# Tutorial 7
# Tutorial on RISC-V Design and Verification

**Kevin McDermott** - *Imperas Software Ltd.*

**Zdenek Prikryl** - *Codasip Ltd.*

**Peter Shields** - *UltraSoC Technologies Ltd.*





# Tutorial on RISC-V Design and Verification

- Speakers:
  - Kevin McDermott - Imperas Software Ltd:
  - Zdenek Prikryl - Codasip Ltd.
  - Peter Shields - UltraSoC Technologies Ltd.

- RISC-V Tutorial overview
  1. Introduction to RISC-V ISA & The RISC-V Foundation: ISA Freedom & innovation
  2. Imperas: adding RISC-V custom instructions for software development
  3. Codasip: hardware design flow for RISC-V IP core and extensions
  4. UltraSoC: on-chip Analytics for SoC, and heterogeneous architectures

# Introduction to RISC-V ISA

- https://riscv.org

- RISC-V (pronounced “risk-five”) is an open, free ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation.

# RISC-V Background

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, it was time for the Computer Science team at UC Berkeley to look at what ISAs to use for their next set of projects
- Obvious choices: x86 and ARM
  - x86 impossible – too complex, IP issues
  - ARM mostly impossible – complex, IP issues
- So UC Berkeley started “3-month project” during the summer of 2010 to develop their own clean-slate ISA

# RISC-V Background (cont'd)

- Four years later, in May of 2014, UC Berkeley released frozen base user spec
  - many tapeouts and several research publications along the way
- The name RISC-V (pronounced *risk-five*), was chosen to represent the fifth major RISC ISA design effort at UC Berkeley
  - RISC-I, RISC-II, SOAR, and SPUR were the first four projects with the original RISC-I publications dating back to 1981
- In August 2015, articles of incorporation were filed to create a non-profit RISC-V Foundation to govern the ISA

# What's Different about RISC-V?

- ***Simple***
  - Far smaller than other commercial ISAs
- ***Clean-slate design***
  - Clear separation between user and privileged ISA
  - Avoids µarchitecture or technology-dependent features
- A ***modular*** ISA
  - Small standard base ISA
  - Multiple standard extensions
- Designed for ***extensibility/specialization***
  - Variable-length instruction encoding
  - Vast opcode space available for instruction-set extensions
- ***Stable***
  - Base and standard extensions are frozen
  - Additions via optional extensions, not new versions

# RISC-V Base Plus Standard Extensions

- Four base integer ISAs
  - RV32E, RV32I, RV64I, RV128I
  - Only <50 hardware instructions needed for base
- Standard extensions
  - M: Integer multiply/divide
  - A: Atomic memory operations (AMOs + LR/SC)
  - F: Single-precision floating-point
  - D: Double-precision floating-point
  - G = IMAFD, “General-purpose” ISA
  - Q: Quad-precision floating-point
  - C: compressed 16b encodings for 32b instructions
- All the above are a fairly standard RISC encoding in a fixed 32-bit instruction format

# RV32I / RV64I / RV128I

RISC-V ①

② ③ **RISC-V Reference Card** ④

| Base Integer Instructions (32\|64\|128) | | | |
|---|---|---|---|
| **Category** | Name | Fmt | RV{32\|64\|128}I Base |
| **Loads** | Load Byte | I | LB rd,rs1,imm |
| | Load Halfword | I | LH rd,rs1,imm |
| | Load Word | I | L{W\|D\|Q} rd,rs1,imm |
| | Load Byte Unsigned | I | LBU rd,rs1,imm |
| | Load Half Unsigned | I | L{H\|W\|D}U rd,rs1,imm |
| **Stores** | Store Byte | S | SB rs1,rs2,imm |
| | Store Halfword | S | SH rs1,rs2,imm |
| | Store Word | S | S{W\|D\|Q} rs1,rs2,imm |
| **Shifts** | Shift Left | R | SLL{\|W\|D} rd,rs1,rs2 |
| | Shift Left Immediate | I | SLLI{\|W\|D} rd,rs1,shamt |
| | Shift Right | R | SRL{\|W\|D} rd,rs1,rs2 |
| | Shift Right Immediate | I | SRLI{\|W\|D} rd,rs1,shamt |
| | Shift Right Arithmetic | R | SRA{\|W\|D} rd,rs1,rs2 |
| | Shift Right Arith Imm | I | SRAI{\|W\|D} rd,rs1,shamt |
| **Arithmetic** | ADD | R | ADD{\|W\|D} rd,rs1,rs2 |
| | ADD Immediate | I | ADDI{\|W\|D} rd,rs1,imm |
| | SUBtract | R | SUB{\|W\|D} rd,rs1,rs2 |
| | Load Upper Imm | U | LUI rd,imm |
| | Add Upper Imm to PC | U | AUIPC rd,imm |
| **Logical** | XOR | R | XOR rd,rs1,rs2 |
| | XOR Immediate | I | XORI rd,rs1,imm |
| | OR | R | OR rd,rs1,rs2 |
| | OR Immediate | I | ORI rd,rs1,imm |
| | AND | R | AND rd,rs1,rs2 |
| | AND Immediate | I | ANDI rd,rs1,imm |
| **Compare** | Set < | R | SLT rd,rs1,rs2 |
| | Set < Immediate | I | SLTI rd,rs1,imm |
| | Set < Unsigned | R | SLTU rd,rs1,rs2 |
| | Set < Imm Unsigned | I | SLTIU rd,rs1,imm |
| **Branches** | Branch = | SB | BEQ rs1,rs2,imm |
| | Branch ≠ | SB | BNE rs1,rs2,imm |
| | Branch < | SB | BLT rs1,rs2,imm |
| | Branch ≥ | SB | BGE rs1,rs2,imm |
| | Branch < Unsigned | SB | BLTU rs1,rs2,imm |
| | Branch ≥ Unsigned | SB | BGEU rs1,rs2,imm |
| **Jump & Link** | J&L | UJ | JAL rd,imm |
| | Jump & Link Register | I | JALR rd,rs1,imm |
| **Synch** | Synch thread | I | FENCE |
| | Synch Instr & Data | I | FENCE.I |
| **System** | System CALL | I | SCALL |
| | System BREAK | I | SBREAK |
| **Counters** | ReaD CYCLE | I | RDCYCLE rd |
| | ReaD CYCLE upper Half | I | RDCYCLEH rd |
| | ReaD TIME | I | RDTIME rd |
| | ReaD TIME upper Half | I | RDTIMEH rd |
| | ReaD INSTR RETired | I | RDINSTRET rd |
| | ReaD INSTR upper Half | I | RDINSTRETH rd |

+14
Privileged

+ 8 for M

+ 34
for F, D, Q

+ 46 for C

+ 11 for A

*32-bit Instruction Formats*

| | 31 | 30 | 25 24 | 21 | 20 | 19 | 15 14 | 12 11 | 8 | 7 | 6 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **R** | funct7 | | | rs2 | | rs1 | funct3 | rd | | | opcode | |
| **I** | imm[11:0] | | | | | rs1 | funct3 | rd | | | opcode | |
| **S** | imm[11:5] | | | rs2 | | rs1 | funct3 | imm[4:0] | | | opcode | |
| **SB** | imm[12] | imm[10:5] | | rs2 | | rs1 | funct3 | imm[4:1] | | imm[11] | opcode | |
| **U** | imm[31:12] | | | | | | | rd | | | opcode | |
| **UJ** | imm[20] | imm[10:1] | | | imm[11] | imm[19:12] | | rd | | | opcode | |





accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# RV32I / RV64I / RV128I + M, A, F, D, Q, C

RISC-V ① ② ③ **RISC-V Reference Card** ④

## Base Integer Instructions (32|64|128)

| Category | Name | Fmt | RV{32\|64\|128}I Base | |
|---|---|---|---|---|
| **Loads** | Load Byte | I | LB | rd,rs1,imm |
| | Load Halfword | I | LH | rd,rs1,imm |
| | Load Word | I | L{W\|D\|Q} | rd,rs1,imm |
| | Load Byte Unsigned | I | LBU | rd,rs1,imm |
| | Load Half Unsigned | I | L{H\|W\|D}U | rd,rs1,imm |
| **Stores** | Store Byte | S | SB | rs1,rs2,imm |
| | Store Halfword | S | SH | rs1,rs2,imm |
| | Store Word | S | S{W\|D\|Q} | rs1,rs2,imm |
| **Shifts** | Shift Left | R | SLL{\|W\|D} | rd,rs1,rs2 |
| | Shift Left Immediate | I | SLLI{\|W\|D} | rd,rs1,shamt |
| | Shift Right | R | SRL{\|W\|D} | rd,rs1,rs2 |
| | Shift Right Immediate | I | SRLI{\|W\|D} | rd,rs1,shamt |
| | Shift Right Arithmetic | R | SRA{\|W\|D} | rd,rs1,rs2 |
| | Shift Right Arith Imm | I | SRAI{\|W\|D} | rd,rs1,shamt |
| **Arithmetic** | ADD | R | ADD{\|W\|D} | rd,rs1,rs2 |
| | ADD Immediate | I | ADDI{\|W\|D} | rd,rs1,imm |
| | SUBtract | R | SUB{\|W\|D} | rd,rs1,rs2 |
| | Load Upper Imm | U | LUI | rd,imm |
| | Add Upper Imm to PC | U | AUIPC | rd,imm |
| **Logical** | XOR | R | XOR | rd,rs1,rs2 |
| | XOR Immediate | I | XORI | rd,rs1,imm |
| | OR | R | OR | rd,rs1,rs2 |
| | OR Immediate | I | ORI | rd,rs1,imm |
| | AND | R | AND | rd,rs1,rs2 |
| | AND Immediate | I | ANDI | rd,rs1,imm |
| **Compare** | Set < | R | SLT | rd,rs1,rs2 |
| | Set < Immediate | I | SLTI | rd,rs1,imm |
| | Set < Unsigned | R | SLTU | rd,rs1,rs2 |
| | Set < Imm Unsigned | I | SLTIU | rd,rs1,imm |
| **Branches** | Branch = | SB | BEQ | rs1,rs2,imm |
| | Branch ≠ | SB | BNE | rs1,rs2,imm |
| | Branch < | SB | BLT | rs1,rs2,imm |
| | Branch ≥ | SB | BGE | rs1,rs2,imm |
| | Branch < Unsigned | SB | BLTU | rs1,rs2,imm |
| | Branch ≥ Unsigned | SB | BGEU | rs1,rs2,imm |
| **Jump & Link** | J&L | UJ | JAL | rd,imm |
| | Jump & Link Register | I | JALR | rd,rs1,imm |
| **Synch** | Synch thread | I | FENCE | |
| | Synch Instr & Data | I | FENCE.I | |
| **System** | System CALL | I | SCALL | |
| | System BREAK | I | SBREAK | |
| **Counters** | ReaD CYCLE | I | RDCYCLE | rd |
| | ReaD CYCLE upper Half | I | RDCYCLEH | rd |
| | ReaD TIME | I | RDTIME | rd |
| | ReaD TIME upper Half | I | RDTIMEH | rd |
| | ReaD INSTR RETired | I | RDINSTRET | rd |
| | ReaD INSTR upper Half | I | RDINSTRETH | rd |

## RV Privileged Instructions (32|64|128)

| Category | Name | Fmt | RV mnemonic | |
|---|---|---|---|---|
| **CSR Access** | Atomic R/W | R | CSRRW | rd,csr,rs1 |
| | Atomic Read & Set Bit | R | CSRRS | rd,csr,rs1 |
| | Atomic Read & Clear Bit | R | CSRRC | rd,csr,rs1 |
| | Atomic R/W Imm | R | CSRRWI | rd,csr,imm |
| | Atomic Read & Set Bit Imm | R | CSRRSI | rd,csr,imm |
| | Atomic Read & Clear Bit Imm | R | CSRRCI | rd,csr,imm |
| **Change Level** | Env. Call | R | ECALL | |
| | Environment Breakpoint | R | EBREAK | |
| | Environment Return | R | ERET | |
| **Trap Redirect** | to Superviso | R | MRTS | |
| | Redirect Trap to Hypervisor | R | MRTH | |
| | Hypervisor Trap to Supervisor | R | HRTS | |
| **Interrupt** | Wait for Interrup | R | WFI | |
| **MMU** | Supervisor FENCE | R | SFENCE.VM | rs1 |

## Optional Multiply-Divide Extension: RV32M

| Category | Name | Fmt | RV32M (Mult-Div) | |
|---|---|---|---|---|
| **Multiply** | MULtiply | R | MUL{\|W\|D} | rd,rs1,rs2 |
| | MULtiply upper Half | R | MULH | rd,rs1,rs2 |
| | MULtiply Half Sign/Uns | R | MULHSU | rd,rs1,rs2 |
| | MULtiply upper Half Uns | R | MULHU | rd,rs1,rs2 |
| **Divide** | DIVide | R | DIV{\|W\|D} | rd,rs1,rs2 |
| | DIVide Unsigned | R | DIVU | rd,rs1,rs2 |
| **Remainder** | REMainder | R | REM{\|W\|D} | rd,rs1,rs2 |
| | REMainder Unsigned | R | REMU{\|W\|D} | rd,rs1,rs2 |

## Optional Atomic Instruction Extension: RVA

| Category | Name | Fmt | RV{32\|64\|128}A (Atomic) | |
|---|---|---|---|---|
| **Load** | Load Reserved | R | LR.{W\|D\|Q} | rd,rs1 |
| **Store** | Store Conditiona | R | SC.{W\|D\|Q} | rd,rs1,rs2 |
| **Swap** | SWAP | R | AMOSWAP.{W\|D\|Q} | rd,rs1,rs2 |
| **Add** | ADD | R | AMOADD.{W\|D\|Q} | rd,rs1,rs2 |
| **Logical** | XOR | R | AMOXOR.{W\|D\|Q} | rd,rs1,rs2 |
| | AND | R | AMOAND.{W\|D\|Q} | rd,rs1,rs2 |
| | OR | R | AMOOR.{W\|D\|Q} | rd,rs1,rs2 |
| **Min/Max** | MINimum | R | AMOMIN.{W\|D\|Q} | rd,rs1,rs2 |
| | MAXimum | R | AMOMAX.{W\|D\|Q} | rd,rs1,rs2 |
| | MINimum Unsigned | R | AMOMINU.{W\|D\|Q} | rd,rs1,rs2 |
| | MAXimum Unsigned | R | AMOMAXU.{W\|D\|Q} | rd,rs1,rs2 |

## 3 Optional FP Extensions: RV32{F|D|Q}

| Category | Name | Fmt | RV{F\|D\|Q} (HP/SP,DP,QP) | |
|---|---|---|---|---|
| **Load** | Load | I | FL{W,D,Q} | rd,rs1,imm |
| **Store** | Store | S | FS{W,D,Q} | rs1,rs2,imm |
| **Arithmetic** | ADD | R | FADD.{S\|D\|Q} | rd,rs1,rs2 |
| | SUBtract | R | FSUB.{S\|D\|Q} | rd,rs1,rs2 |
| | MULtiply | R | FMUL.{S\|D\|Q} | rd,rs1,rs2 |
| | DIVide | R | FDIV.{S\|D\|Q} | rd,rs1,rs2 |
| | SQuare RooT | R | FSQRT.{S\|D\|Q} | rd,rs1 |
| **Mul-Add** | Multiply-ADD | R | FMADD.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| | Multiply-SUBtract | R | FMSUB.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| | Negative Multiply-SUBtract | R | FMNSUB.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| | Negative Multiply-ADD | R | FMNADD.{S\|D\|Q} | rd,rs1,rs2,rs3 |
| **Sign Inject** | SiGN source | R | FSGNJ.{S\|D\|Q} | rd,rs1,rs2 |
| | Negative SiGN source | R | FSGNJN.{S\|D\|Q} | rd,rs1,rs2 |
| | Xor SiGN source | R | FSGNJX.{S\|D\|Q} | rd,rs1,rs2 |
| **Min/Max** | MINimum | R | FMIN.{S\|D\|Q} | rd,rs1,rs2 |
| | MAXimum | R | FMAX.{S\|D\|Q} | rd,rs1,rs2 |
| **Compare** | Compare Float = | R | FEQ.{S\|D\|Q} | rd,rs1,rs2 |
| | Compare Float < | R | FLT.{S\|D\|Q} | rd,rs1,rs2 |
| | Compare Float ≤ | R | FLE.{S\|D\|Q} | rd,rs1,rs2 |
| **Categorize** | Classify Typ | R | FCLASS.{S\|D\|Q} | rd,rs1 |
| **Move** | Move from Integer | R | FMV.S.X | rd,rs1 |
| | Move to Integer | R | FMV.X.S | rd,rs1 |
| **Convert** | Convert from Int | R | FCVT.{S\|D\|Q}.W | rd,rs1 |
| | Convert from Int Unsigned | R | FCVT.{S\|D\|Q}.WU | rd,rs1 |
| | Convert to Int | R | FCVT.W.{S\|D\|Q} | rd,rs1 |
| | Convert to Int Unsigned | R | FCVT.WU.{S\|D\|Q} | rd,rs1 |
| **Configuration** | Read Stat | R | FRCSR | rd |
| | Read Rounding Mode | R | FRRM | rd |
| | Read Flags | R | FRFLAGS | rd |
| | Swap Status Reg | R | FSCSR | rd,rs1 |
| | Swap Rounding Mode | R | FSRM | rd,rs1 |
| | Swap Flags | R | FSFLAG | rd,rs1 |
| | Swap Rounding Mode Imm | I | FSRMI | rd,imm |
| | Swap Flags Imm | I | FSFLAGSI | rd,imm |

