SYNOPSYS®
Silicon to Software™

Extending RISC-V for Application-Specific Requirements

5th RISC-V Workshop
November 29, 2016
Google – Quad Campus

Steve Cox (scox@synopsys.com)
Drew Taussig (dtaussig@synopsys.com)

DesignWare Processor IP

Unrivaled Efficiency for Embedded Applications

ARC EM Family

- Optimized for **ultra low power** IoT
- 3-stage pipeline w/ high efficiency DSP
- Power as low as 3uW/ MHz
- Area as small as 0.01mm^2 in 28HPM

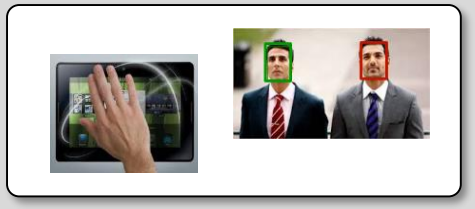
ARC SEM Family

- **Security** processors for IoT and mobile
- Protection against HW, SW, and side channel attacks
- SecureShield enables Trusted Execution Environments

ARC HS Family

- **Highest performance** ARC cores to date
- High speed 10- stage pipeline
- SMP Linux support
- Single, dual, quad core configurations

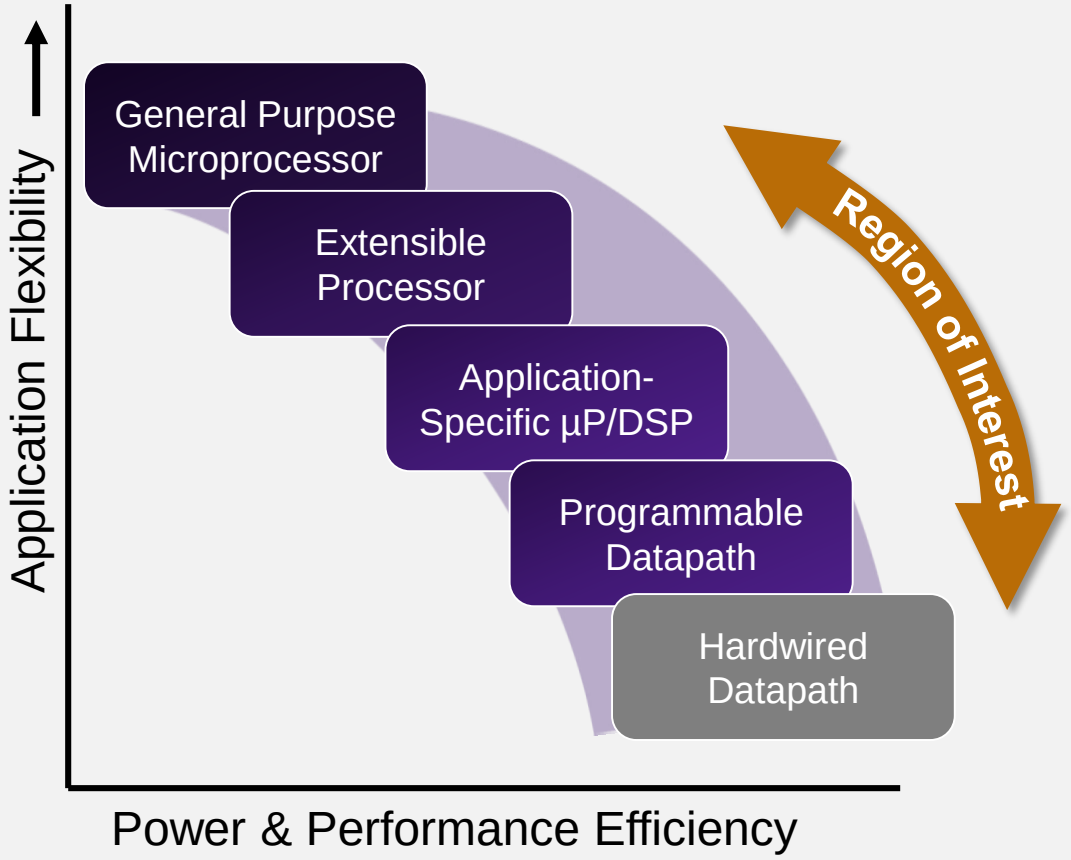
EV Family



- Heterogeneous multicore for **vision** processing
- State-of-the-art convolutional neural network (CNN)
- 5X better power efficiency than existing solutions

SYNOPSYS®

But Sometimes a Pre-Defined ISA Is Insufficient

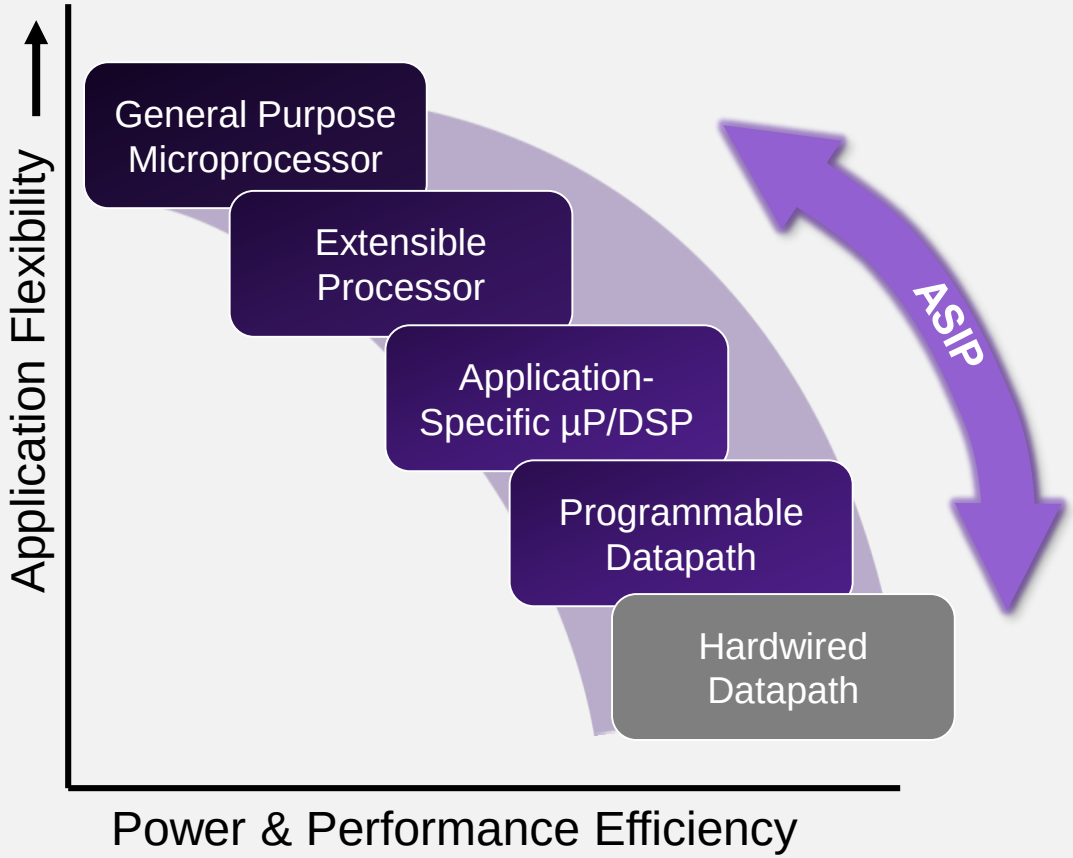


An application-optimized ISA may be required which blends **performance**, **power efficiency**, and **programmability**

SYNOPSYS®

Synopsys Processor Solutions

IP & Tools Address Broadest Range of CPU & DSP Requirements



An application-optimized ISA may be required which blends **performance**, **power efficiency**, and **programmability**

