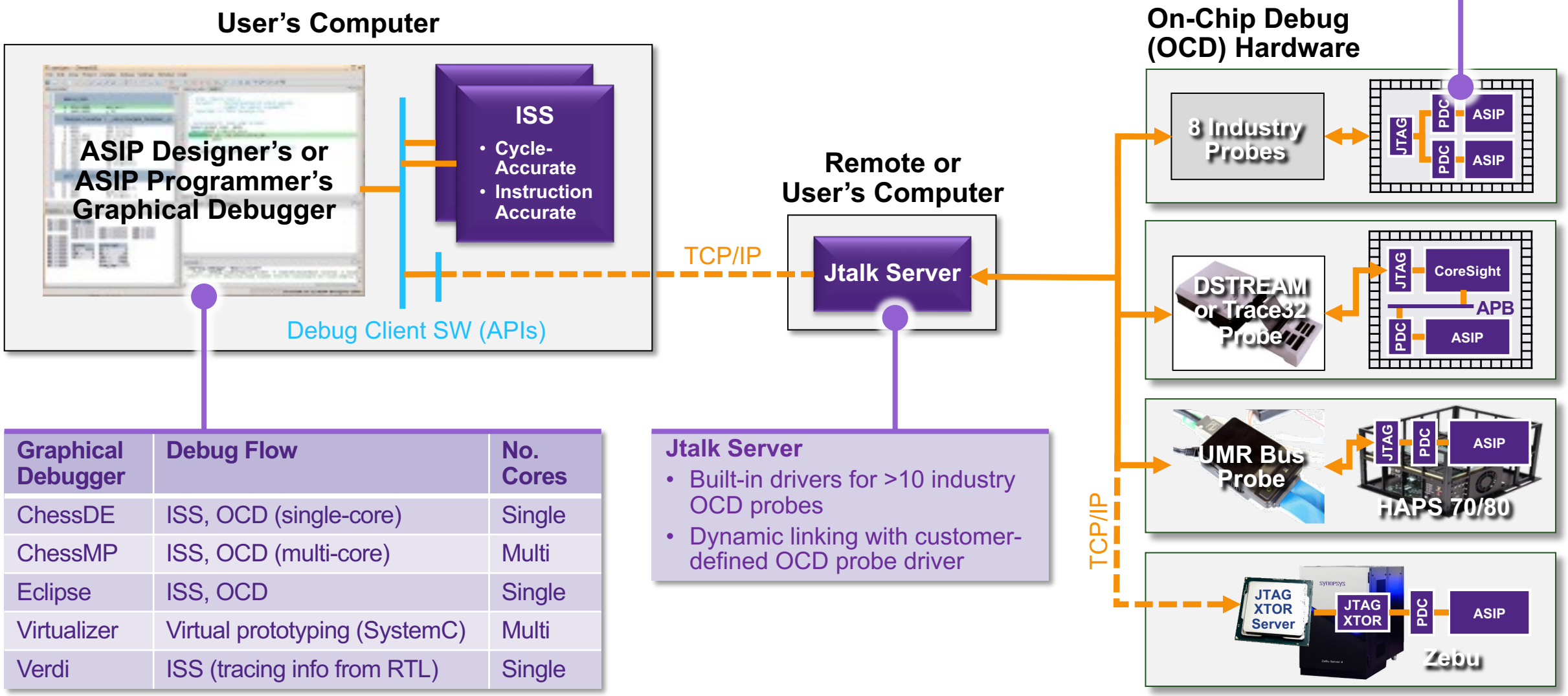


Supported design steps

- Modeling of instruction-set architectures: 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

Debugging

Comprehensive Integrated Debug Solution



Graphical Debugger	Debug Flow	No. Cores
ChessDE	ISS, OCD (single-core)	Single
ChessMP	ISS, OCD (multi-core)	Multi
Eclipse	ISS, OCD	Single
Virtualizer	Virtual prototyping (SystemC)	Multi
Verdi	ISS (tracing info from RTL)	Single

Jtalk Server

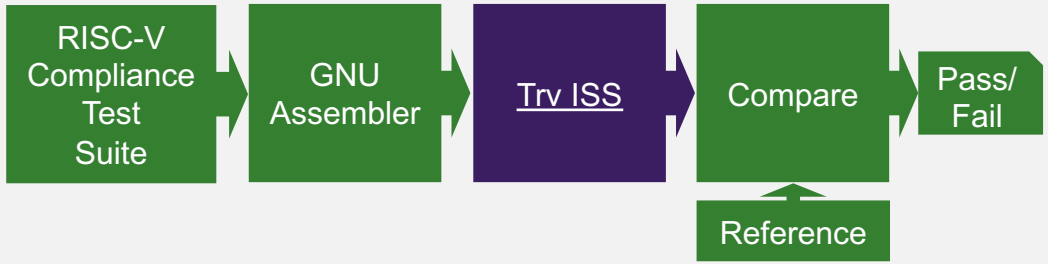
- Built-in drivers for >10 industry OCD probes
- Dynamic linking with customer-defined OCD probe driver