F, C for

64{F|D|Q}/

128{F|D|Q}

## Optional Compressed Instructions: RVC

| Category | Name | Fmt | RVC | |
|---|---|---|---|---|
| **Loads** | Load Word | CL | C.LW | rd',rs1',imm |
| | Load Word SP | CI | C.LWSP | rd,imm |
| | Load Double | CL | C.LD | rd',rs1',imm |
| | Load Double SP | CI | C.LWSP | rd,imm |
| | Load Quad | CL | C.LQ | rd',rs1',imm |
| | Load Quad SP | CI | C.LQSP | rd,imm |
| | Load Byte Unsigned | CL | C.LBU | rd',rs1',imm |
| | Float Load Word | CL | C.FLW | rd',rs1',imm |
| | Float Load Double | CL | C.FLD | rd',rs1',imm |
| | Float Load Word SP | CI | C.FLWSP | rd,imm |
| | Float Load Double SP | CI | C.FLDSP | rd,imm |
| **Stores** | Store Word | CS | C.SW | rs1',rs2',imm |
| | Store Word SP | CSS | C.SWSP | rs2,imm |
| | Store Double | CS | C.SD | rs1',rs2',imm |
| | Store Double SP | CSS | C.SDSP | rs2,imm |
| | Store Quad | CS | C.SQ | rs1',rs2',imm |
| | Store Quad SP | CSS | C.SQSP | rs2,imm |
| | Float Store Word | CSS | C.FSW | rd',rs1',imm |
| | Float Store Double | CSS | C.FSD | rd',rs1',imm |
| | Float Store Word SP | CSS | C.FSWSP | rd,imm |
| | Float Store Double SP | CSS | C.FSDSP | rd,imm |
| **Arithmetic** | ADD | CR | C.ADD | rd,rs1 |
| | ADD Word | CR | C.ADDW | rd',rs2' |
| | ADD Immediate | CI | C.ADDI | rd,imm |
| | ADD Word Imm | CI | C.ADDIW | rd,imm |
| | ADD SP Imm * 16 | CI | C.ADDI16SP | x0,imm |
| | ADD SP Imm * 4 | CIW | C.ADDI4SPN | rd',imm |
| | Load Immediate | CI | C.LI | rd,imm |
| | Load Upper Imm | CI | C.LUI | rd,imm |
| | MoVe | CR | C.MV | rd,rs1 |
| | SUB | CR | C.SUB | rd',rs2' |
| | SUB Word | CR | C.SUBW | rd',rs2' |
| **Logical** | XOR | CS | C.XOR | rd',rs2' |
| | OR | CS | C.OR | rd',rs2' |
| | AND | CS | C.AND | rd',rs2' |
| | AND Immediate | CB | C.ANDI | rd',rs2' |
| **Shifts** | Shift Left Imm | CI | C.SLLI | rd,imm |
| | Shift Right Immediate | CB | C.SRLI | rd',imm |
| | Shift Right Arith Imm | CB | C.SRAI | rd',imm |
| **Branches** | Branch=0 | CB | C.BEQZ | rs1',imm |
| | Branch≠0 | CB | C.BNEZ | rs1',imm |
| **Jump** | Jump | CJ | C.J | imm |
| | Jump Register | CR | C.JR | rd,rs1 |
| **Jump & Link** | J&L | CJ | C.JAL | imm |
| | Jump & Link Register | CR | C.JALR | rs1 |
| **System** | Env. BREAK | CI | C.EBREAK | |

## 16-bit (RVC) and 32-bit Instruction Formats

| | 15 14 13 | 12 | 11 10 9 | 8 7 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|
| **CI** | funct4 | | rd/rs1 | | rs2 | op |
| **CSS** | funct3 | imm | rd/rs1 | | imm | op |
| **CIW** | funct3 | | imm | | rs2 | op |
| **CL** | funct3 | | imm | | rd′ | op |
| **CS** | funct3 | imm | rs1′ | imm | rd′ | op |
| **CB** | funct3 | imm | rs1′ | imm | rs2′ | op |
| **CJ** | funct3 | offset | rs1′ | offset | | op |
| | funct3 | | jump target | | | op |

| | 31 30 | 25 24 | 21 20 | 19 | 15 14 | 12 11 | 8 7 | 6 0 |
|---|---|---|---|---|---|---|---|---|
| **R** | funct7 | | rs2 | rs1 | funct3 | rd | | opcode |
| **I** | imm[11:0] | | | rs1 | funct3 | rd | | opcode |
| **S** | imm[11:5] | | rs2 | rs1 | funct3 | imm[4:0] | | opcode |
| **SB** | imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode |
| **U** | imm[31:12] | | | | | rd | | opcode |
| **UJ** | imm[20] | imm[10:1] | imm[11] | imm[19:12] | | rd | | opcode |

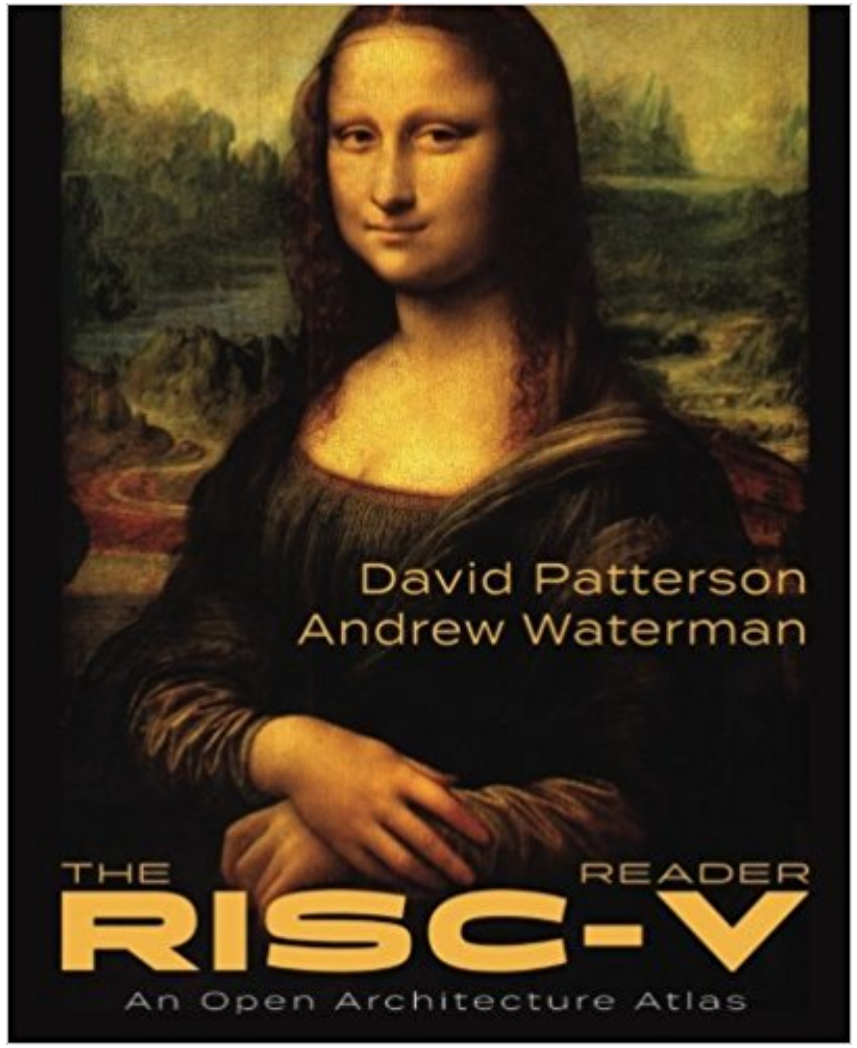


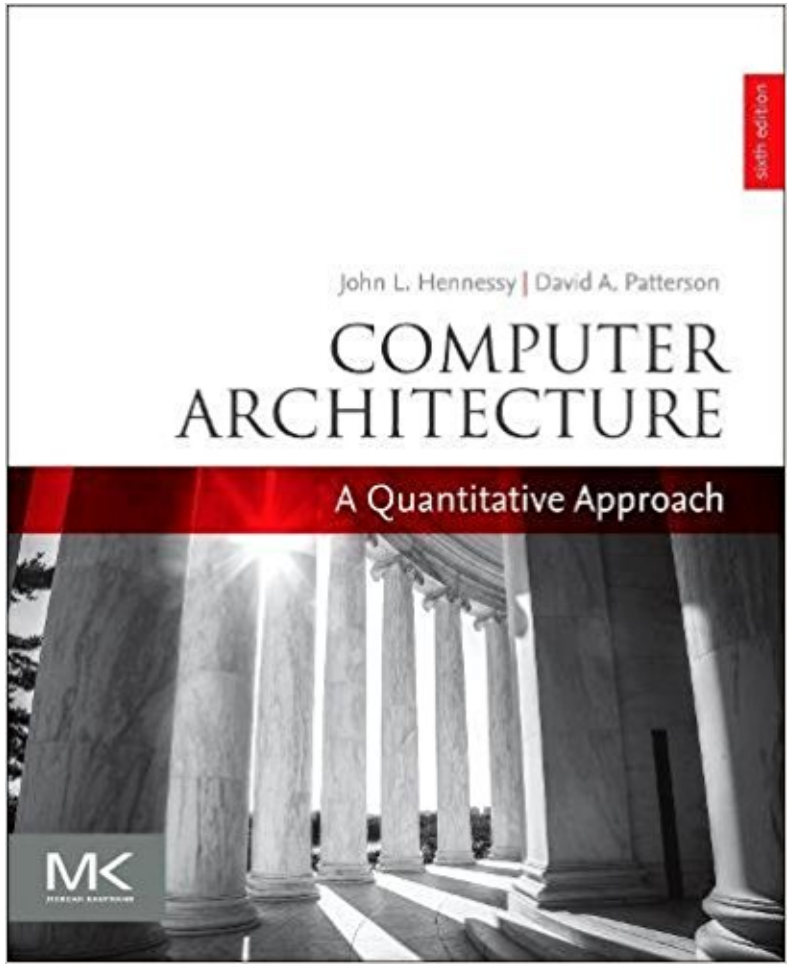
accellera SYSTEMS INITIATIVE



2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# RISC-V in Education, new books!

# RISC-V Foundation Overview

- Incorporated August, 2015 as a 501c6 non-profit Foundation
- Membership Agreement & Bylaws ratified December 2016
- The RISC-V ISA and related standards shall remain open and license-free to all parties
  - RISC-V ISA specifications shall always be publicly available as an online download
- The compliance test suites shall always be publicly available as a source code download
- To protect the standard, only members (with commercial RISC-V products) of the Foundation in good standing can use “RISC-V” and associated trademarks, and only for devices that pass the tests in the open-source compliance suites maintained by the Foundation

# Foundation Organization

- The Board of Directors consists of seven+ members, whose replacements are elected by the membership
- The Board can amend the By-Laws of the RISC-V foundation via a two-thirds affirmative vote
- The Board appoints chairs of ad-hoc committees to address issues concerning RISC-V, and has the final vote of approval of the recommendation of the ad-hoc committees.
  - Technical Committee Chair – Yunsup Lee, SiFive
  - Security Standing Committee Chair - Helena Handschuh, Rambus
  - Marketing Committee Chair – Ted Marena, Western Digital
- All members of committees must be members of the RISC-V Foundation

# RISC-V ISA & Foundation Summary

- The free and open RISC-V ISA is enabling a new innovation frontier for all computing devices
- Strong Industry Support
  - 150+ members; Broad commercial and academic interest
- RISC-V Summit registration is open
  - December 3-6 2018, Santa Clara, CA
- RISC-V Workshop
  - June 2019, Zurich – further details to be announced soon

# RISC-V extended with Custom Instructions, Virtual Platform for Design and Verification

Duncan Graham, Sr. Applications Engineer

Kevin McDermott, VP Marketing

imperas

accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# Imperas Introduction

- Focus on simulation, modeling, tools for embedded software
  - SoC designers that need early software development for design and test
  - Software developers that need platforms before hardware is available
- Founded 2008, based in Thame, UK
- Management Background: Verilog, VCS, Verisity, Exemplar, Arm, MIPS
- Business model based on tools and ecosystem partnerships
- Active members of RISC-V Foundation
  - Technical Task Groups: vector, bitmanip, compliance

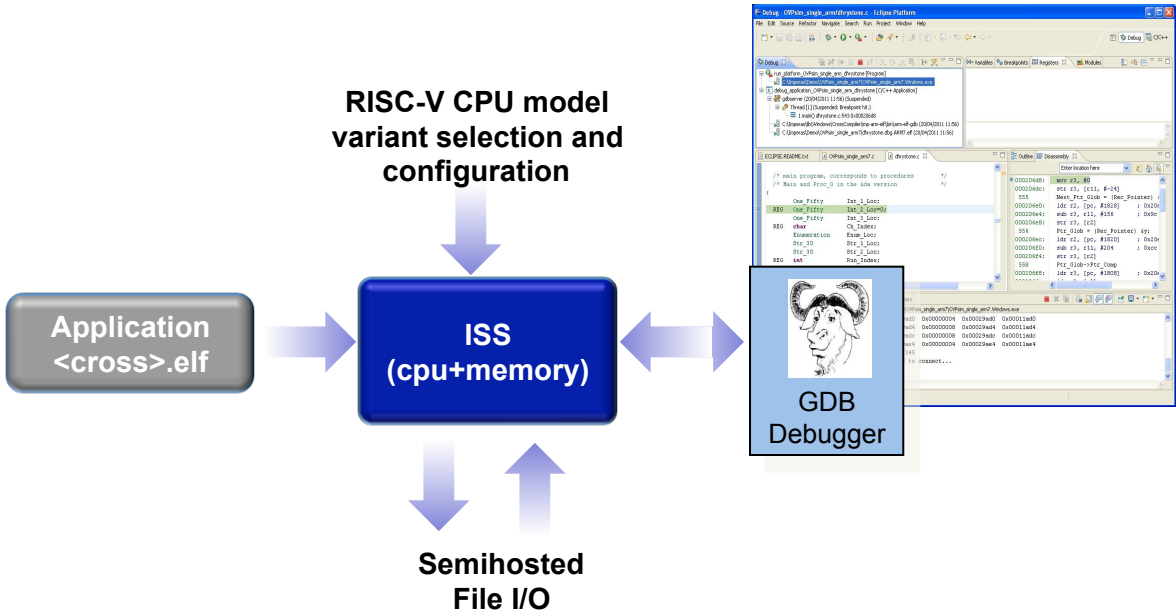
# Most adopters of RISC-V want to add their own custom extension instructions

- Traditional ISA choice has been hard if you want to add your own custom processor instructions to an ISA
- RISC-V as an open standard has specific regions of instruction decode space specifically allocated for users to add their own instructions

- There are multiple issues with the tools that will be needed …
  - You need to evaluate effectiveness and performance gains of new instructions
  - Challenge of how to efficiently and safely add new instructions to existing quality simulation models
  - Need to be able to trace, debug and analyse applications using the new instructions
  - Complete SoC tools covering heterogeneous, multi-core and many-core compute configurations
  - Often need to provide to developers & customers models/platforms without issues of GPL licenses

# In this tutorial…

- Simulator ISS of RISC-V which includes the model + memory
  - Just like the ISS as used in the RISCV.org Compliance Task Group GitHub repository
  - Though in this tutorial we use the Imperas professional simulator which allows model and tool extensions