ASIP: Application-Specific Processor

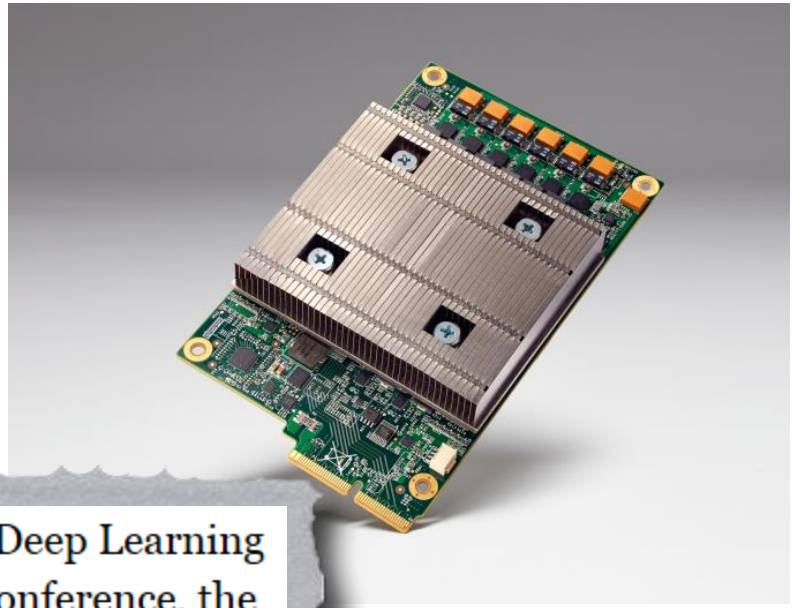
A **purpose-built** processor implementation with an **application-optimized** ISA

SYNOPSYS®

Examples of ASIPs in Industry

Example: Google TPU

- May 18th, 2016: Google discloses the existence of TPU
- Internally developed processor, with specialized ISA
- Google will not disclose details, as TPU is seen as a competitive advantage



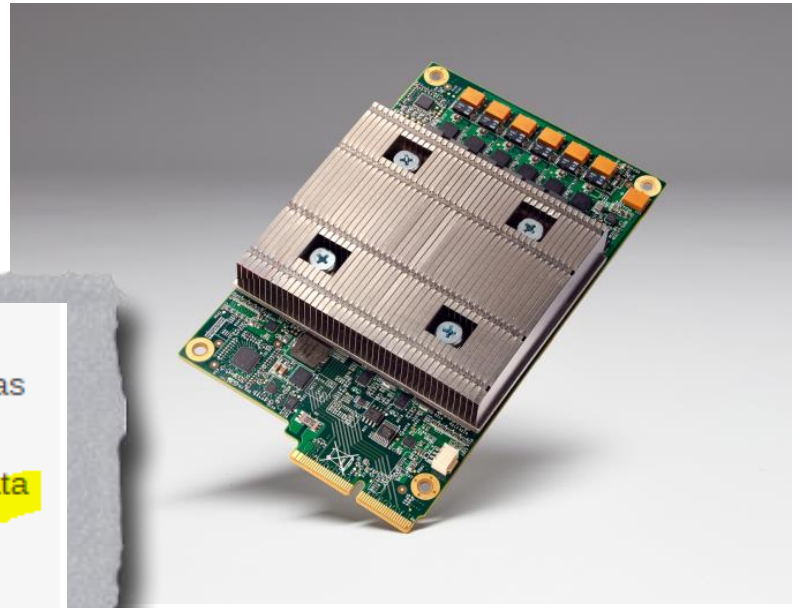
Google GOOGL +0.31%'s announcement last week that they developed a custom chip for Deep Learning has created a lot of press and unanswered questions. At the Google I/O developers conference, the company shared that they have been using an internally-developed processor, called a Tensor Processing Unit (TPU), for over a year to accelerate Deep Learning applications, from Google Street View to their much-heralded win at the game of Go. Rumors have swirled for years that Google may develop their own processors, potentially based on ARM Holdings V8 and / or IBM IBM +0.67% OpenPOWER, to displace Intel INTC +1.21% Xeon server processors. This announcement shows that Google may be more interested in chips that are tailored to accelerate specific workloads, especially for Artificial Intelligence.

http://www.forbes.com/sites/moorinsights/2016/05/26/googles-tpu-chip-creates-more-questions-than-answers

Examples of ASIPs in Industry

Example: Google TPU

- Straying away from standard 32/64 bit data types?
- Like a VLIW DSP with SIMD and “some twist”?

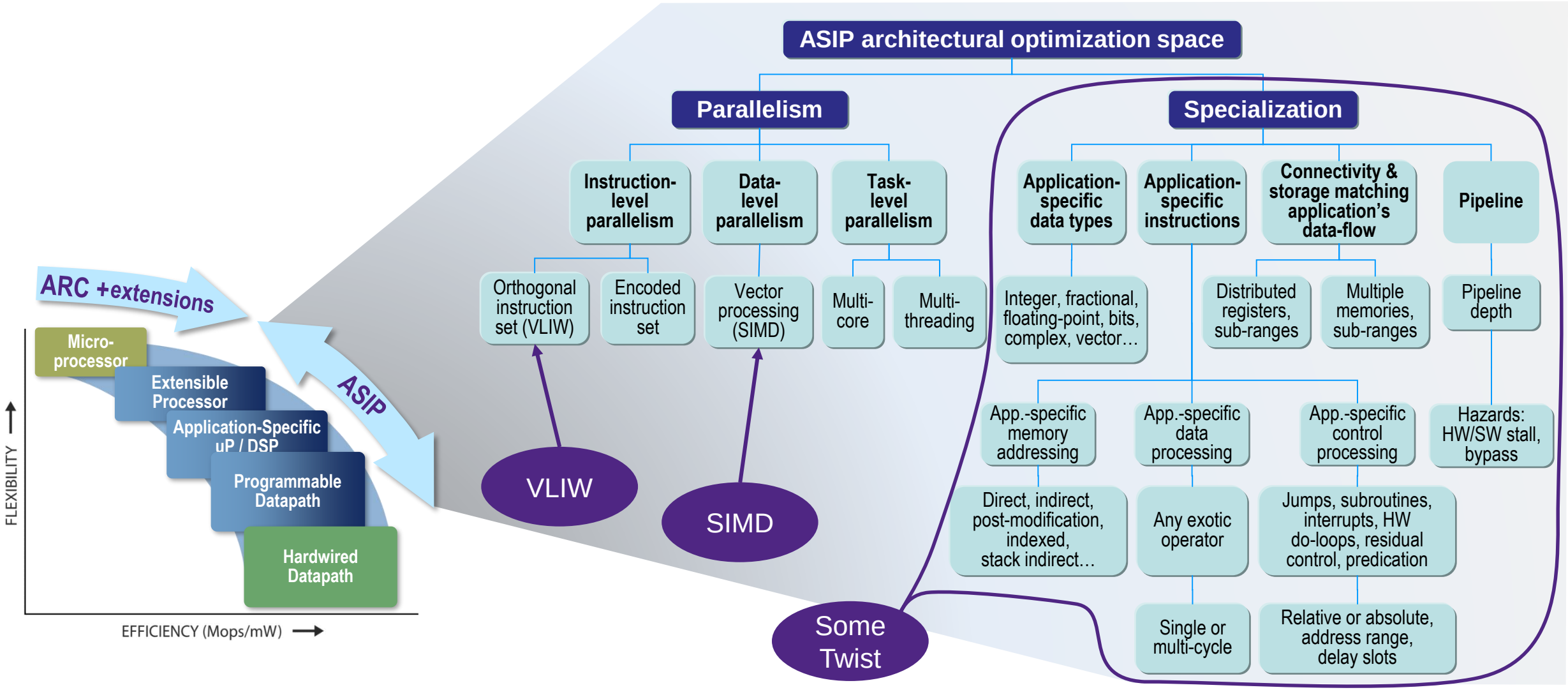


More will come later this year, but for now what we know is that this is an actual processor with an ISA of some kind. What exactly that ISA entails isn't something Google is disclosing at this time - and I'm curious as to whether it's even Turing complete - though in their blog post on the TPU, Google did mention that it uses "reduced computational precision." It's a fair bet that unlike GPUs there is no ISA-level support for 64 bit data types, and given the workload it's likely that we're looking at 16 bit floats or fixed point values, or possibly even 8 bits.