Verification

- RISC-V ISA compliance of Trv32p<n> models has been verified using the RISC-V Compliance Framework*

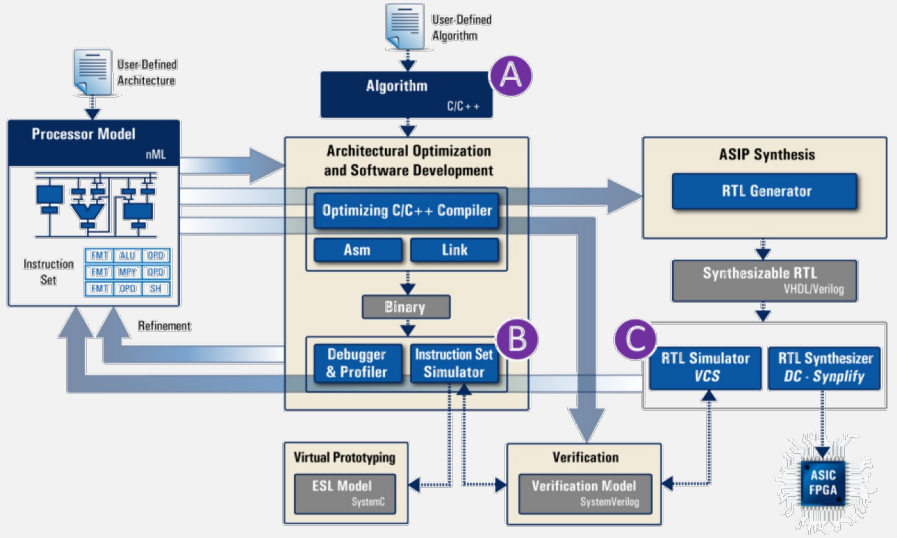
  https://github.com/riscv/riscv-compliance/

  * Excluding traps & exception tests



- Use of ASIP Designer tool-suite offers increased confidence
  - Single processor model ensures that SDK and RTL are in sync
  - 300+ successful tape-outs (not only RISC-V), same tools stress-tested daily by hundreds of designers

- Tool validation process



  - Regression testing at C, ISS & RTL abstractions
  - Applied by customers
  - Applied by Synopsys
    - Automated continuous integration, nightly regression, release testing
    - 225K tests on 135 processor models (incl. Trv)
  - Development process is ISO9001 certified

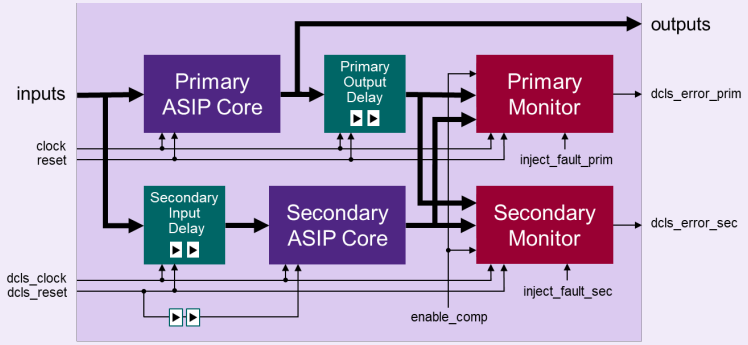
Functional Safety Support in ASIP Designer

Using ASIP Designer Advanced Tool Features

What's HOT – Functional Safety Support in ASIP Designer

ASIP Designer Automates the Generation of RTL with FuSa Extensions

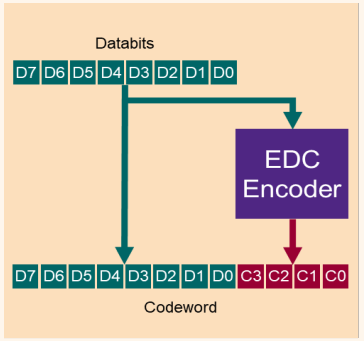
Dual-core lockstep (DCLS) in ASIP core



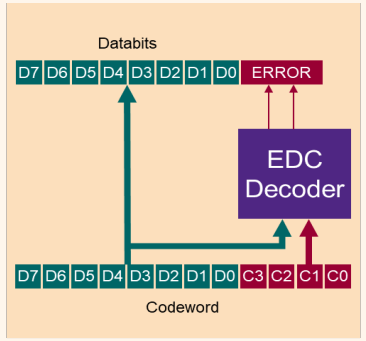
dcls_module

- Automatic duplication of ASIP core
- Automatic addition of monitoring logic (comparator)
- Highly configurable solution
  - No. of delay stages, pipelining, duplicate and alternative monitor implementations, etc.