- Application software is character stream encoder, based on ChaCha20 encryption algorithm
  - Instruction Extensions to RISC-V courtesy of Cerberus Security Laboratories Ltd
  - https://cerberus-laboratories.com
- Tutorial will show adding extension for the model and analysis including timing estimation

# Flow to add new custom instructions

Characterize C Application

- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling

# Flow to add new custom instructions

Characterize C Application

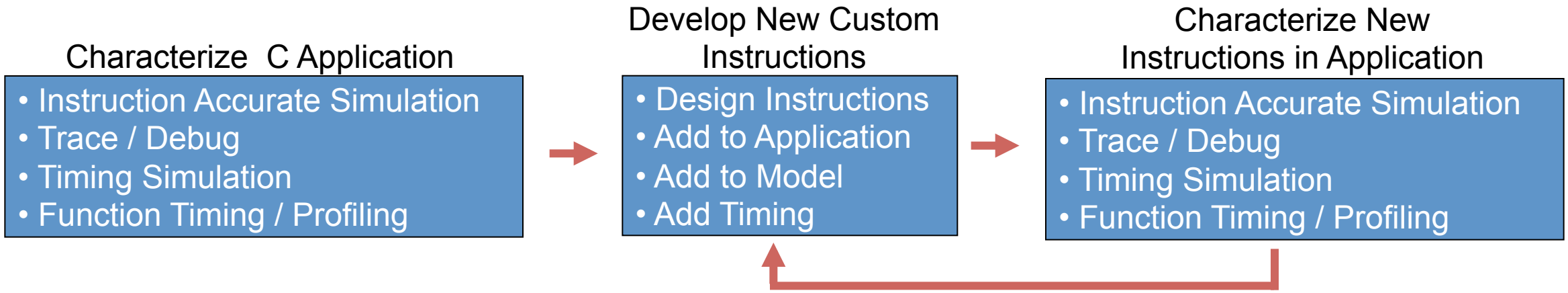
- Instruction Accurate Simulation
- Trace / Debug
- Timing Simulation
- Function Timing / Profiling

→

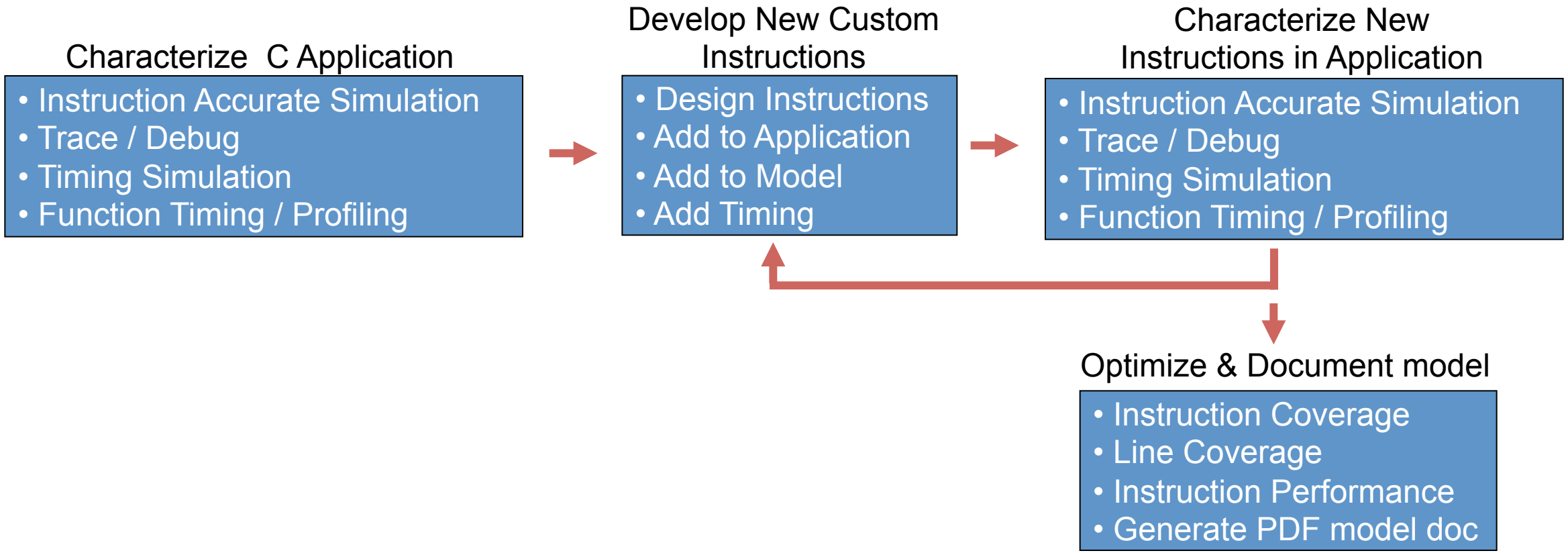
Develop New Custom Instructions

- Design Instructions
- Add to Application
- Add to Model
- Add Timing

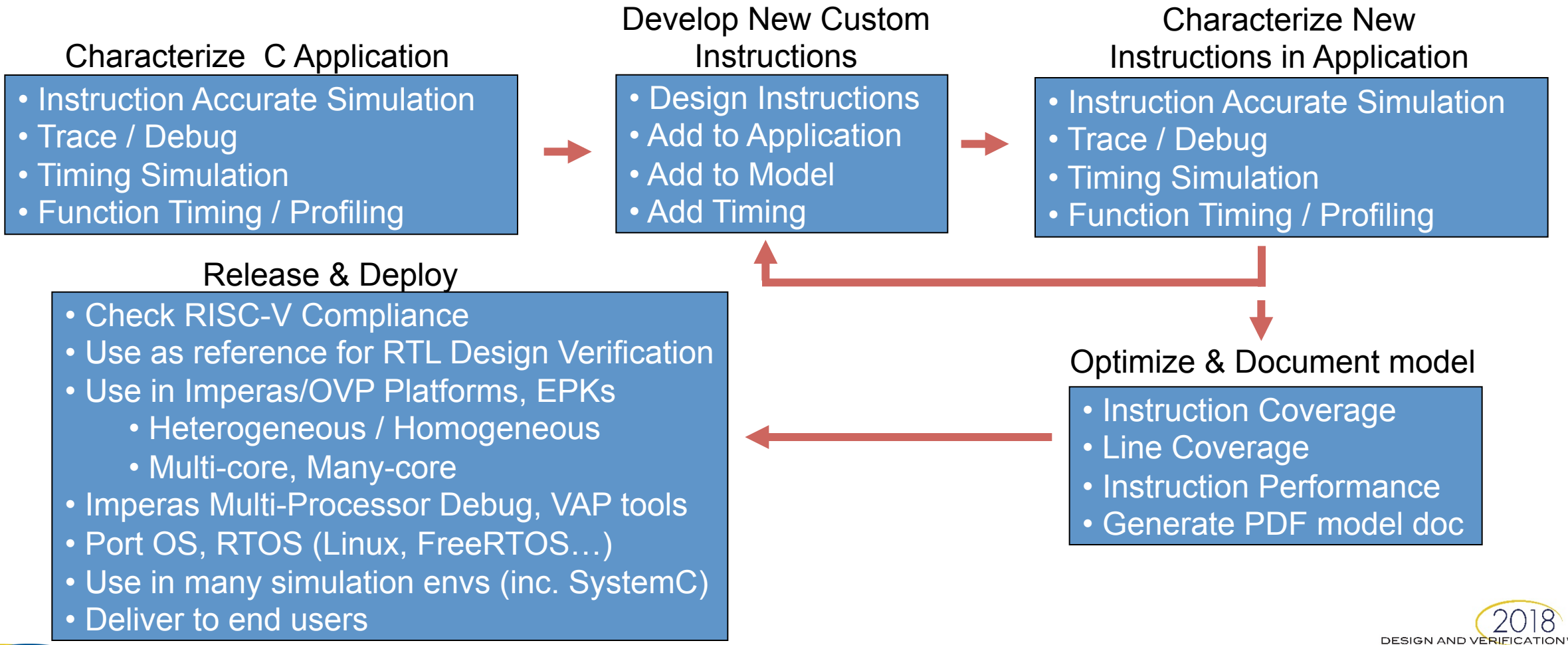
# Flow to add new custom instructions



# Flow to add new custom instructions



# Flow to add new custom instructions



# Checklist and Tasks

- Instruction Accurate simulation of C application
- Cycle Approximate simulation of C application
- Profile the C application
- Add custom instructions to application
- Add custom instructions to model
- Cycle Approximate simulation including custom instructions
- Profile custom instructions application
- Trace custom instructions
- Debug custom instructions
- Documenting custom instructions
- Further tools for model developers

# Instruction Accurate simulation C application

- Cross compiled C application targeting RV32IM
  - Character stream encoder, with ChaCha20 encryption algorithm
- IA simulation
  - Imperas RISC-V ISS with configurable model of RISC-V specification selecting RV32IM
- Semihosting
  - Enables bare metal application to very simply access host I/O

➢ runs fast
  - Over 1 billion instructions a second (standard PC)
    - Linux and Windows supported host OS

test_c.c

```c
unsigned int processLine(unsigned int res, unsigned int word){
    res = qr1_c(res, word);
    res = qr2_c(res, word);
    res = qr3_c(res, word);
    res = qr4_c(res, word);
    res = qr1_c(res, word);
    res = qr2_c(res, word);
    res = qr3_c(res, word);
    res = qr4_c(res, word);
    return res;
}

int main(void) {
    const char *customData = "application/custom.data";
    FILE *fp = fopen(customData, "r");
    if (fp) {
        unsigned int res = 0x84772366;
        unsigned int word;
        unsigned int cnt=0;
        unsigned int iter=0;
        while (iter++ < 16) {
            while (fread(&word,sizeof(unsigned int), 1, fp)) {
                res = processLine(res, word);
            }
            rewind(fp);
        }
        fclose(fp);
        printf("RES = %08X\n", r
    } else {
        printf("Failed to open "
    }

    return 0;
}
```

```
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 11:19:21 2018

Info (OR_OF) Target 'iss/cpu0' has object file read from 'application/test_c.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 0x000173c8 0x000173c8 R-E   1000
Info (OR_PD) LOAD           0x000173c8 0x000283c8 0x000283c8 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
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: 1,289,380,976
Info   Simulated MIPS      : 1151.2
Info ----------------------------------------------------------------
Info
Info ----------------------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time      : 12.89 seconds
Info   User time           : 1.10 seconds
Info   System time         : 0.02 seconds
Info   Elapsed time        : 1.14 seconds
Info   Real time ratio     : 11.31x faster
Info ----------------------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:19:22 2018
```

# Profile C Application

- Same C application
- IA simulation + timing annotation
- With sampled profiling with call stack analysis

➢ Shows cycle approximate timing of each application function

| Name (location) | Arcs in | Samples In | Arcs out | From/To this | Percentage (%) |
|---|---|---|---|---|---|
| Platform: iss | | | | | |
| Processor: iss/cpu0 | | | | | |
| Process: 0_None | | 1659204277 | | | |
| _fread_r | 598189652 | 596534872 | 1654780 | | 35.95% |
| processLine | 925852930 | 354236017 | 571616913 | | 21.35% |
| qr4_c | 150627639 | 150627639 | 0 | | 9.08% |
| qr1_c | 146083640 | 146083640 | 0 | | 8.8% |
| qr2_c | 137682652 | 137682652 | 0 | | 8.3% |
| qr3_c | 137222982 | 137222982 | 0 | | 8.27% |
| __libc_init_array | 0 | 135154865 | 1524049412 | | 8.15% |
| __srefill_r | 1654780 | 1024985 | 629795 | | 0.06% |
| __sread | 629637 | 321116 | 308521 | | 0.02% |
| _read_r | 308521 | 308521 | 0 | | 0.02% |
| _fseeko_r | 2706 | 2126 | 580 | | 0.0% |
| _vfprintf_r | 1874 | 764 | 1110 | | 0.0% |
| __sfvwrite_r | 848 | 752 | 96 | | 0.0% |
| rewind | 3267 | 561 | 2706 | | 0.0% |
| _close_r | 357 | 357 | 0 | | 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 | 734 | 204 | 530 | | 0.0% |

# Cycle Approximate simulation including custom instructions

- IA simulation + timing annotation + custom instructions
  - Includes timing estimation for RV32IM processor
  - Need to add timing estimation for new custom instructions
- Simulate using C code application with inline assembler of custom extensions
- IA simulator + timing tool + custom extension instruction library

➢ See estimated improvement in throughput of application on new processor

test_c.c | test_custom.c | customChaCha20.c | riscv32.c

```
            case IC_mdu_mul : {
                cycles = 5;         // Specify cycles for instruction group
                break;
            }
            case IC_mdu_div : {
                cycles = 16;        // Specify cycles for instruction group
                break;
            }
            case IC_custom : {
                                    // chacha20qr1-chacha20qr4 group same cycles
                cycles = 2;         // Specify cycles for instruction group
                break;
            }
            default: {
                VMI_ABORT("Invalid instructionCLassE value %d (%s)\n", iClass, instrClassName(iClass));
                break;
            }

        }

        if (runtimeCB) { // division run-time callback
            emitCycleEstimation(processor,object,thisPC,regSource1,regSource2,mduMode,iClass,runtimeCB);
        } else {
            addCycleCount(object, thisPC, cycles, iClass);
        }

        // Update previous morph time instruction info
```

```
CpuManagerMulti
Copyright (c) 2
Licensed Softwa
Visit www.IMPER

CpuManagerMulti

Info (OR_OF) Ta
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 (CMD_CC) calling 'iss/cpu0/exTI/cpucycles'
Info (CPUEST_CMD) iss/cpu0: cpucycles on: time stretch enabled
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       : 123.1
Info ----------------------------------------------------------
Info
Info ----------------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 9.21 seconds
Info   User time            : 5.30 seconds
Info   System time          : 0.20 seconds
Info   Elapsed time         : 5.73 seconds
Info   Real time ratio      : 1.61x faster
Info ----------------------------------------------------------

CpuManagerMulti finished: Thu Aug 23 11:58:35 2018


CpuManagerMulti (32-Bit) v99999999 Open Virtual Platform simulator from www.IMPERAS.com,
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

Info (CPUEST_RSLT) Estimated execution time 9.21 seconds, clock cycles 921,006,928
Use script lastRun.sh to re-run with current settings
```

# Profile custom instructions application

- IA simulation + timing annotation + custom instructions with sampled profiling

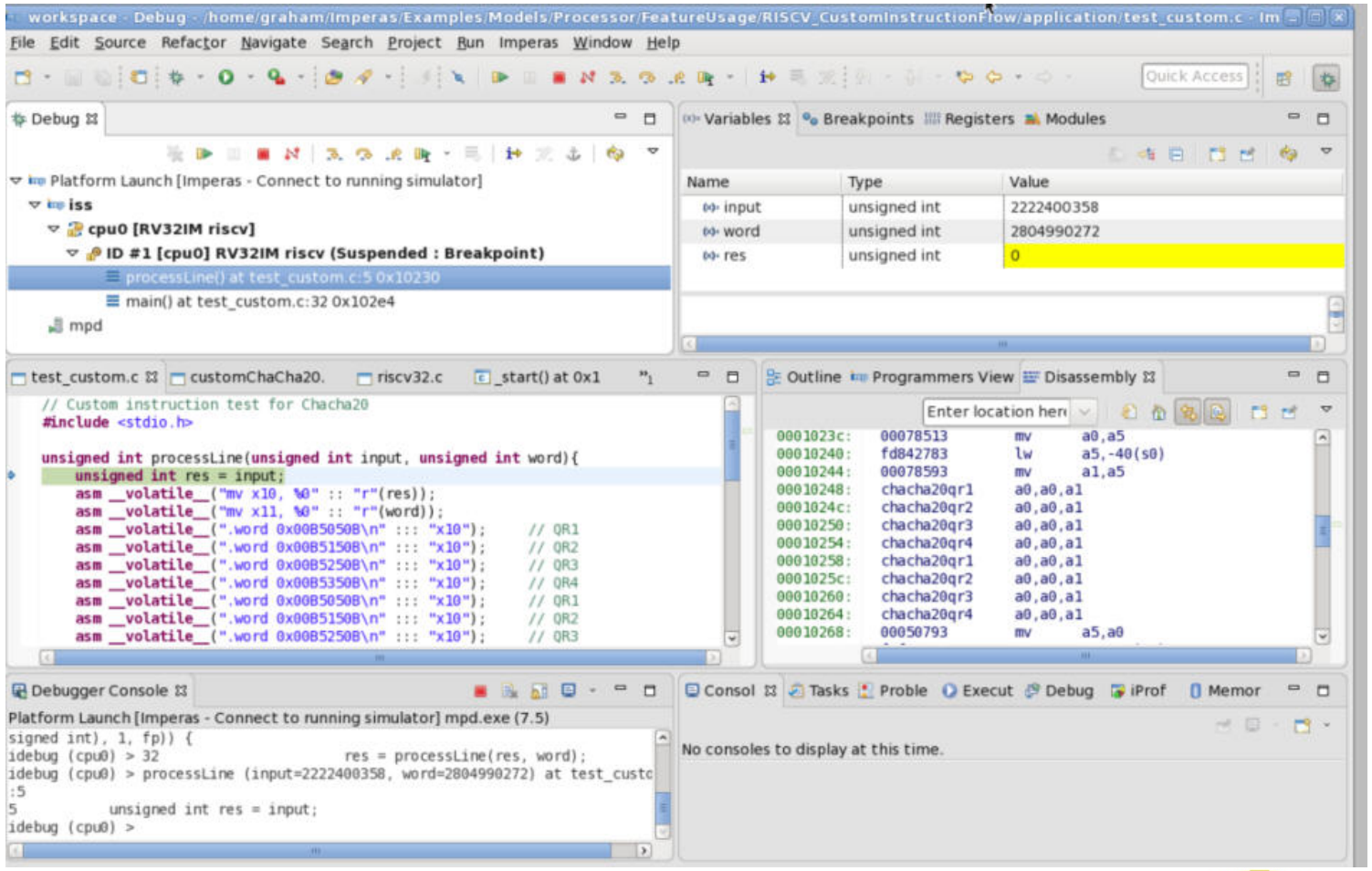
➢ Shows where slowest function is
  – Now much faster…

➢ Shows benefits of using custom instructions

| 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% |

# Debug & Trace custom instructions

- Imperas MPD is Eclipse based source code debug tool
- Can debug using source line or instruction level
- See new custom instructions and any new additional state registers



# Document custom instructions

- Imperas tools automatically generate a processor model document PDF
- Includes all base model registers and any new registers
- Provides detailed documentation of new custom instructions

## Chapter 2

## Instruction Extensions

RISCV processors may add various custom extensions to the basic RISC-V architecture. This processor has been extended, using an extension library, to add several instruction using the Custom0 opcode.