Reaching even further, it's possible that instructions are statically scheduled in the TPU, although this was based on a rather general comment about how static scheduling is more power efficient than dynamic scheduling, which is not really a revelation in any shape or form. I wouldn't be entirely surprised if the TPU actually looks an awful lot like a VLIW DSP with support for massive levels of SIMD and some twist to make it easier to program for, especially given recent research papers and industry discussions regarding the power efficiency and potential for DSPs in machine learning applications. Of course, this is also just idle speculation, so it's entirely possible that I'm completely off the mark here, but it'll definitely be interesting to see exactly what architecture Google has decided is most suited towards machine learning applications.

http://www.anandtech.com/show/10340/googles-tensor-processing-unit-what-we-know

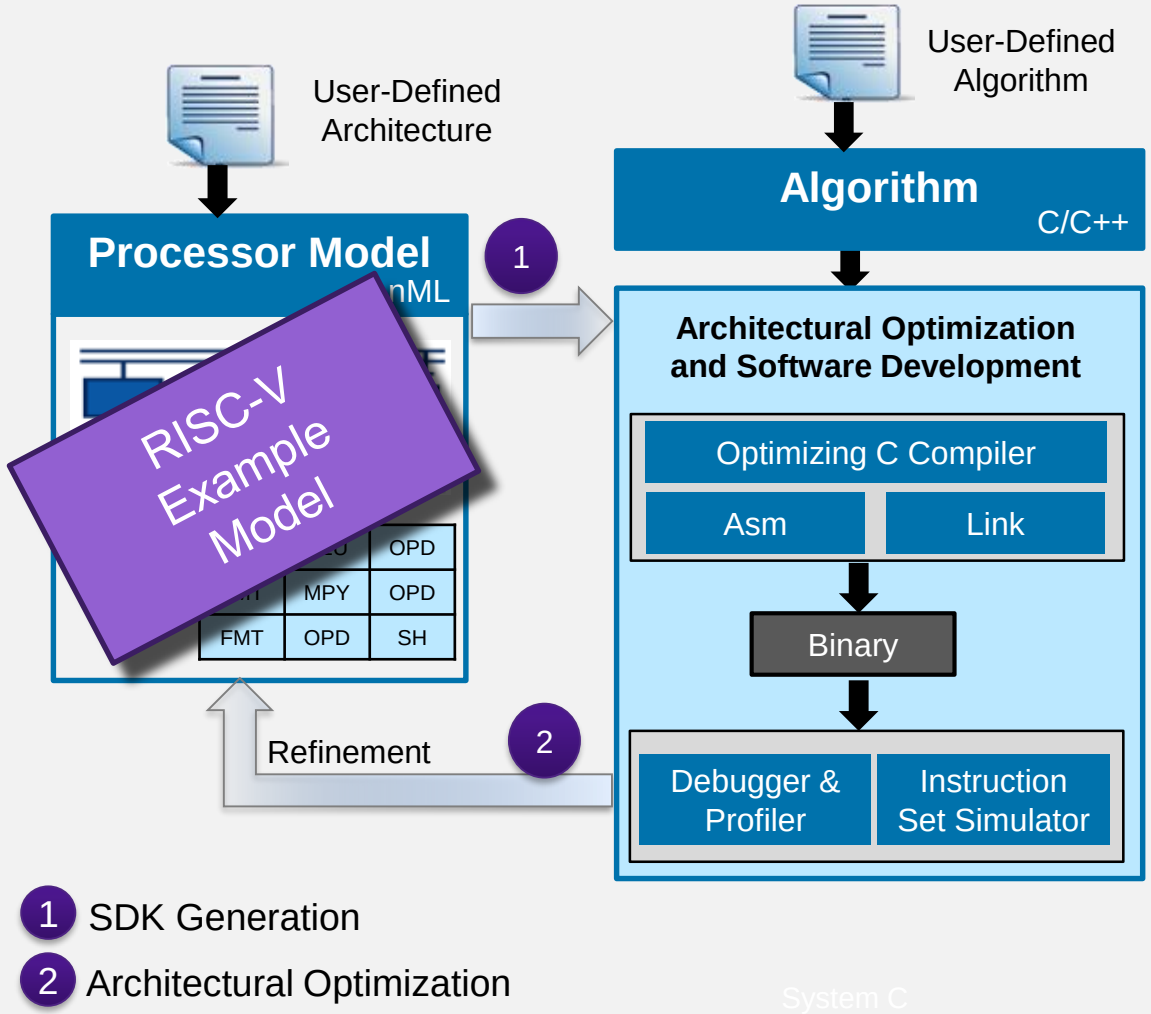
ASIP – Combining Efficiency and Flexibility