Error detection logic (EDC) in ASIP core



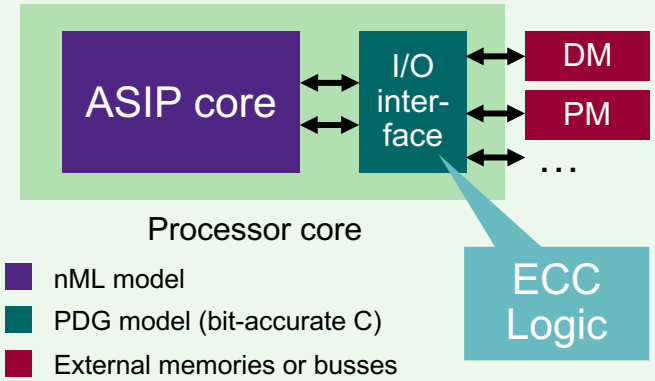
EDC Encoder



EDC Decoder

- Automatic generation of error detection logic
  - For registers, register files, pipeline registers
  - Uses Hamming codes or parity bits
  - Hierarchical generation of error codes (based on RTL hierarchy)
  - Template-based, for enhanced flexibility

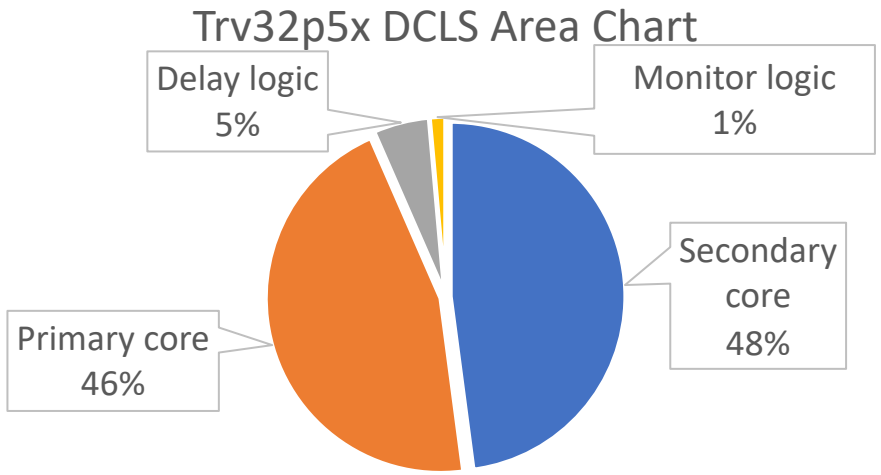
Error correcting codes (ECC) in memory



- Designers can describe custom ECC logic in the I/O interface model
- ASIP Designer automatically generates RTL implementation and bit-accurate simulation model

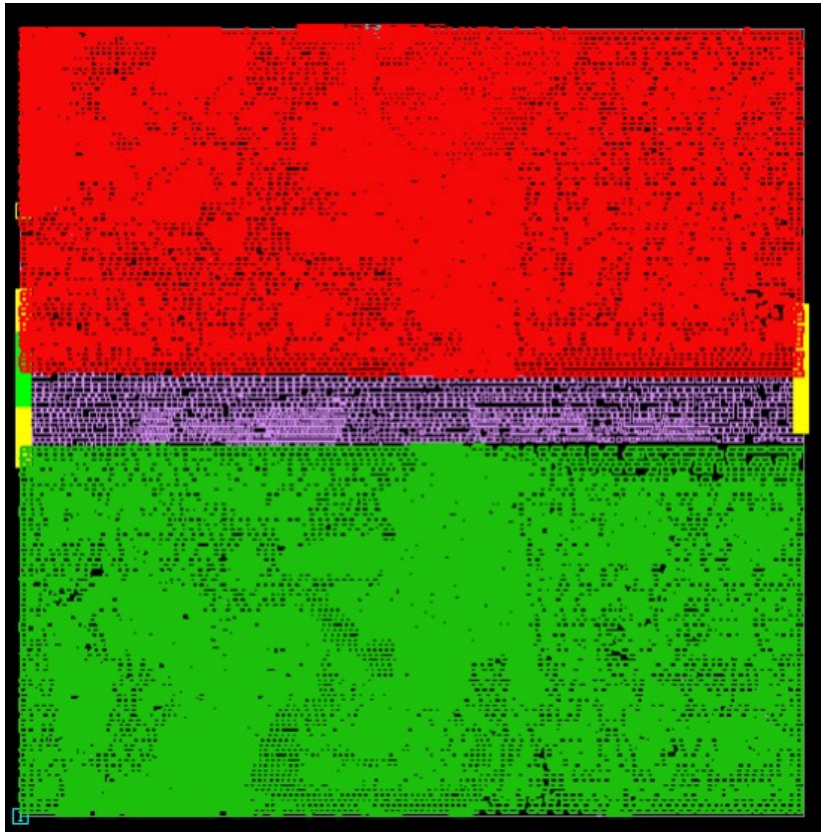
These capabilities support ASIP Designer's adoption in automotive and security markets