### 2.1 Custom Instructions

This model includes four Chacha20 acceleration instructions (one for each rotate distance) are added to encode the XOR and ROTATE parts of the quarter rounds.

#### 2.1.1 chacha20qr1

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| 0000000 | Rs2 | Rs1 | 000 (QR1) | Rd | Custom0 0001011 |

dst = (src1 ⊕ src2) <<<16

#### 2.1.2 chacha20qr2

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| 0000000 | Rs2 | Rs1 | 001 (QR2) | Rd | Custom0 0001011 |

dst = (src1 ⊕ src2) <<<12

#### 2.1.3 chacha20qr3

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| 0000000 | Rs2 | Rs1 | 010 (QR3) | Rd | Custom0 0001011 |

dst = (src1 ⊕ src2) <<<8

# Further tools for model developers

- Model source line coverage
  - To see how completely the tests exercise the model

type filter text

| Name | Total Lines | Instrumente | Executed Lir | Coverage % |
|---|---|---|---|---|
| Summary | 16,900 | 4,411 | 2,834 | 64.25% |
| customChaCha20.c | 374 | 87 | 42 | 48.28% |
| riscvBus.c | 175 | 46 | 1 | 2.17% |
| riscvCSR.c | 2,573 | 758 | 549 | 72.43% |
| riscvConfigList.c | 70 | 2 | 0 | 0.0% |
| riscvDebug.c | 527 | 167 | 72 | 43.11% |
| riscvDecode.c | 1,599 | 360 | 164 | 45.56% |
| riscvDisassemble.c | 514 | 185 | 185 | 100.0% |
| riscvDoc.c | 725 | 143 | 30 | 20.98% |
| riscvExceptions.c | 1,408 | 420 | 317 | 75.48% |
| riscvInfo.c | 83 | 3 | 0 | 0.0% |
| riscvMain.c | 383 | 147 | 25 | 17.01% |
| riscvMorph.c | 2,887 | 1,032 | 776 | 75.19% |
| riscvParameters.c | 589 | 145 | 9 | 6.21% |
| riscvRegisterTypes.h | 115 | 10 | 6 | 60.0% |
| riscvSemiHost.c | 44 | 6 | 3 | 50.0% |
| riscvStructure.h | 204 | 4 | 2 | 50.0% |
| riscvUtils.c | 493 | 122 | 45 | 36.89% |
| riscvVM.c | 2,695 | 770 | 608 | 78.96% |
| vmiMt.h | 1,442 | 4 | 0 | 0.0% |

test_c.c | test_custom.c | customChaCha20.c | riscv32.c | _start() at 0x100d0

```
//
// Emit code implementing exchange instruction
//
static void emitChaCha20(
    vmiProcessorP processor,
    vmiosObjectP  object,
    Uns32         instruction,
    Uns32         rotl
) {

    // 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(
    vmiReg reg_tmp = vmimtGetExtTemp

    vmimtGetR(processor, RISCV_REG_BITS, reg_rs1, object->riscvRegs[rs1]);
    vmimtGetR(processor, RISCV_REG_BITS, reg_rs2, object->riscvRegs[rs2]);
    vmimtBinopRRR(32, vmi_XOR, reg_tmp, reg_rs1, reg_rs2, 0);
    vmimtBinopRC(32, vmi_ROL, reg_tmp, rotl, 0);

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

line executed 16 times

Press 'F2' for focus

# Checking RISC-V compliance

- Imperas ISS can be used to check RISC-V specification compliance
- Imperas ISS is used in the RISCV.org Compliance Working Group's compliance tests is a version of the Imperas ISS
- With your new custom instructions modelled you can run the RISC-V Foundation's compliance suite and ensure that your processor is still RISC-V compliant
  - The official RISC-V Compliance suite is available at
    - Compliance Task Group GitHub repository https://github.com/riscv/riscv-compliance

# Summary

- If you are adding new instructions or state to a processor – then the tools and design flow will need to cover:
  - Modelling, simulation, timing, tracing, debug, coverage, and profiling of new instructions
  - Documentation, and checklist methodology / solution
- Leverage the technical working groups of the RISC-V Foundation
  - Compliance Working Group GitHub repository is a useful starting point
  - Ensure design Compliance and follow similar methodology on the new instructions
- Software teams can start porting software to the new model/instructions
- Virtual Platforms also provide a basis for early software development at customers and partners as pre-sales evaluation and development

# Contacts and Links

- https://riscv.org
- Visit www.imperas.com and www.OVPworld.org for more information
- RISC-V Foundation Compliance Suite & riscvOVPsim download
  https://github.com/riscv/riscv-compliance
- Duncan Graham, Sr. Applications Engineer - graham@imperas.com
- Kevin McDermott, VP Marketing - kevinm@imperas.com

# Questions

Finalize slide set with questions slide

# RISC-V Configuration and Customization

Zdeněk Přikryl, Chris Jones



# Who Is Codasip

- Leading provider of RISC-V processor IP
  - Introduced its first RISC-V processor in November 2015
  - Offers its own portfolio of RISC-V processors (Codasip Bk)
  - Provides unique design automation tools for easy modification of RISC-V processors
- Founding member of RISC-V Foundation (www.riscv.org)
  - Member of several working groups within the Foundation
- Active contributor to LLVM and other open-source projects

# Configuration vs Customization

- RISC-V offers a wide range of ISA extensions:
  - **I/E** for integer instructions
  - **M** for multiplication and division
  - **C** for compact instruction
  - WIP: **B**, **P**, **V**, …
  - and others
- Configuration: Selecting multiple ISA extensions
  - Enabled by some vendors or open-source projects
  - *Still insufficient* for some application domains

# Customer Use Case

## Performance improvement through high-level optimization:

FIR implementation in C with 200 16-bit input samples and 16 16-bit coefficients

Design Iterations

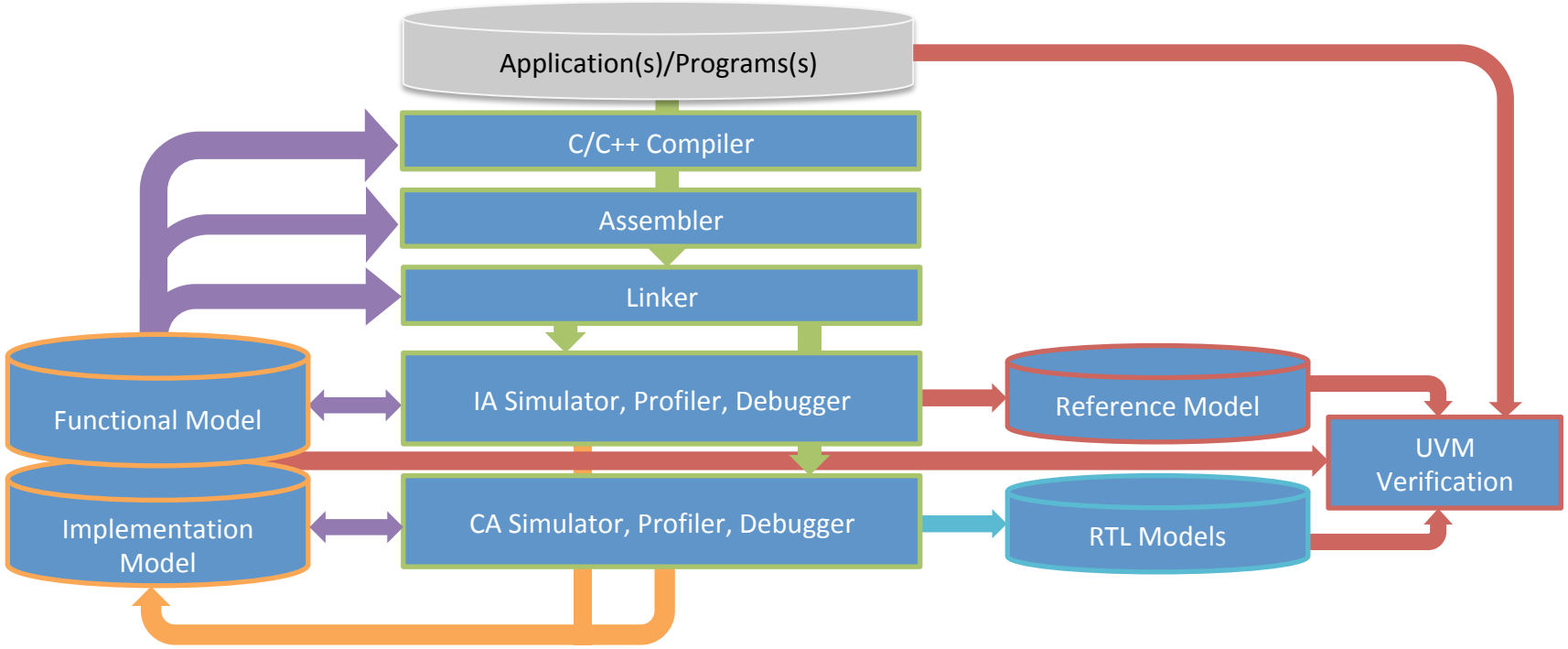
| **Configuration** | Clock Cycles[1] | Code size[2] | **Speedup** Against Base | Area (Gates)[3] | **Area Expansion** Against Base |
|---|---|---|---|---|---|
| Base | 1,764,256 | 232 | | 16.0k | |
| Base + Serial Multiplier | 427,561 | 148 | 4.12 x | 19.7k | 1.24 x |
| Base + Parallel Multiplier | 133,061 | 148 | 13.26 x | 26.2k | 1.64 x |
| Base + DSP Extensions | 31,371 | 64 | 56.24 x | 38.7k | 2.43 x |

[1] Fewer clock cycles → same software takes less time to run.
[2] Smaller code size (optimized software) → less memory saves money.
[3] More gates in advanced cores → higher cost. Here, only **2x** area increase provides **50x** performance gain.

# *Standard* Approach to Customization

- Manually adding new instructions to the RISC-V ISA:
  - Model and simulate the instruction
  - Modify the compiler
  - Add support in the debugger
  - Write Verilog to implement in hardware
  - Verify, verify, verify, …
- Challenging and time-consuming
- **Automation** desirable for each step above

# *Automatization* Approach to Customization

# *Codasip* Approach to Customization

Codasip Studio:

- Introduced in 2014
- Silicon-proven by major vendors
- Allows for fast & easy customization of base instruction set:
  - Single cycle MAC
  - Floating point
  - Custom crypto functions
  - …

## Codasip Studio

**RTL Automation**
Powerful High-level Syntheses

**Verilog** **VHDL**

**SDK automation**
Standards-based tools & models

LLVM COMPILER INFRASTRUCTURE

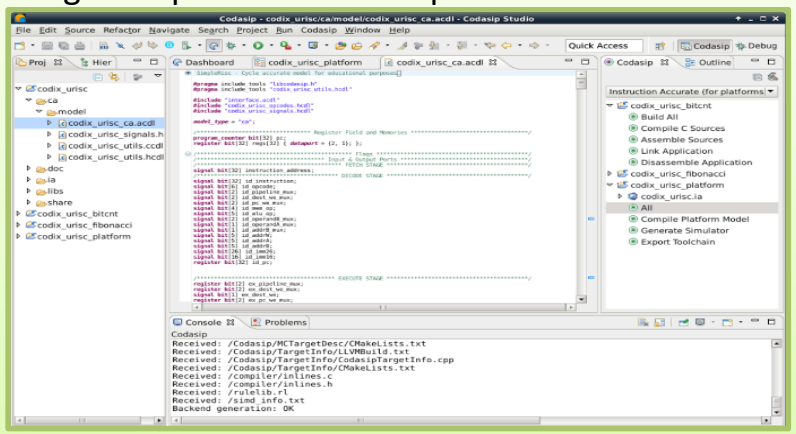
**Verification Automation**
VSP and processor validation

UVM SYSTEMC

**CodAL** – processor description language

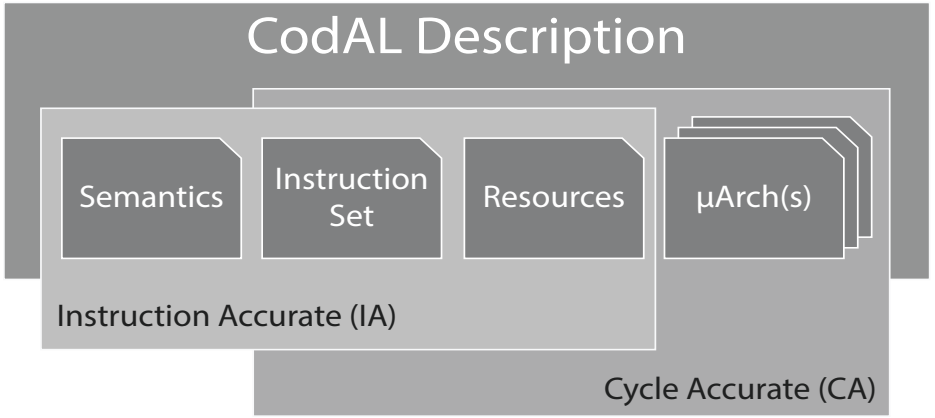
```
element i_mac {
   use reg as dst, src1, src2;
   assembler { "mac" dst "," src1 "," src2 };
   binary { OP_MAC:8 dst src1 src2 0:9 };
      semantics {
         rf[dst] += rf[src1] * rf[src2];
      };
};
```

Integrated processor development environment



# CodAL Models

- Processor IP at High Level of Abstraction
- Easy to understand C-like language
- Features:
  - Can model a rich set of processor capabilities
  - Can implement multiple microarchitectures in a single model
- Usage:
  - Used to model and verify all Codasip processors
  - Provided to Codasip IP customers as a starting point for their processor optimizations and modifications




```
/*  Multiply and accumulate: semantics
            dst += src1 * src2              */

element i_mac {
        use reg as dst, src1, src2;
        assembler { "mac" dst "," src1 "," src2 };
        binary { OP_MAC:bit[8] dst src1 src2 0:bit[9] };
        semantics {
                rf[dst] += rf[src1] * rf[src2];
        };
};
```

# **B** ISA Extension

- Bit manipulation instructions, ca 30:
  - Bit insert and extract
  - Byte swapping
  - Rotations
  - Bit swapping/shuffling
  - Zero/one counters
  - …
- Not yet ratified
  - Must be implemented as custom extensions

# Functional Model

- Written in CodAL
  - in 10 days by a single engineer
- 900 LOC
- Software development kit (SDK) automatically generated by Studio, including
  - C compiler
  - Instruction set simulator (ISS)
  - Profiler for checking the impact of the extensions