SYNOPSYS®

ASIP Designer - Automating ASIP Design

Architecture Definition and Optimization

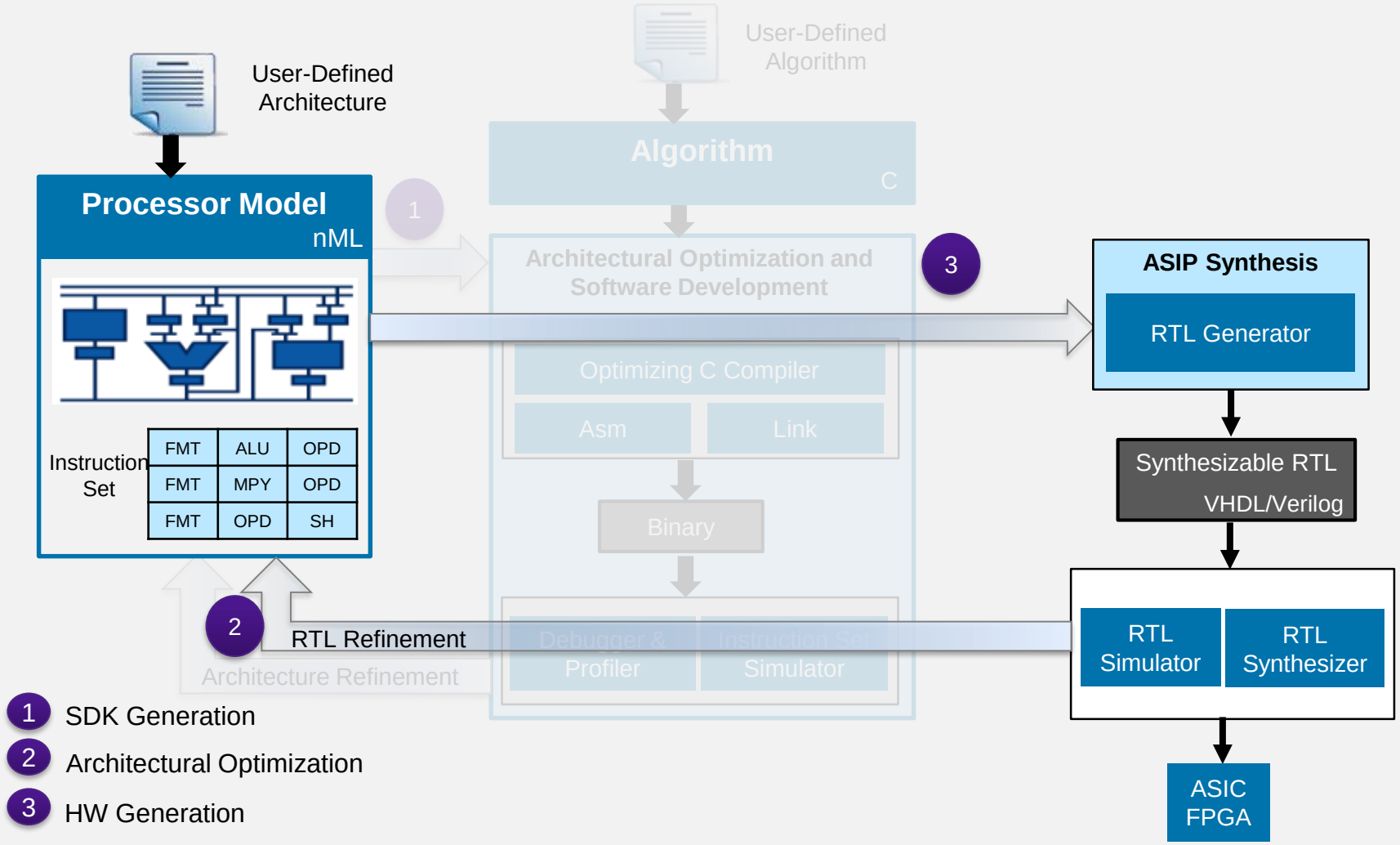


- User describes architecture in processor model (nML)
- ASIP Designer creates full SDK
  - ISS (Instruction Set Simulator)
  - Graphical/interactive debugger
  - Assembler, disassembler, linker
  - Optimizing C/C++-compiler
    - LLVM front end + proven, adaptable back-end enabling wide architectural flexibility
- Compiler-in-the-loop optimization
  - Performance critical code is highlighted
  - Architecture is refined for improved performance
- Process can start with a pre-existing example model
  - For example, RISC-V (available now)

SYNOPSYS®

ASIP Designer - Automating ASIP Design

RTL Optimization



- Tool also generates synthesizable RTL
  - From same ASIP model
- Designer assesses RTL
  - Design size
  - Max operating frequency
  - Critical paths
  - Power estimation
- Analysis of RTL seeds further refinement/optimization
  - ASIP model is refined
  - SDK is automatically adapted
  - All elements stay in-sync, minimizing verification

SYNOPSYS®

ASIP Designer Starting Point Examples

Besides RISC-V, there are many other examples provided with various characteristics already modeled

Name	Description
Microcontrollers	
Tnano	16 bit microcontroller, lightweight and configurable
Tmicro	16 bit microcontroller, fully features
DLX–TLX-FLX–ILX	Variants of 32-bit microcontroller
Tmcu	32-bit microcontroller
RISC-V	RISC-V Z-Scale implementation
DSPs and generic parallel processors	
Tdsp	16/32-bit DSP
Tvec	Variants of SIMD processor
Tvliw	Variants of VLIW processor
Domain-specific processors	
Tmotion	Accelerator of motion estimation kernel
Tcom8	SIMD processor optimized for some communication kernels
FFTcore	scalar implementation of complex FFT
MXcore	Matrix processing ASIP for communication kernels
Primecore	SIMD implementation of prime-factor algorithm for FFT & DFT
JEMA, JEMB	Dual ASIP for JPEG encoding (accelerating DCT, VLC)

- Many examples are provided
  - Microcontrollers
  - DSPs
  - SIMD
  - VLIW
  - Multi-threading
  - Domain specific
- Examples are good starting points
  - Start with known working example
  - User provides relevant benchmark(s)
  - Tool leads designers through architectural optimization process

Long History of Broad Use across domains

Hundreds of products shipping today built with ASIP Designer

Only publicly announced customers are shown

- Broadly adopted across a range of applications
- Used in more than 250 unique SoC products

SYNOPSYS®

ASIP Design Methodology Example

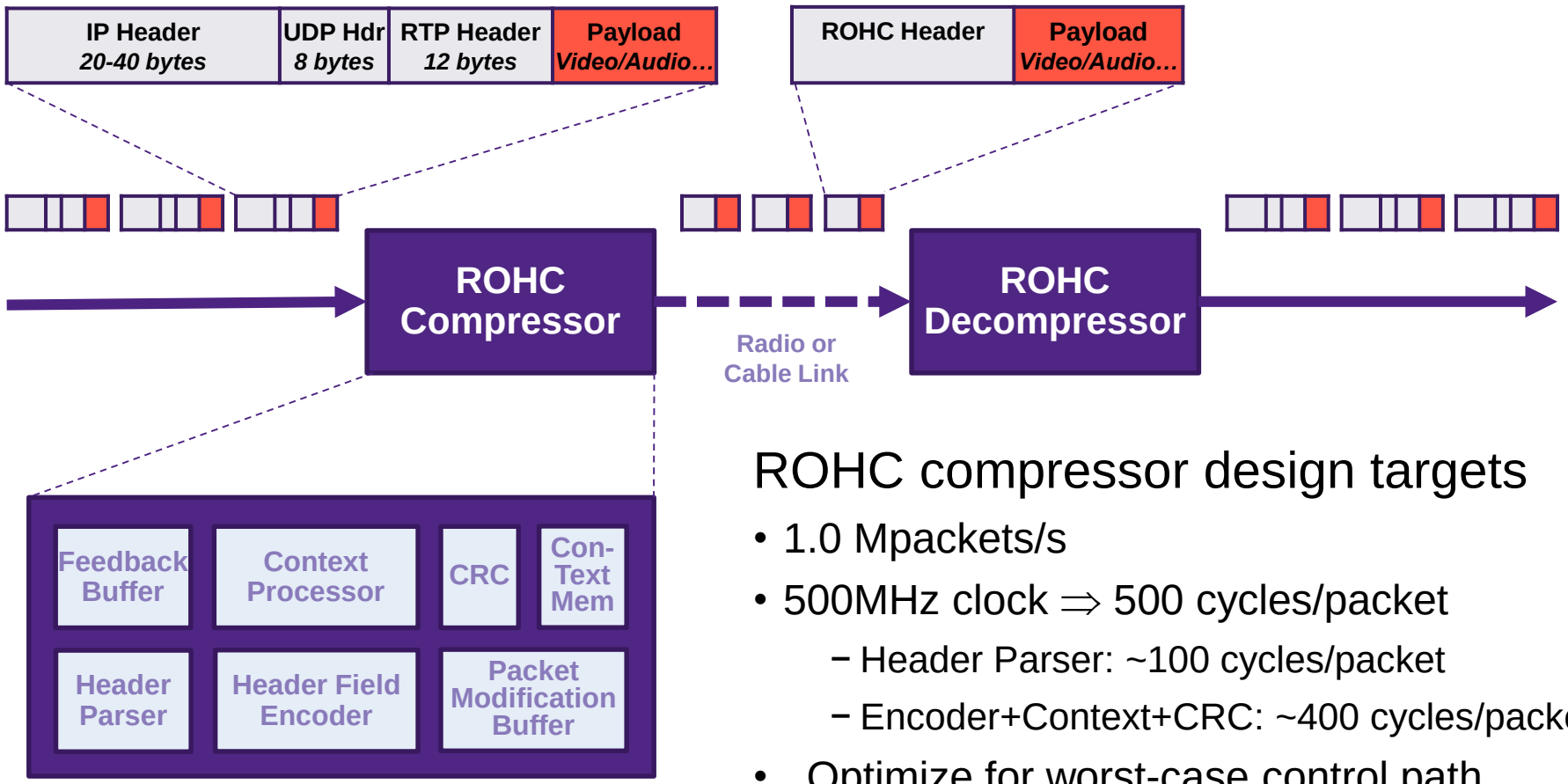
Network Processing – Robust Header Compression (RoHC)