Trv32p5x with DCLS



ISA	Frequency & Gate Count (28nm TSMC)		
	Optimization objective	Freq.	Gates*
Trv32p5x DCLS	Max. freq. (HS) >	1.20 GHz	109 kGE
	Min. area (HD) >	50MHz**	102 kGE
Trv32p5x (IM + ILP + zero-ovh. loop + post-modify)	Max. freq. (HS) >	1.40 GHz	40.6 kGE
	Min. area (HD) >	50MHz**	34.7 kGE

Layout post-synthesis, cores are highlighted red (primary) and green (secondary)



* 2-input drive-1 NAND gates
** Clock frequency target set, resulting in low area

STMicroelectronics Accelerates Design of Special-Purpose, High-Reliability Processors with Synopsys ASIP Designer

ST Achieves 5x Faster Design and Verification Turnaround Time using Synopsys ASIP Designer


Customer Spotlight: STMicroelectronics Accelerates Design of Special-Purpose, High-Reliability Processors with Synopsys ASIP Designer

By Patrick Verbist, S

For applications tha
(ASIPs) provide a go
are more computati
function RTL design
designed to efficien
baseband process

"ST has long experienced the engineering productivity benefits of ASIP Designer and its automatic generation of the C compiler, instruction set simulator, debugger, and RTL. With the new DCLS features in Synopsys ASIP Designer, we can rapidly explore different RTL instantiations of our dual-core processors, saving substantial time and effort while ensuring that we have a reliable design for safety-critical applications."

life.augmented

Christophe Monat,
Manager of Computing and Compilers Center, ST

- ASIP Designer tool maximized productivity of ST engineering team, enabling them to meet their customers' time-to-market demands
- Rapid exploration of different instantiations of their dual core saved ST substantial time and effort while ensuring a reliable design for safety-critical applications
- ST contributed domain expertise, helping Synopsys to enhance ASIP Designer with new, highly configurable DCLS FuSa and security features

NSITEXE Develops Multiple RISC-V Based Custom Processors for Automotive Applications in Half the Time with Synopsys ASIP Designer

Rapid Architectural Exploration and Automatic Generation of SDKs Accelerates Design Time

NSITEXE Successfully Develops Multiple Custom Processors for Automotive Applications in Half the Time with Synopsys ASIP Designer Tool

ASIP Designer Tool Provides Rapid Architectural Exploration and Automatic Generation of Software Development Kits to Accelerate Design Time

MOUNTAIN VIEW, Calif., Sept. 16, 2020 /PRNewswire/ --

- NSITEXE used ASIP Designer to develop five specialized custom processors for compute-intensive signal processing functions
- Automatic generation of software development kit enabled NSITEXE to complete the design with
- R

Synops
Corpor
Design
proces
was ab
develo

"Synopsys' ASIP Designer Tool provided us with ready-to-use processor examples that could be extended to implement our custom ISA, allowing us to rapidly develop our custom vector extensions that satisfied our functionality and performance requirements. ASIP Designer enabled us to develop the data flow processor model in record time and deliver it to the customer on schedule."

NSI-TEXE

Sadahiro Kimura,
Manager, Semiconductor IP R&D Unit,
Advanced Technology Development Section at NSITEXE, Inc.

- NSITEXE used ASIP Designer to develop five specialized custom processors for compute-intensive signal processing functions
- Automatic generation of software development kit enabled NSITEXE to complete the design with limited resources and on a tight schedule
- Rapid setup of a virtual prototype of NSITEXE's multicore DFP enabled by the ASIP Designer's ability to export the processor's simulation model with SystemC interfaces

SYNOPSYS®
Silicon to Software™

Thank You