```
element i_gzip
{
    use opc_gzip as opc;
    use reg_any as dst, src1;
    use shift_imm  as imm ;
    assembler { opc dst "," src1 "," imm};
    binary { opc[OPC_FRAG_SHIFT] imm src1 opc[OPC_FRAG1] dst opc[OPC_FRAG0] };
    semantics
    {
       rf_gpr_write (dst, gzip_uXlen(rf_gpr_read(src1), imm));
    };
};
set isa_b += i_gzip;
```

```
uXlen gzip_uXlen (const uXlen rsc1 , const uXlen mode)
{
    uint32 zip_mode, x ;
    x =  rsc1;
    zip_mode = mode & 31;
    if(zip_mode & 1)
    {
        if(zip_mode & 2)
            x = gzip_stage (x, MASK_ZIP2_L, MASK_ZIP2_R, 1);
        if(zip_mode & 4)
            x = gzip_stage (x, MASK_ZIP4_L, MASK_ZIP4_R, 2);
        if(zip_mode & 8)
            x = gzip_stage (x, MASK_ZIP8_L, MASK_ZIP8_R, 4);
        if(zip_mode & 16)
            x = gzip_stage (x, MASK_ZIP16_L, MASK_ZIP16_R, 8);
    }
```

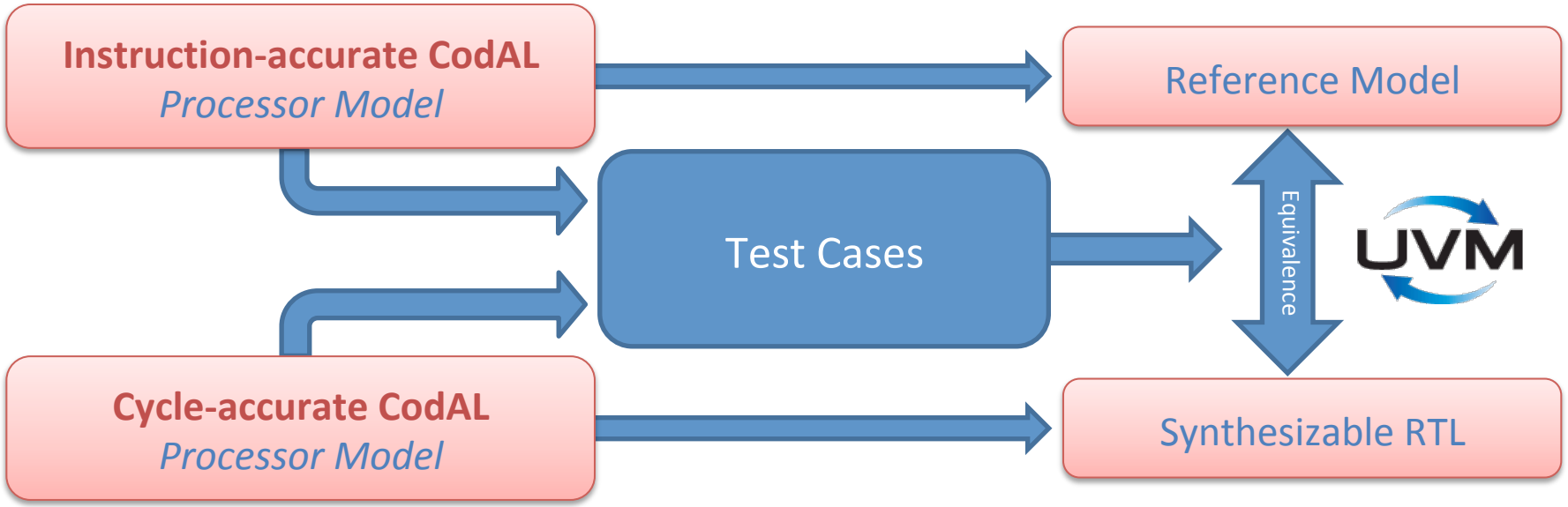
# Implementation Model

- Written in CodAL
  - in 3 weeks by a single engineer
- 1500 LOC
- Hardware design kit (HDK) automatically generated by Studio, including
  - RTL
  - Testbench
  - UVM-based verification environment