Using RISC-V Example Model as Starting Point

SYNOPSYS®

Challenge: Accelerating ROHC in Network Processing

High Performance Streaming Data (IP/UDP/RTP Protocol)

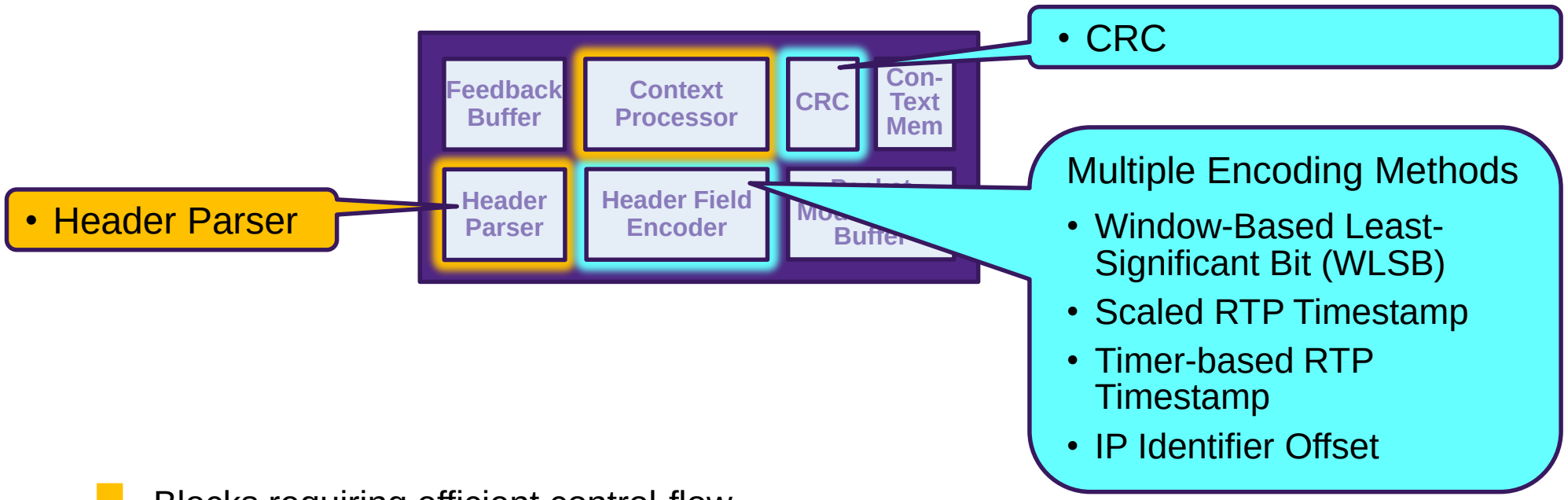


ROHC compressor design targets

- 1.0 Mpackets/s
- 500MHz clock ⇒ 500 cycles/packet
  - Header Parser: ~100 cycles/packet
  - Encoder+Context+CRC: ~400 cycles/packet
- Optimize for worst-case control path

SYNOPSYS®

ROHC Implementation – Acceleration Candidates



Blocks requiring efficient control-flow

→ Microprocessor with efficient branching and logic operations

Blocks requiring efficient control-flow and data processing

→ Microprocessor with parallelism and application-specific instructions

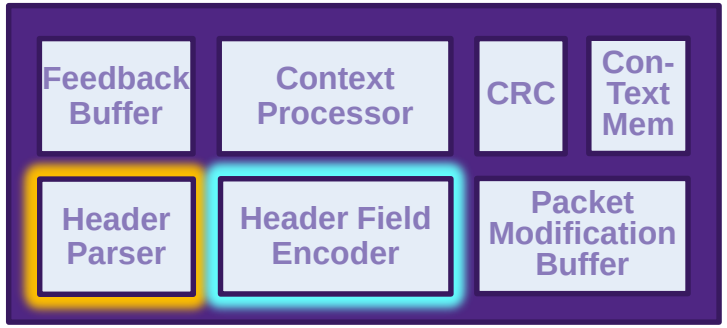
ASIP Designer enables designers to explore optimizations for both kinds of algorithms

SYNOPSYS®

Approach

- Run select ROHC algorithms* on baseline RISC-V model
  - In this presentation: WLSB (first), Header Parser (second)
- Identify issues constraining performance
- Devise architectural enhancements to improve performance and revise model
- Repeat as needed to improve performance

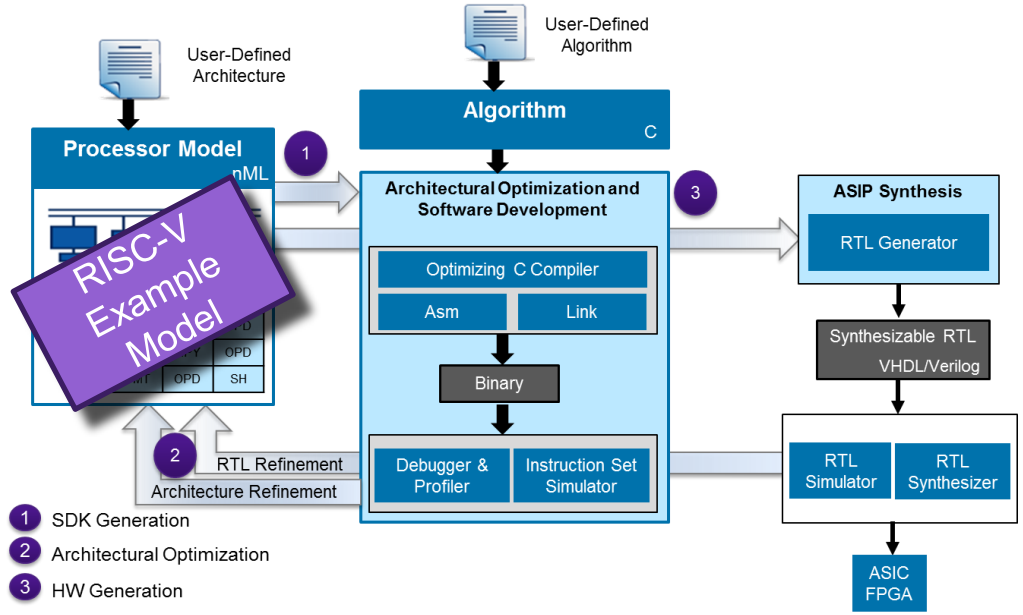
- Consider architectural optimizations such as
  - Parallelism (VLIW, SIMD)
  - “Some Twist” (e.g. application-specific instructions)



* This presentation includes source code fragments from: ROHC Library (rohc-lib.org)

RISC-V Example Model

Provided with ASIP Designer



- RISCV32IM Z-Scale implementation
  - Example model name: “Tzscale”
- Processor features
  - 32 bit data, 32 bit addresses, 32 bit instructions.
  - Central register file with 16 or 32 fields (selectable).
  - Byte addressed data and program memory.
  - Single-issue, three stage protected pipeline: IF – ID/EX – WB.
- Modelling features contained in example
  - Register bypasses and hardware stall rules.
  - Multi-cycle functional unit for division, single-cycle multiplier
  - IO interface implements 8/16/32 bit unaligned memory access.
  - Software emulation of IEEE floating point operations.
- Model can be easily modified to change characteristics or add functionality
  - VLIW, SIMD, or “some twist”

SYNOPSYS®

Baseline RISC-V Example Model Characteristics

ASIP Designer Tzscale model – before ROHC optimizations

- Software benchmarks
  - Dhrystone

Compiler	*DMIPS/MHz*
ASIP Designer (code in 2-separate files)	1.25
ASIP Designer (code combined into a single file)	1.62
UC Berkeley GCC (Z-Scale)	1.35 *

  - ROHC WLSB encoder 6-packet test program

Compiler	*# of instructions executed*	*# of cycles*	*# of cycles/packet*
ASIP Designer	1344	1516	253
gcc (riscv-tools 5.3.0)	1353	?	?