```
#ifdef OPTION_EXTENSION_B
            case SLO:
                ex_result = ones_shifter_32(SLO, ex_aluop1, ex_aluop2);
                break;
            case SRO:
                ex_result = ones_shifter_32(SRO, ex_aluop1, ex_aluop2);
                break;
            case ANDC:
                ex_result = (uXlen)ex_aluop1 & (~ ex_aluop2);
                break;
            case ROTR :
                ex_result = ex_aluop1 >>> ex_aluop2;
                break;
            case ROTL :
                ex_result = ex_aluop1 <<< ex_aluop2;
                break;
            case CTZ :
                ex_result = codasip_ctlz_uint32(ex_aluop1);
                break;
            case CLZ :
                ex_result = codasip_cttz_uint32(ex_aluop1);
                break;
#endif
```

# Verification

- Consistency checker
- Random assembler program generator
- UVM Verification Environment
  - For checking that RTL corresponds to specification (in this case, IA model definition)
  - Environment in SystemVerilog generated automatically from Codasip Studio



# Conclusion

- Configurability
  - Useful, but often not sufficient
- Customization
  - When the best PPA for your application domain is required
- Standard (manual) approach for custom ISA extensions
  - Error-prone and time-consuming

Codasip offers an easy, automatized way to add your secret sauce:

- Custom ISA extensions
- Microarchitectural improvements

*Example: **B** ISA extension, supported by SDK and RTL, done in a couple of weeks.*

# Questions

# Post Silicon Debug and Analytics for RISC-V Based SoCs

**Peter Shields**

*UltraSoC Technologies Ltd.*

DVCon Europe October, 2018

# Corporate Overview

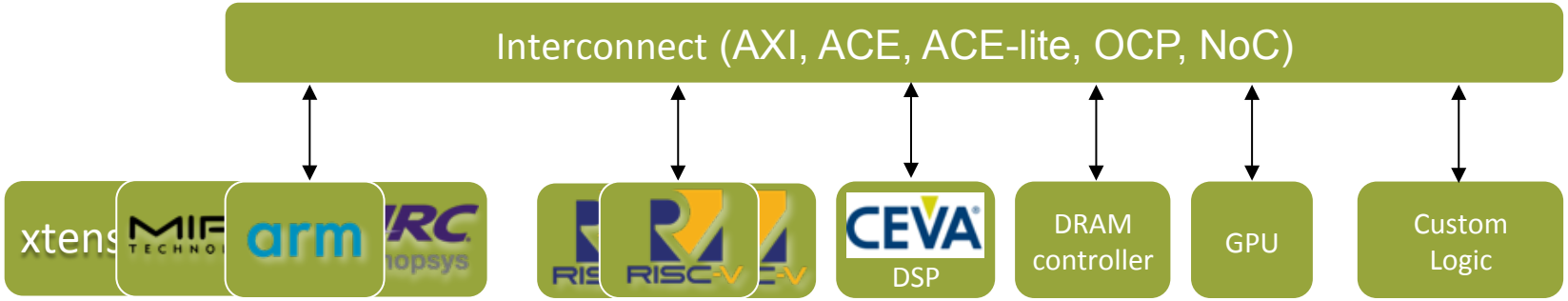
- We are a provider of SoC analytics solutions consisting of on-chip RTL IP and software
- VC-funded and based in Cambridge UK
- £4.7M VC round in 2017 with addition of Alberto Sangiovanni-Vincentelli