- Implementation
  - TSMC 28HPM @ 500Mhz – 32GPRs

	Cell Area (gates)
Tzscale	9462 (24.5K)
vscale (Z-scale)	10530 (27.3K) **

* - https://riscv.org/wp-content/uploads/2015/06/riscv-zscale-workshop-june2015.pdf

** - RTL source obtained from https://github.com/ucb-bar/vscale and synthesized under same constraints as Tzscale

1st iteration: Consider Instruction Level Parallelism

Common type of optimization and easy to try

- Load/store instructions could execute in parallel with ALU/control instructions
  - Example from WLSB encoder: load next “interval” while calculating current “interval”

```
for (k = 0; k < bits_nr; k++) {
  interval = rohc_f_32bits(v_ref, k, p);
  if (interval.min <= interval.max) {
    if (v >= interval.min && v <= interval.max) {
      break;
    }
  }
  else {
    if (v >= interval.min || v <= interval.max) {
      break;
    }
  }
}
```

  - ASIP Designer’s C-Compiler uses software pipelining to take advantage of instruction-level parallelism available in the architecture

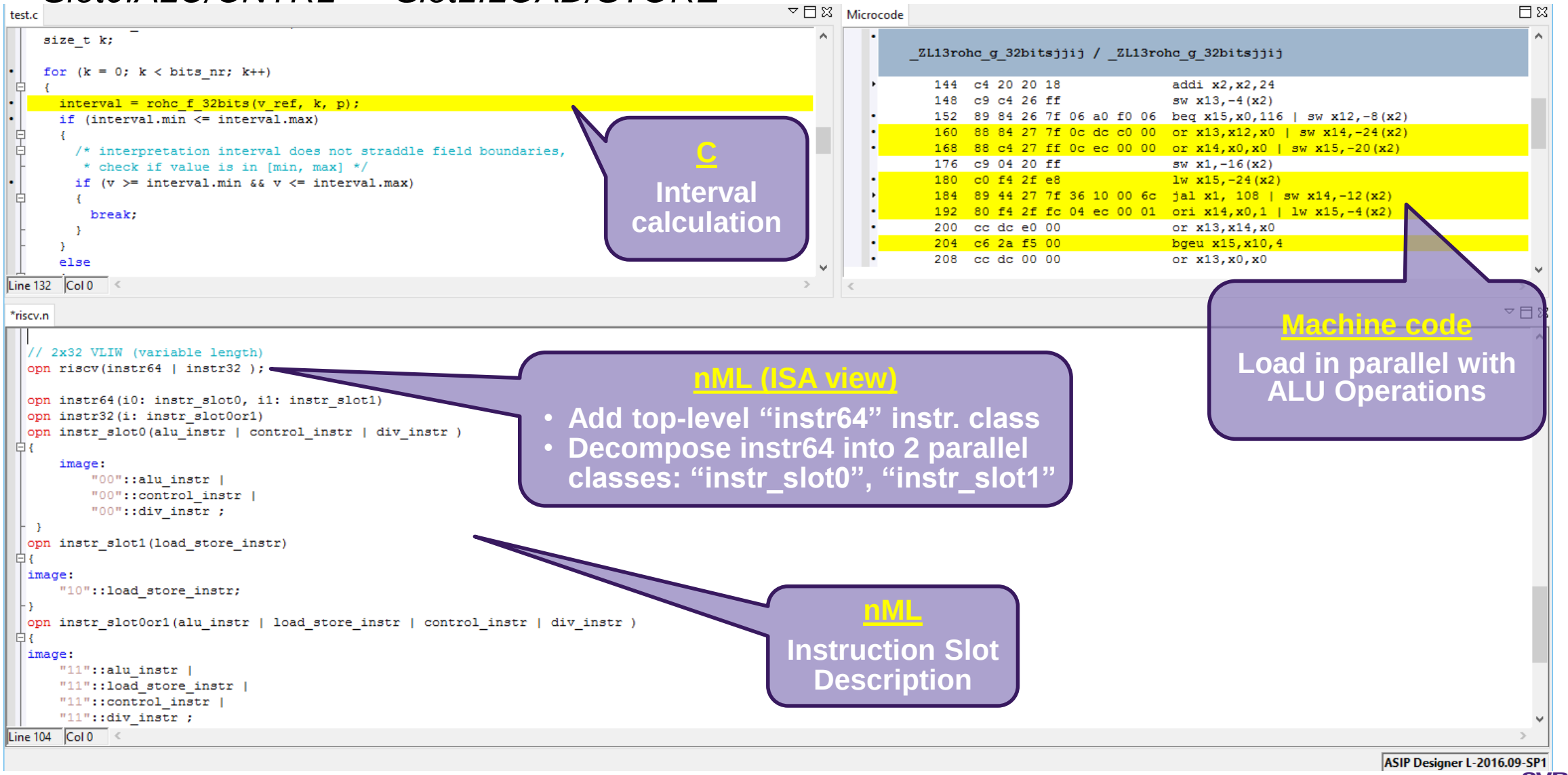
- Maybe switching to a 2-slot VLIW will help
  - Slot0: arithmetic/control instructions
  - Slot1: ld/st instructions
- Easy implementation in ASIP Designer model:
  - Compose 2 slot instruction out of pre-existing instructions

```
// 2x32 VLIW (variable length)
opn riscv(instr64 | instr32 );

opn instr64(i0: instr_slot0, i1: instr_slot1)

opn instr32(i: instr_slot0or1)

opn instr_slot0(alu_instr | control_instr | div_instr )
{
    image:
        "00"::alu_instr |
        "00"::control_instr |
        "00"::div_instr ;
 }
opn instr_slot1(load_store_instr)
{
     image:
        "10"::load_store_instr;
}
```


SYNOPSYS®

Instruction Level Parallelism – 2 Slot VLIW

Slot0:ALU/CNTRL Slot1:LOAD/STORE



SYNOPSYS®

Instruction Level Parallelism

Results: Adding 2-slot VLIW (Benchmark: 6-packet test program)

WLSB	Tzscale	Tzcale+VLIW	
Code size (bytes)	452	456	+1%
Cycle count	1516	1257	-21%
Gate count (28nm HPM @ 500MHz)	24.5K	32.1K	+31%

Results:

- 210 cycles/packet (total Encoder+Context+CRC budget: ~400 cycles/packet)

Effort:

- A few lines of nML code (leveraging pre-existing RISC-V example model) and a few minutes re-running benchmark
- No changes to C code

2nd Iteration: Consider “Some Twist”

rohc_f32 Application-Specific Instruction

- The RoHC WLSB encoder uses a lot of alu functions mixed with control code
  - This can be implemented in a single instruction which could run in hardware in a single cycle

```
unsigned rohc_f_32bits(unsigned old, unsigned v_ref,
unsigned v, unsigned k, int p, unsigned& mask)
{
  unsigned ivmin,ivmax;
  unsigned r = old;
  rohc_f_true32bits(v_ref, k, p, ivmin, ivmax, mask);
  if (ivmin <= ivmax) {
    if (v >= ivmin && v <= ivmax) {
      r=k;
    }
  }
  else {
    if (v >= ivmin || v <= ivmax) {
      r=k;
    }
  }

  mask = (mask<<1)|1;

  return k<old?r:old;
}
```