- 25 patents granted + 16 pending
- Seasoned management team
- Key partners & ecosystem
- Silicon proven technology in multiple customer designs
- Revenue, blue-chip customers, repeat business

# On-chip Analytics for SoC

**A coherent architecture to debug, develop, optimize & secure**

- Full SoC visibility, HW & SW
- Support all architectures: Freedom of IP selection
- Real-time & non-intrusive
- Advanced analytics & forensics
- Power/Performance optimization
- “in life” analytics & SLA compliance
- Supports Functional Safety
- Supports Bare Metal Security™
- High-speed debug over USB or SerDes

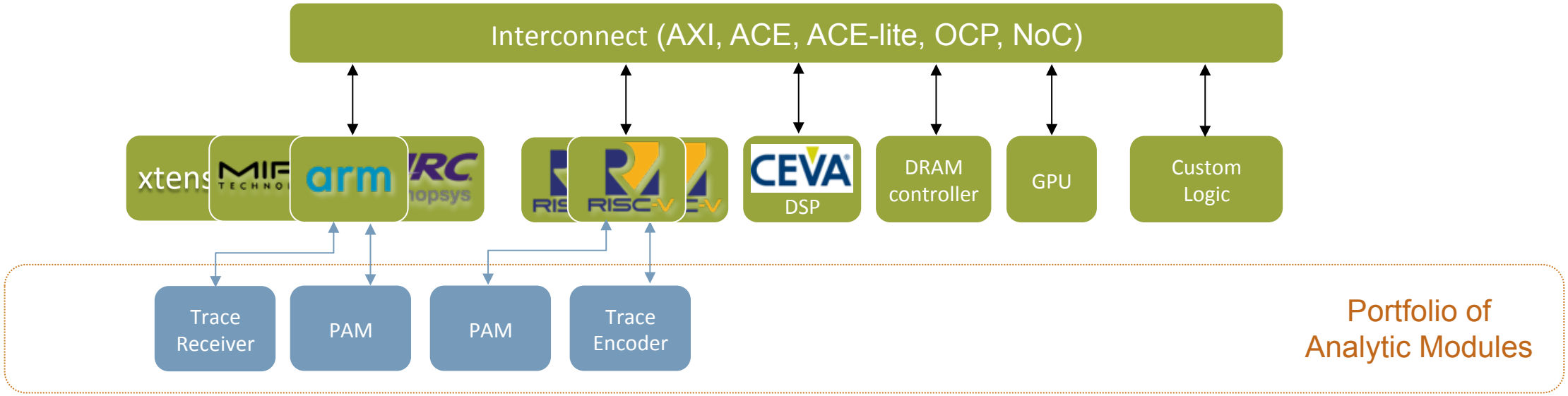
# A High Level View of SoC




System Block

UltraSoC IP

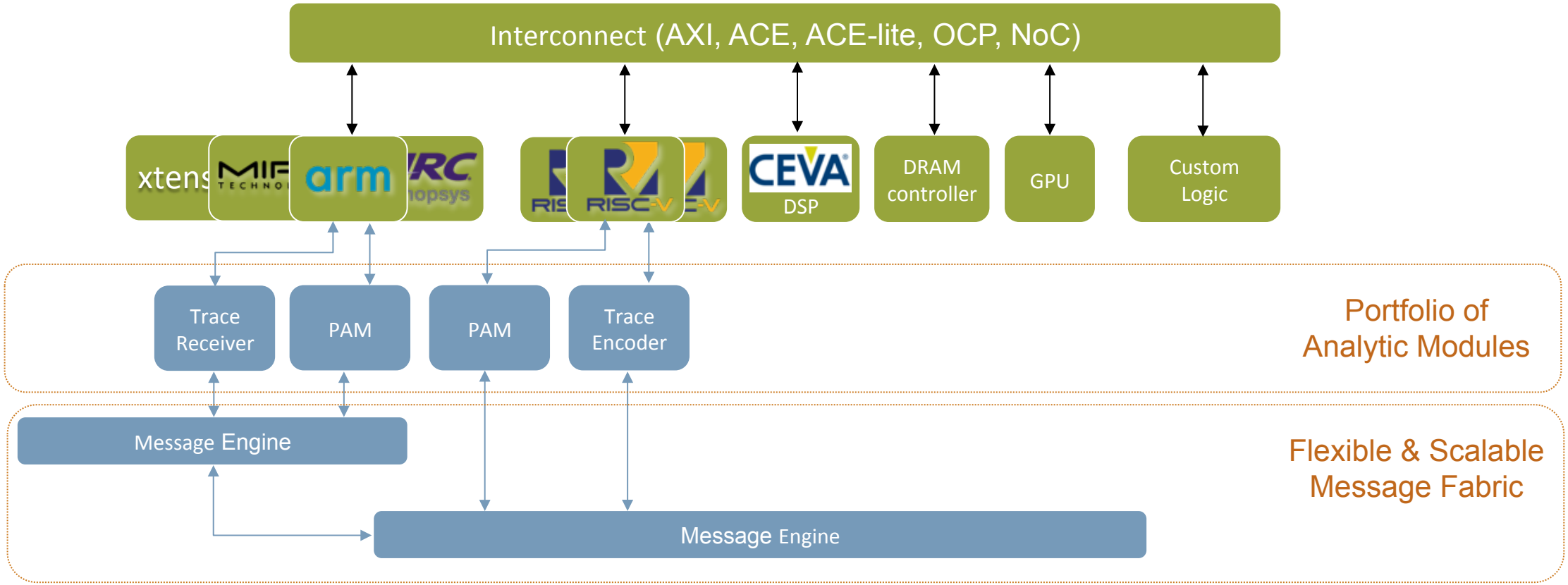
# Processor Control and Trace




System Block

UltraSoC IP

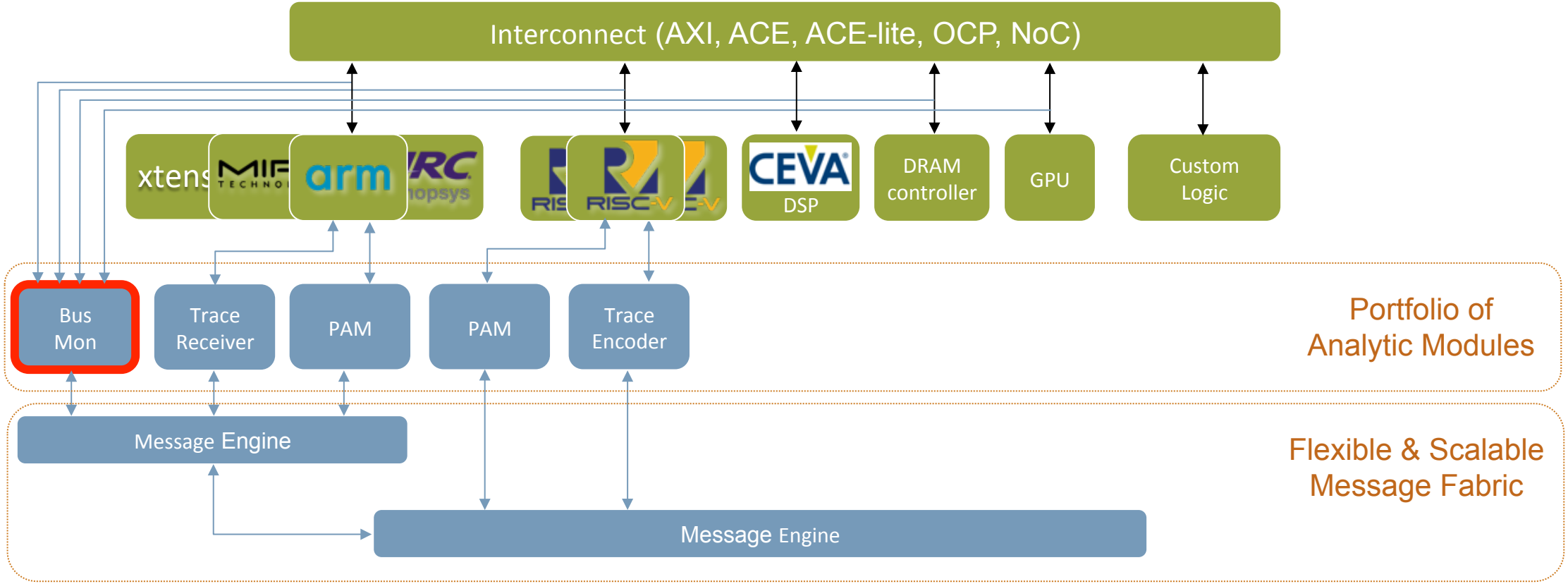
# Messaging Subsystem




System Block

UltraSoC IP

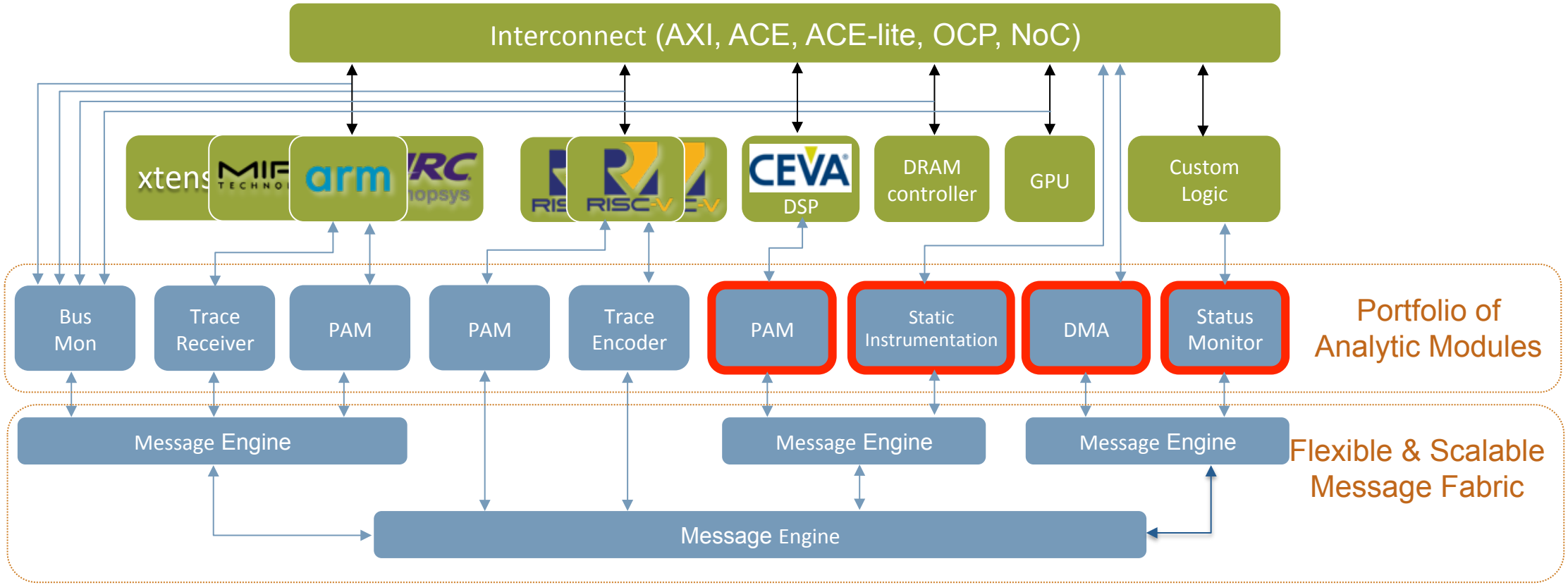
# Transaction Aware Bus Monitoring




System Block

UltraSoC IP

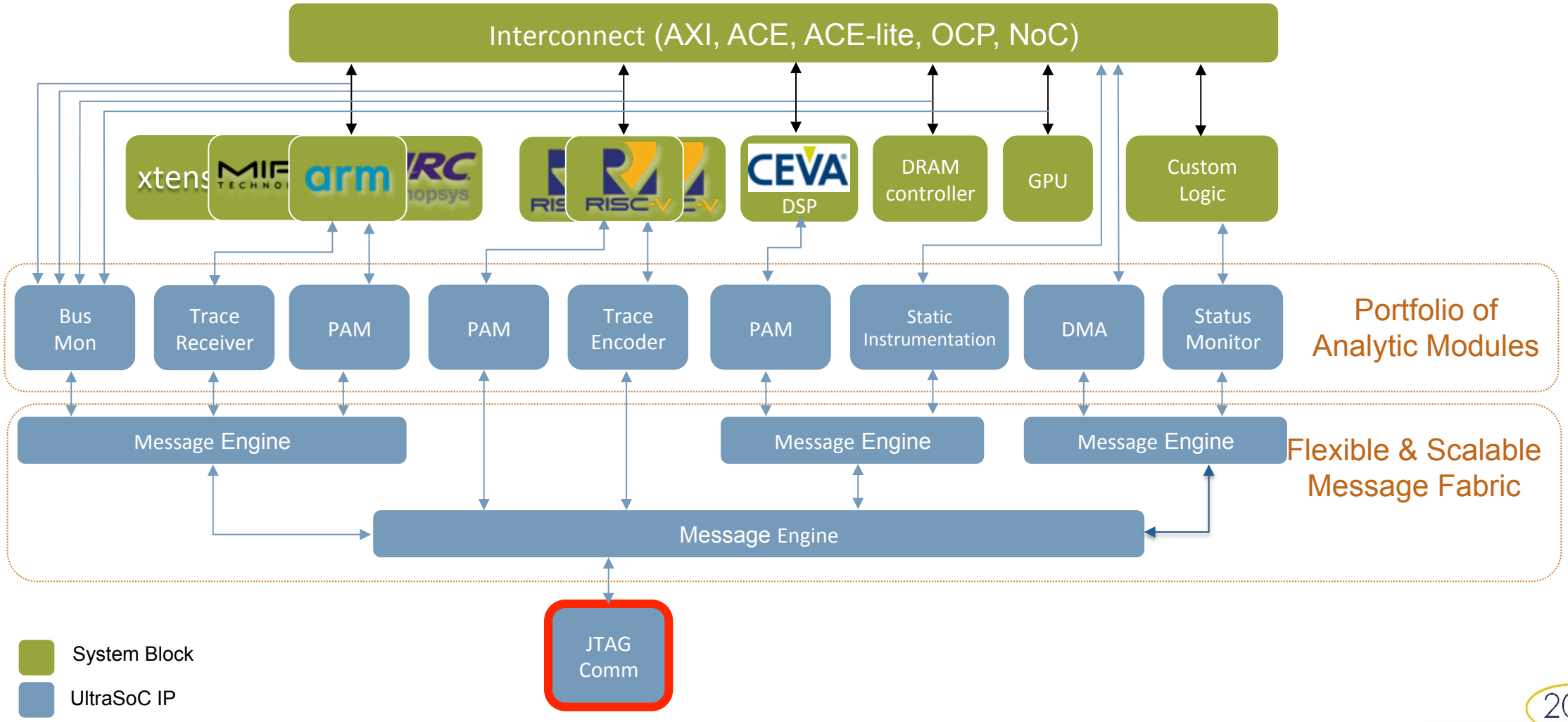
# Additional Monitors



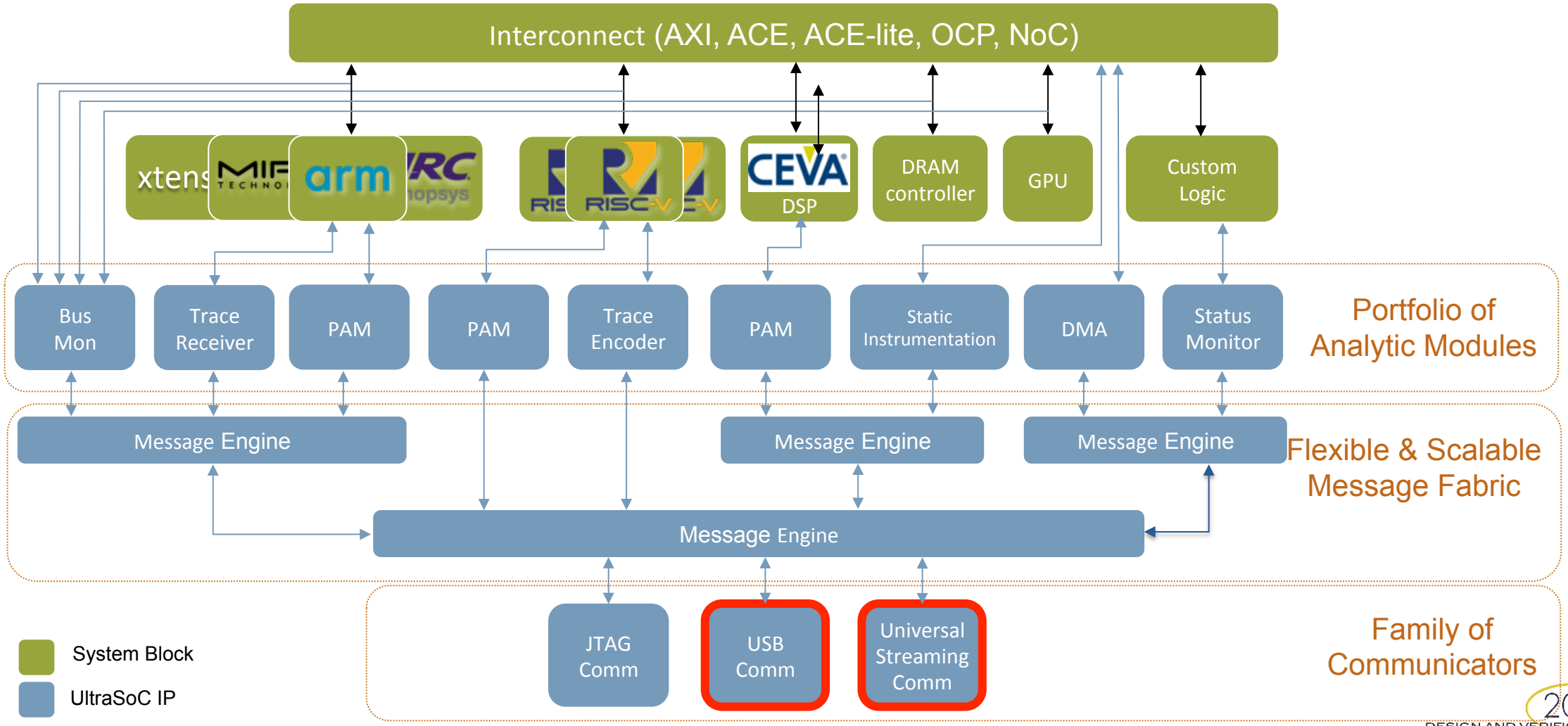

System Block

UltraSoC IP

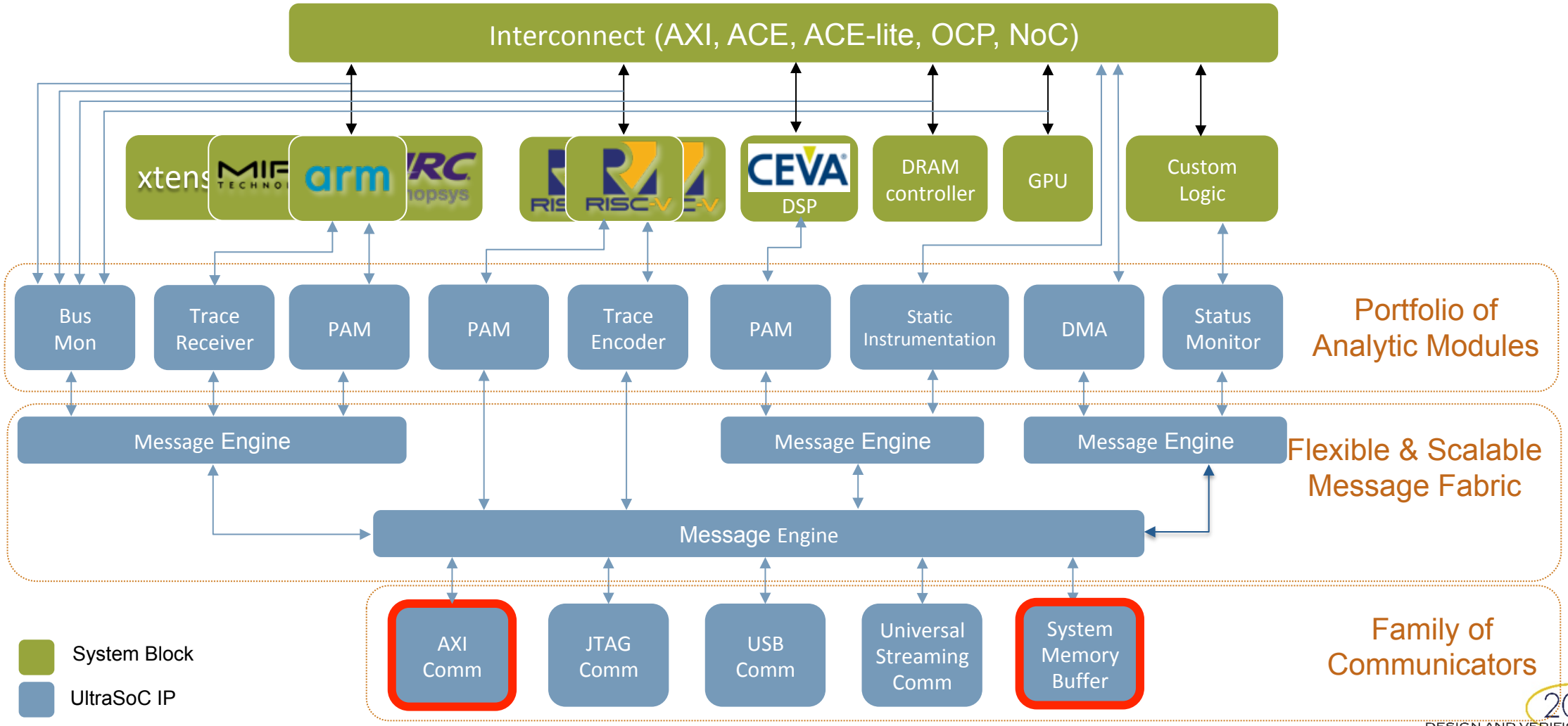
# Control and Data Off Chip



# High Speed Communicators

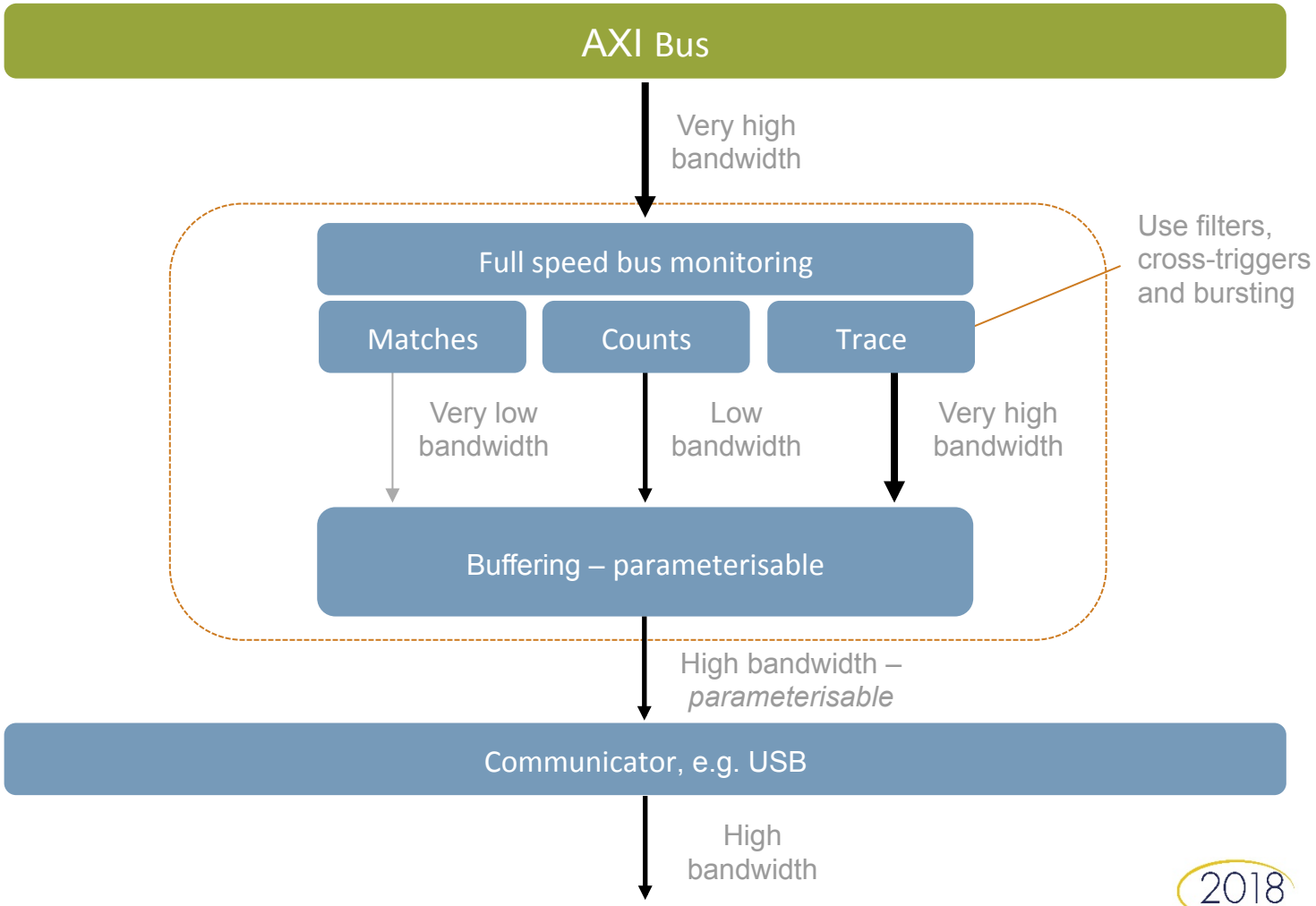


# On-Chip Data Storage and Control



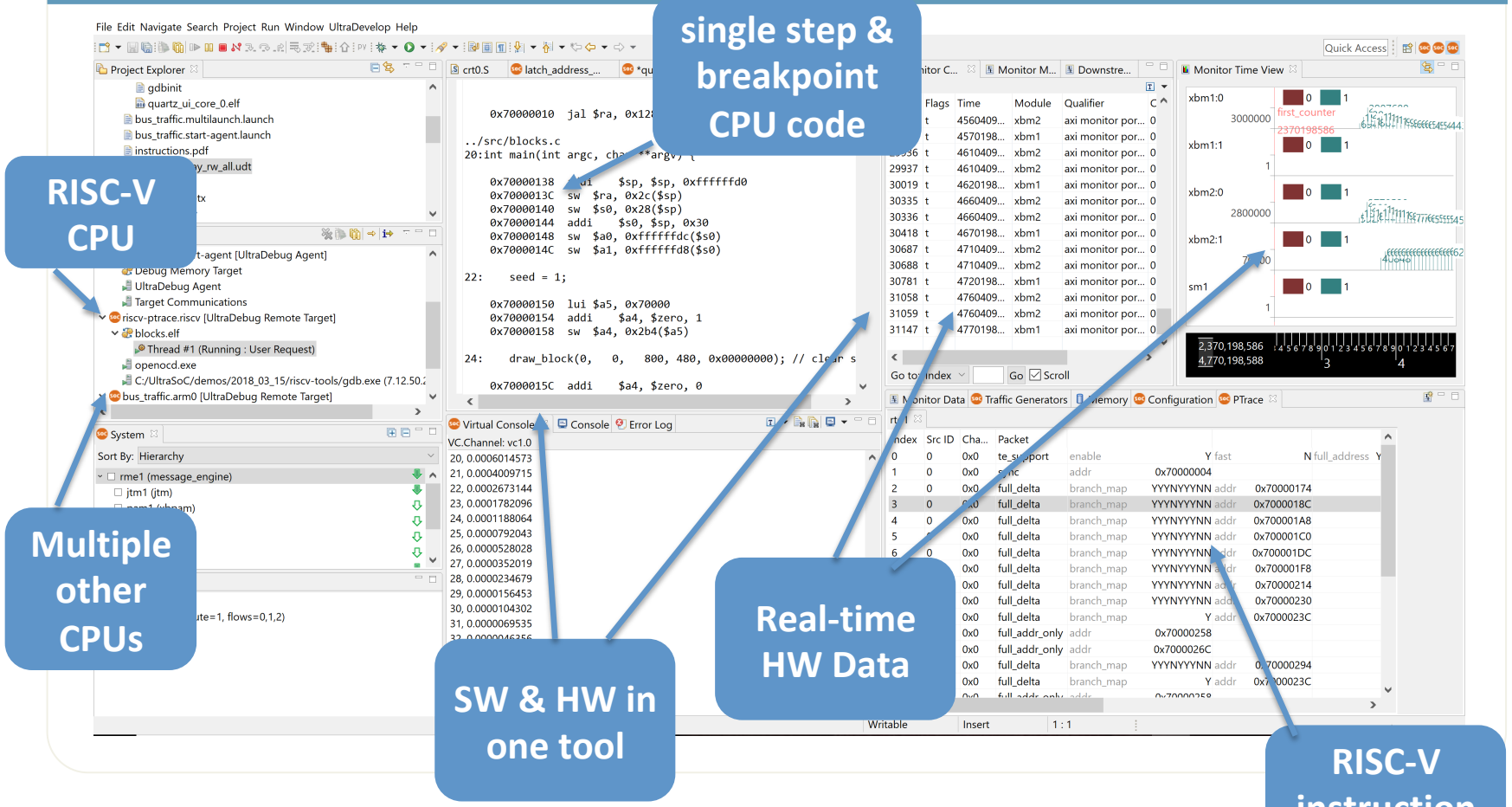
# Intelligent Analytic Modules

- Take a Bus Monitor as an example
- Configurable number of
  - filters
  - counters
  - trace buffer size
- Run-time programmability
  - Filter matching for triggering
  - Filter matching for counting
  - Filtering for bus trace
- Gather statistics (best, worst, average)
- Only meaningful information sent
- Reduces bandwidth & data volume
- Focus on what is relevant



# Software tools for data-driven insights

## Eclipse based UltraDevelop IDE



# RISC-V Run Control

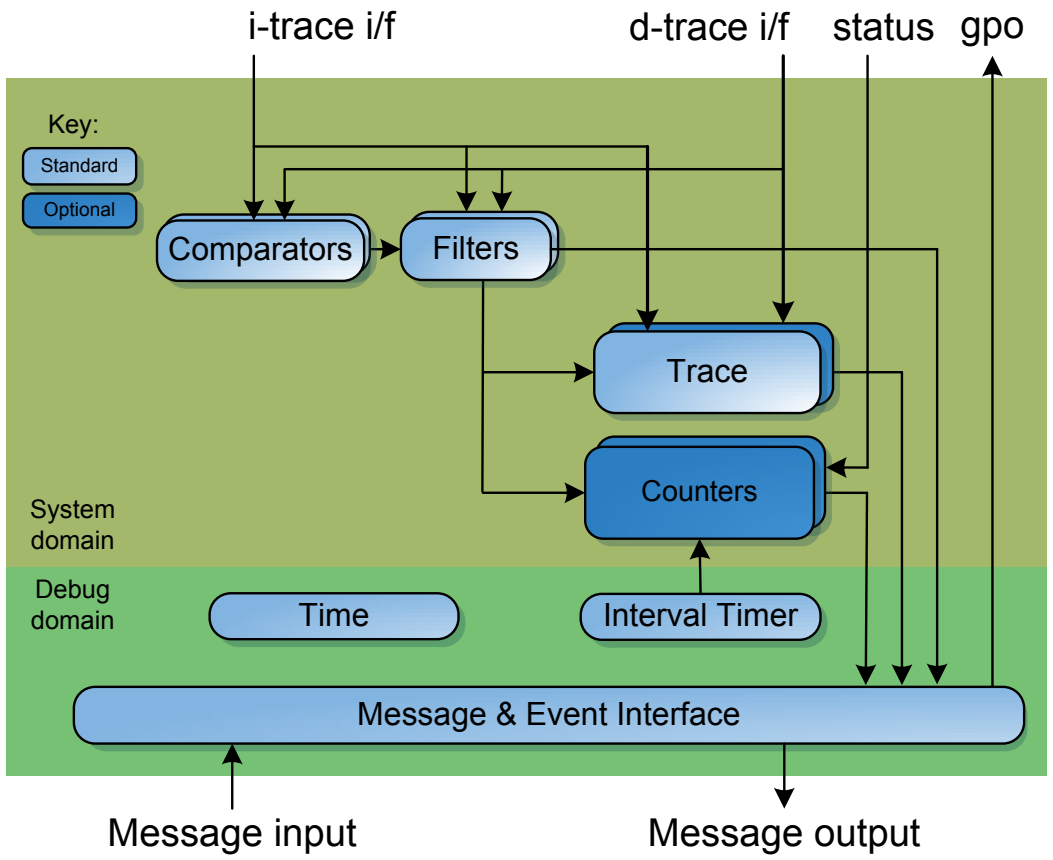