Step 1: Add rohc_f32 to ISA in nML:

```
opn rohc_f32(s0: mR1,d0:mR1) {
    action {
      stage DE:
         PD = aluC_ls = add (aluA_ls=rohc_s0=s0,aluB_ls=1) @alu;
         R3 = r3_w = rohc_f32(r3_r=R3, r4_r=R4, r5_r=R5, rohc_s,
                              r6_r=R6, r7_r=R7, R7=r7_w) @rohc;
      stage WB:
         d0 = PD;

    }
    syntax : "rohc_f32 (x3,x7,"d0"),(x3,x4,x5,x6,x7," s0")";
    image  : opc.custom_1::s0::d0::"xxxxxxxxxxxxxxx";
}
```

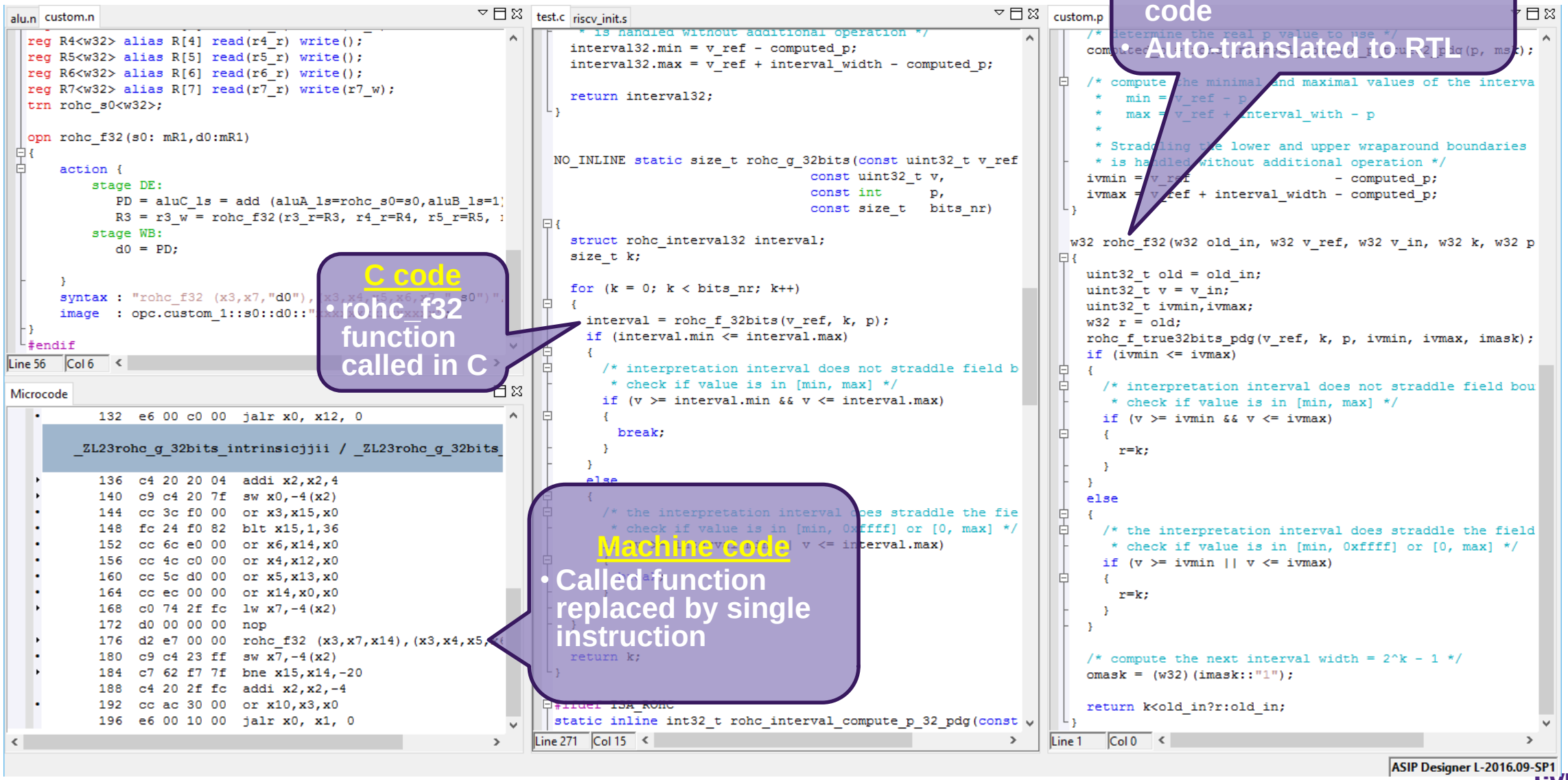
Step 2: Describe rohc_f32 behavior in nML:

```
w32 rohc_f32(w32 old_in, w32 v_ref, w32 v_in, w32 k, w32
p, w32 imask, w32& omask)
{
  uint32_t old = old_in;
  uint32_t v = v_in;
  uint32_t ivmin,ivmax;
  w32 r = old;
  rohc_f_true32bits_pdg(v_ref, k, p, ivmin, ivmax, imask);
  if (ivmin <= ivmax) {
    if (v >= ivmin && v <= ivmax) {
      r=k;
    }
  }
  else { …
```


SYNOPSYS®

Accelerated Data Processing

WLSB Encoder: rohc_f32 Instruction



SYNOPSYS®

Accelerated Data Processing

Results: Adding Application-Specific Instruction (Benchmark: 6-packet test program)

	Tzscale	Tzscale + rohc_f32	
Code size (bytes)	452	200	-56%
Cycle count	1516	502	-67%
Gate count (28nm HPM @ 500MHz)	24.5K	26.8K	+9%

Results:

- 84 cycles/packet (total Encoder+Context+CRC budget: ~400 cycles/packet)

Effort:

- A few more lines of nML code and a few more minutes re-running benchmark
- Simple change to C code to call intrinsic instead of C-function

Accelerated Data Processing + VLIW

Results: Adding Application-Specific Instruction + 2-slot VLIW

	Tzscale	Tzscale + rohc_f32 + VLIW	
Code size (bytes)	452	208	-54%
Cycle count	1516	402	-73%
Gate count (28nm HPM @ 500MHz)	24.5K	35.8K	+46%

Results:

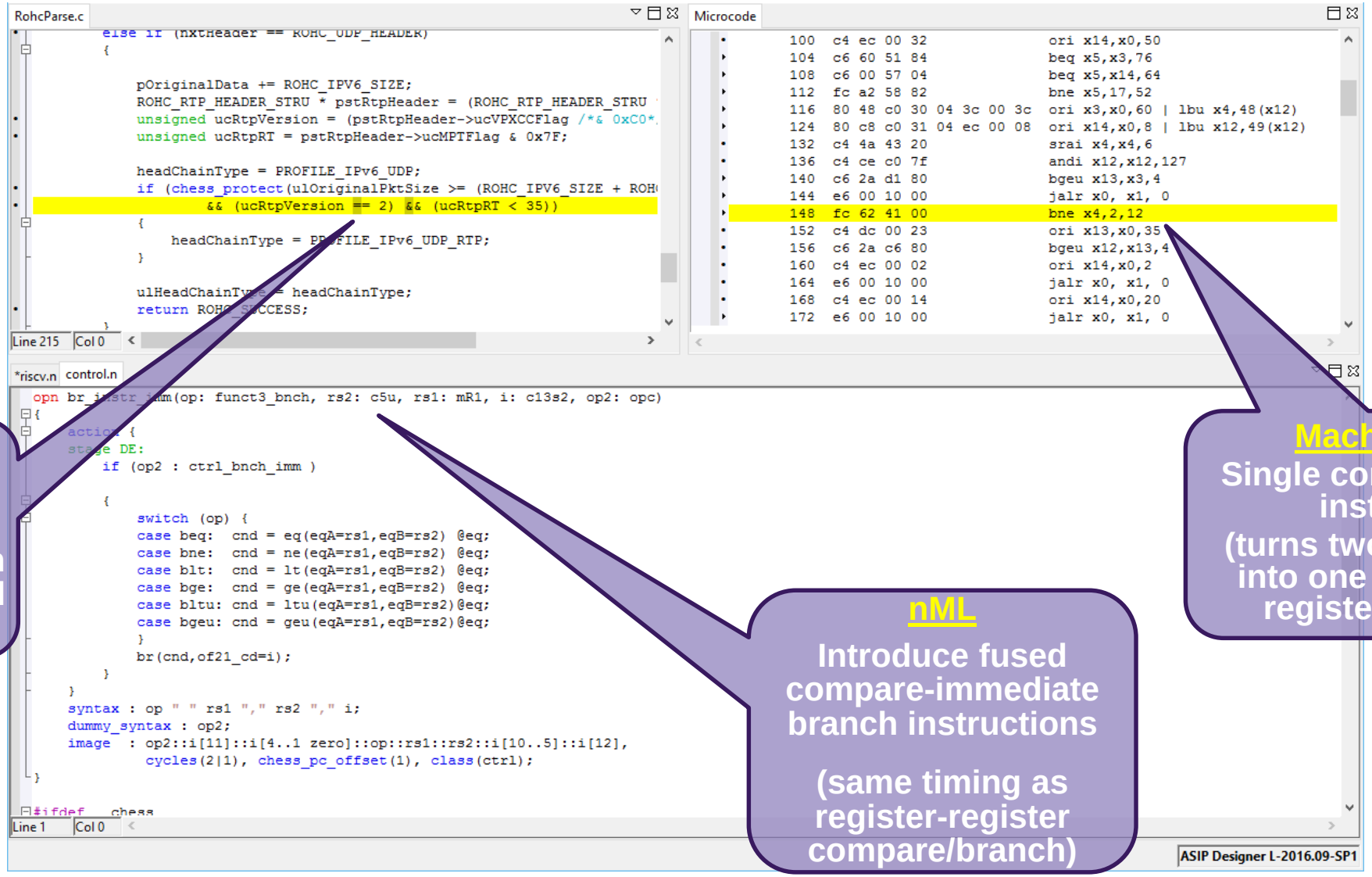
- 67 cycles/packet (total Encoder+Context+CRC budget: ~400 cycles/packet)

Effort:

- Combine both features in the nML code and re-run the benchmark
- No incremental changes to the C-code

3rd Iteration: Look at Header Parser Code

Consider a compare immediate instruction (for small constants) to accelerate control code



SYNOPSYS®

Accelerated Control Processing + VLIW

Results: Header Parser (Benchmark: 4 packet test program)

	Tzscale + rohc_f32 + VLIW	Tzscale + bne_imm + rohc_f32 + VLIW	
Code size	892	824	-8%
Cycle count	289	242	-16%
Gate count (28nm HPM @ 500MHz)	35.8K	35.9K	0%

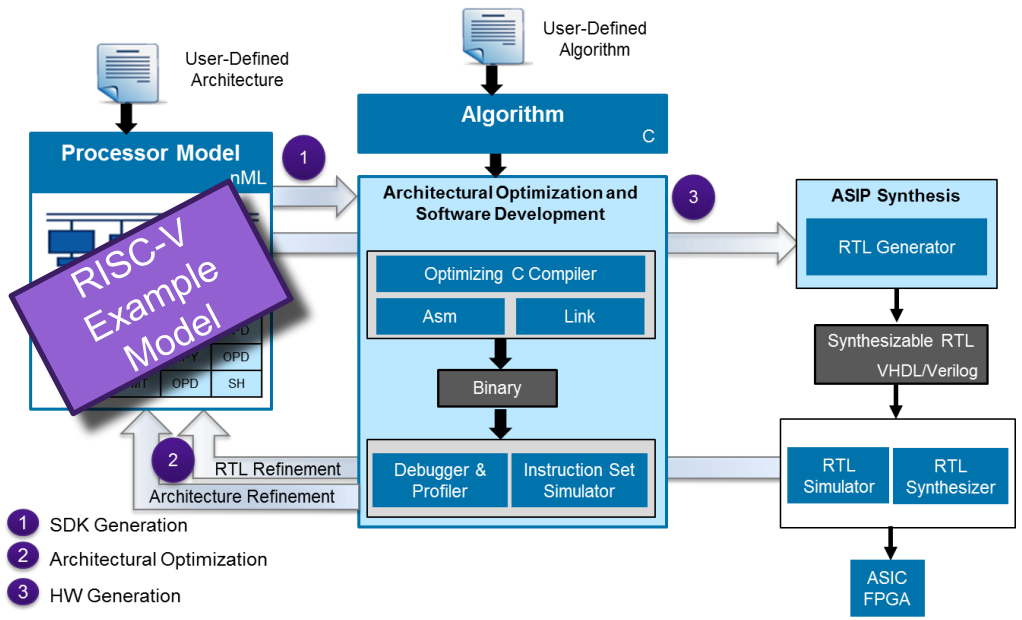
Results:

- 61 cycles/packet (total Header Parser budget: ~100 cycles/packet)

Effort:

- A few more lines of nML code and a few more minutes re-running benchmark
- No changes to the C-code

Take-Aways



- ASIPs contain application-specific architectural optimizations
- ASIP Designer automates design process
- ASIP Designer's RISC-V example model provides a useful jump start

```
start riscv;

opn riscv (bit32_instr | bit16_instr);

opn bit32_instr(alu_instr | load_store_instr | control_instr | div_instr );
opn alu_instr(alu_rrr | alu_rri | alu_lui);
opn alu_rrr(alu_rrr_ar | alu_rrr_sh | alu_rrr_mul | alu_rrr_sub | alu_rrr_sra);

opn alu_rrr_ar(op: funct3_rrr, rd: mR1, rs1: mR1, rs2: mR2) {
  action {
    stage DE:
      switch (op) {
        case add  : aluC = add (aluA=rs1,aluB=rs2) @alu;
        case slt  : aluC = slt (aluA=rs1,aluB=rs2) @alu;
        case sltu : aluC = sltu(aluA=rs1,aluB=rs2) @alu;
        case xor  : aluC = bxor(aluA=rs1,aluB=rs2) @alu;
        case or   : aluC = bor (aluA=rs1,aluB=rs2) @alu;
        case and  : aluC = band(aluA=rs1,aluB=rs2) @alu;
      }
      PD = aluC;
    stage WB:
      rd = PD;
    }
    syntax : op " " rd "," rs1 "," rs2;
    image  : opc.alu_rrr::rd::op::rs1::rs2::funct7_rri.value_0x00;
}
```

- Options can be evaluated swiftly
  - Architectural enhancements
  - Implementation-specific RISC-V characteristics
  - Extensions to RISC-V ISA
- For each alternative ASIP Designer automatically generates a production-level C/C++ compiler, simulator, debugger, RTL model – all in-sync by construction

SYNOPSYS®

SYNOPSYS®
Silicon to Software™

Thank You

Steve Cox (scox@synopsys.com)
Drew Taussig (dtaussig@synopsys.com)