- Run control includes halt, resume, read/write of RAM and registers, and setting and clearing of breakpoints.
  - Extensions include watchpoints, running arbitrary code, semi-hosting and reverse debugging.
- Proposed debug standard for RISC-V
  - *Debug Module*
  - *Debug Transport Mechanism*
- Transport could be JTAG, Bus-mapped or other (USB etc)
- *https://github.com/riscv/riscv-debug-spec*

# RISC-V Trace Encoding

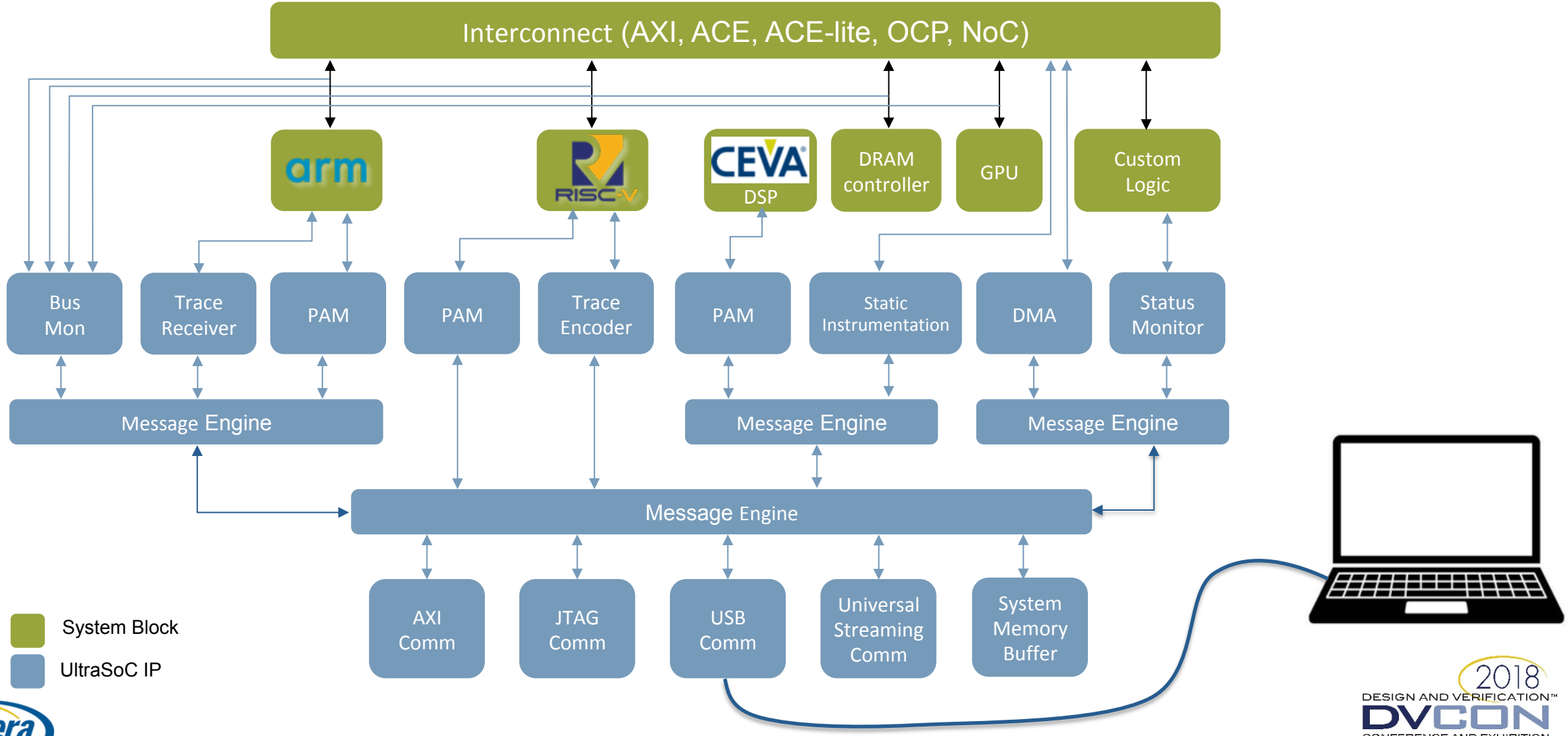
- Processor Trace Task Group
  - *standardize both a hardware interface to the RISC-V core and a packet/data format*
  - *commercial and open source trace encoders*
  - *provide enough information for instruction trace*
  - *in-order and out-of-order cores with extensions*
  - *standardize the data format for compressed branch trace so that program flow can be reconstructed by debugging tools*
- Proposed standard for RISC-V consists of
  - Trace Interface
  - Trace Encoding algorithm
- Efficiency and impact on trace bandwidth
- *https://github.com/riscv/riscv-trace-spec*

# RISC-V Trace Encoder

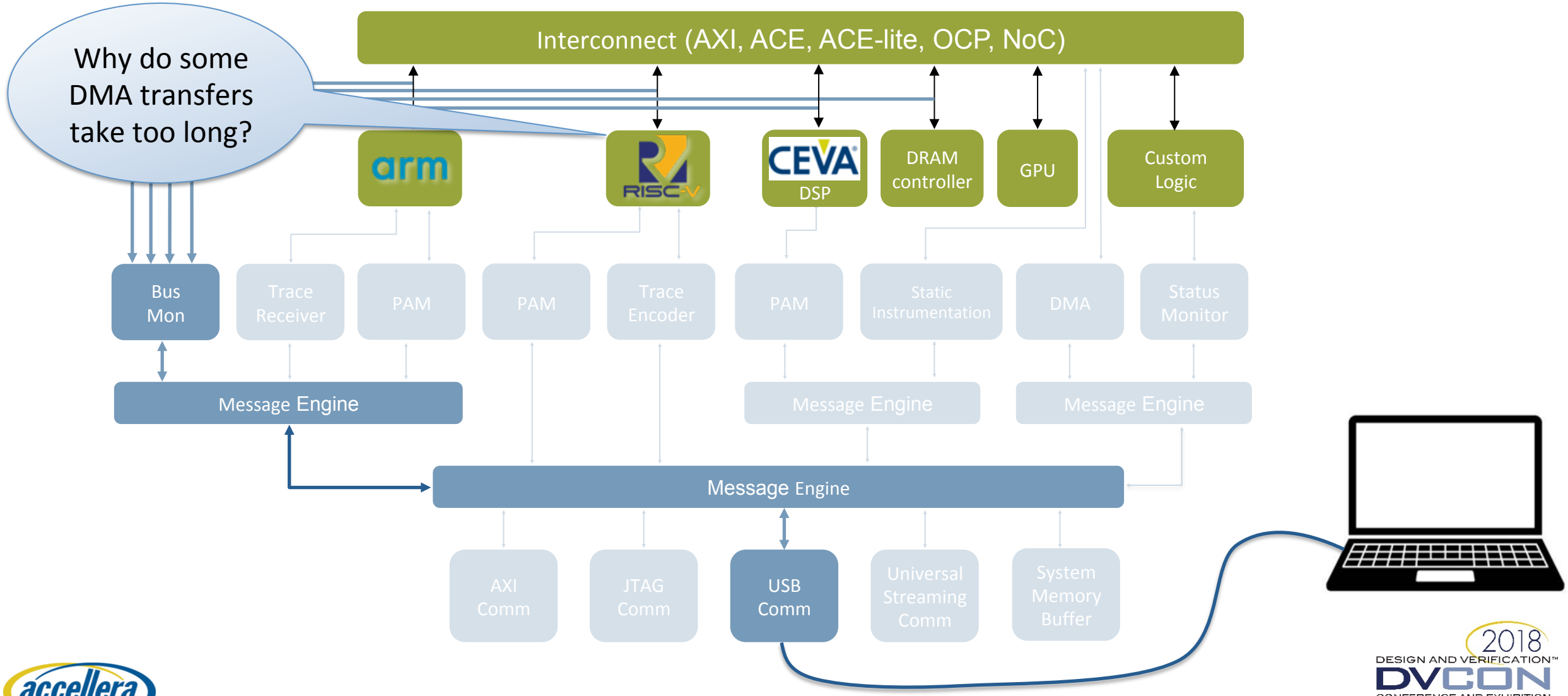
- Comparators and filters
  - What and when to trace
- Trace Encoding algorithm
- Trace Buffer
- Counters
  - Statistics
  - Performance



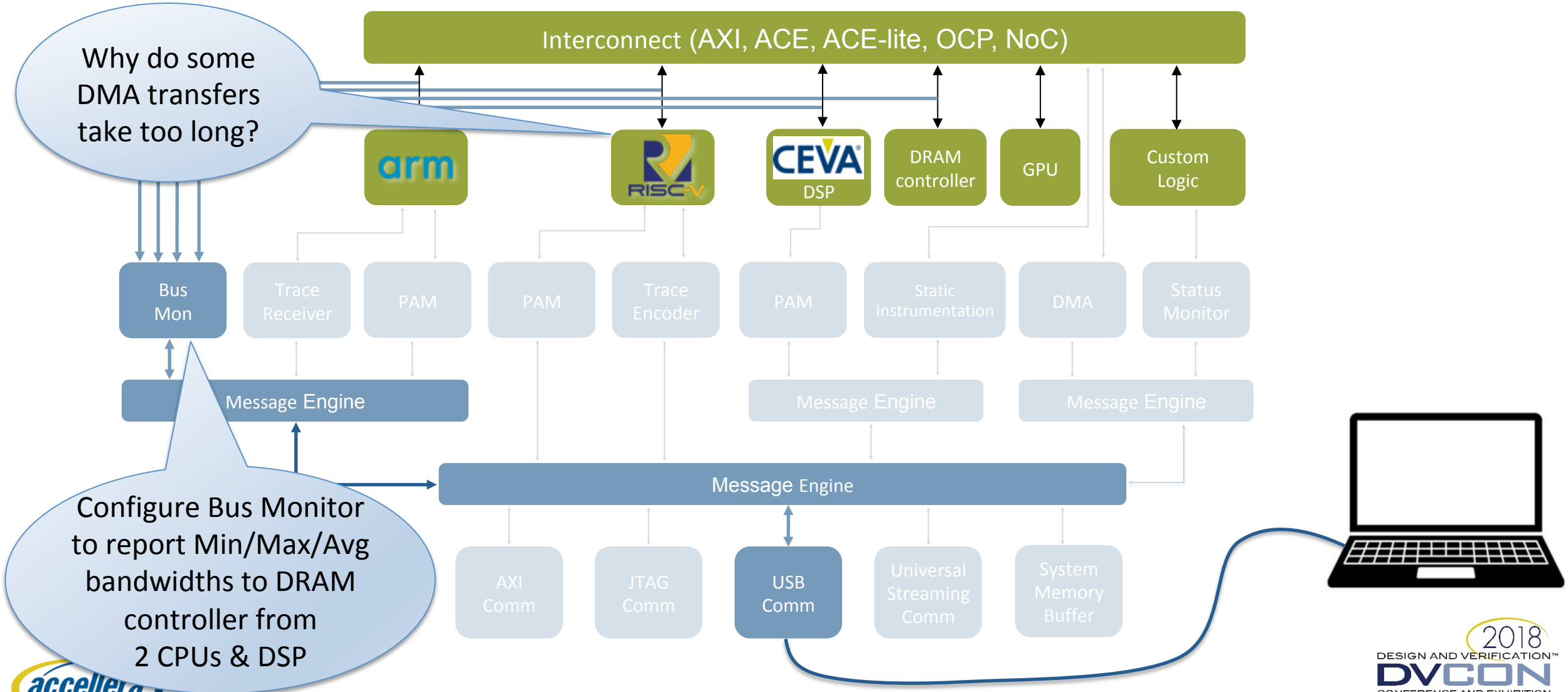
# Example Design



# DDR Bandwidth Example



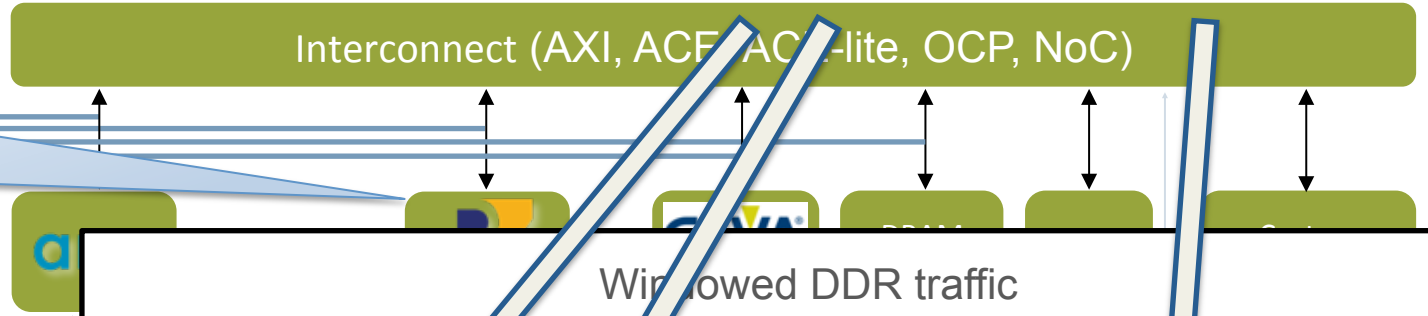
# DDR Bandwidth Example

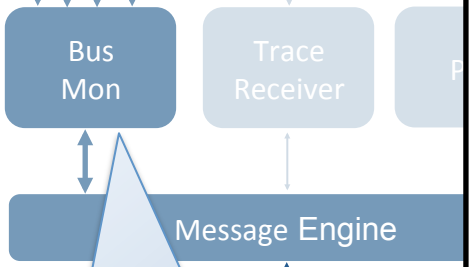


# DDR Bandwidth

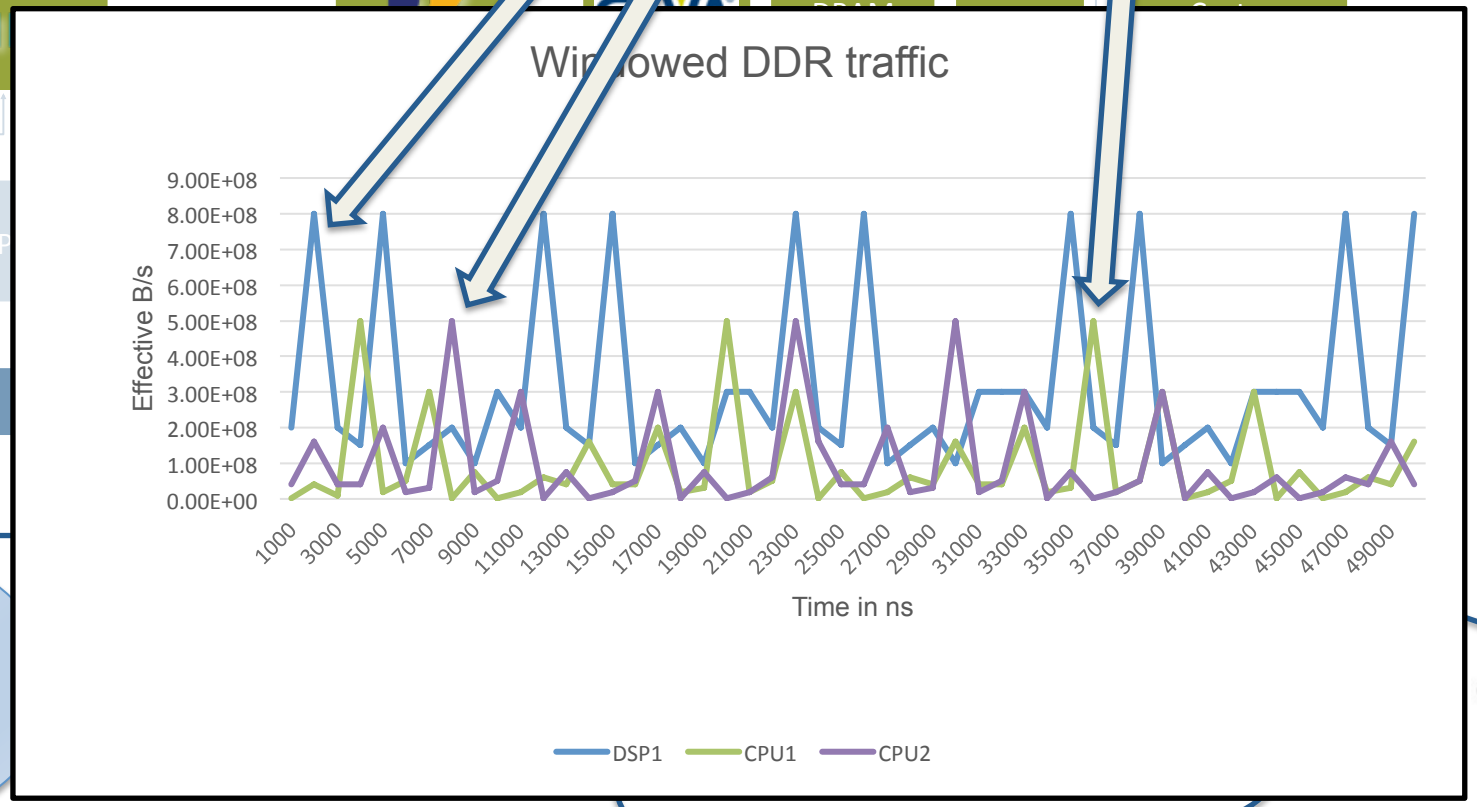
Aggregate bandwidth within spec (570Mbps avg)

Why do some DMA transfers take too long?




Configure Bus Monitor to report Min/Max/Avg bandwidths to DRAM controller from 2 CPUs & DSP




# DDR Bandwidth

Why do some DMA transfers take too long?

Interconnect (AXI, ACE, ACE-lite, OCP, NoC)

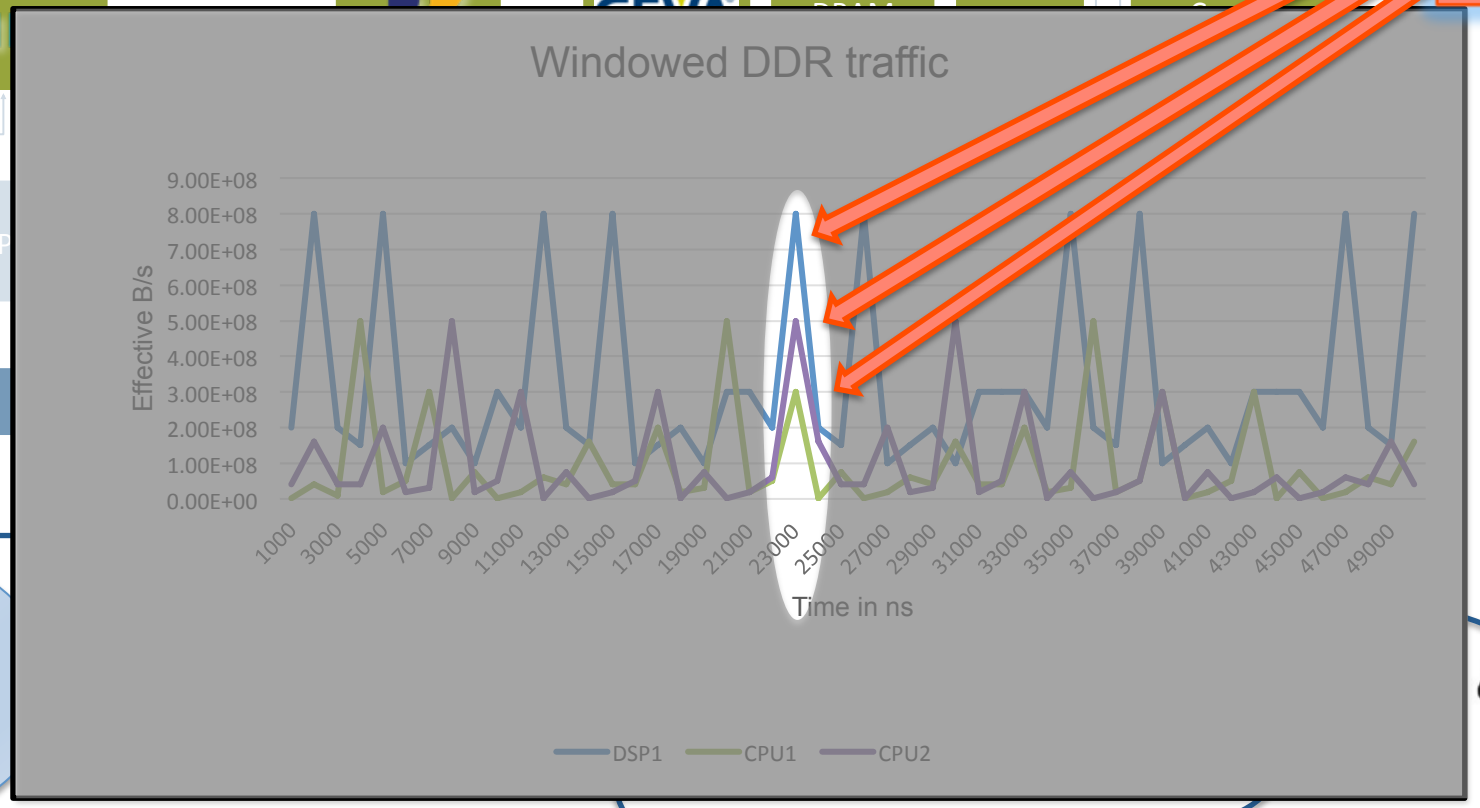
Combined peak bandwidth >2Gbps

Bus Mon

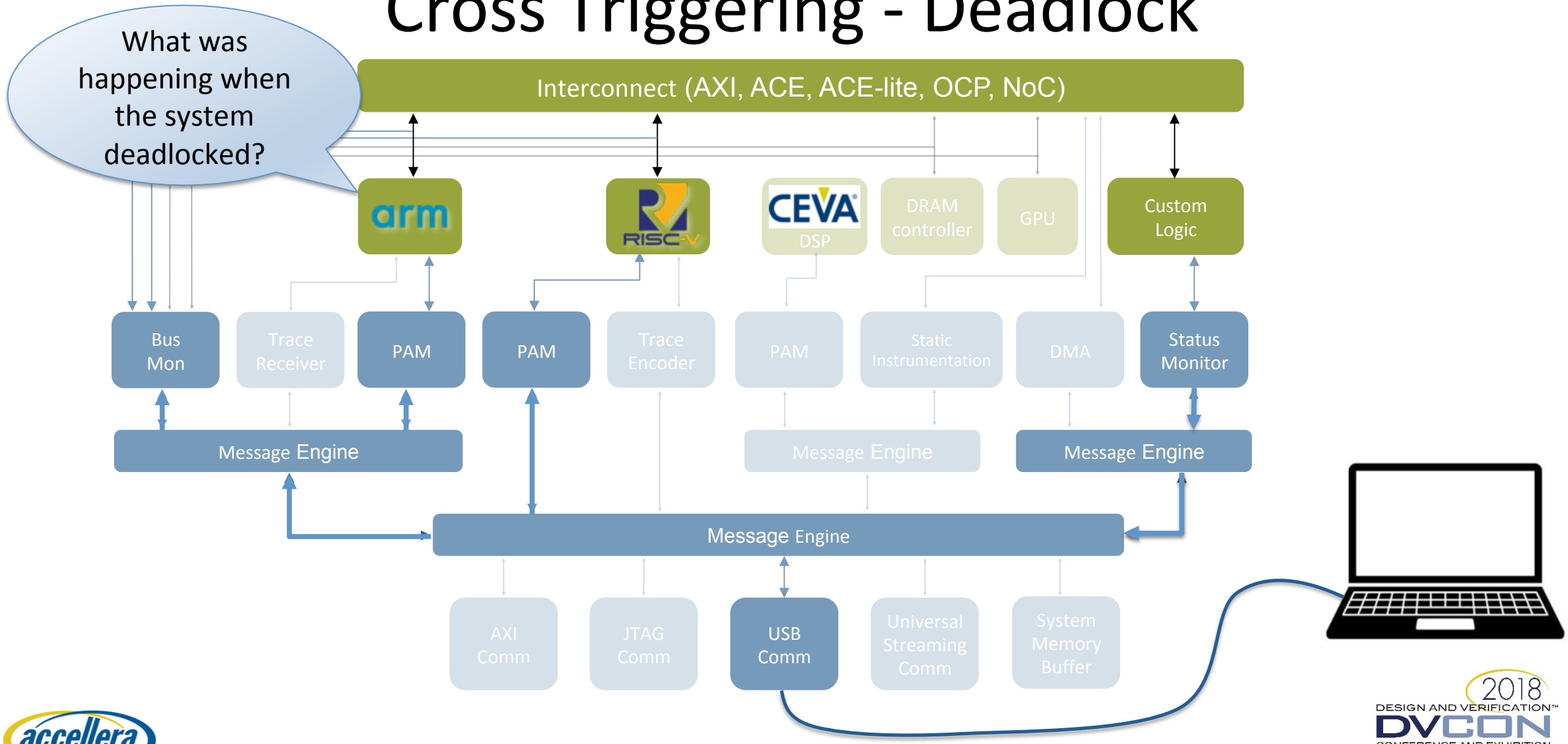
Trace Receiver

Message Engine

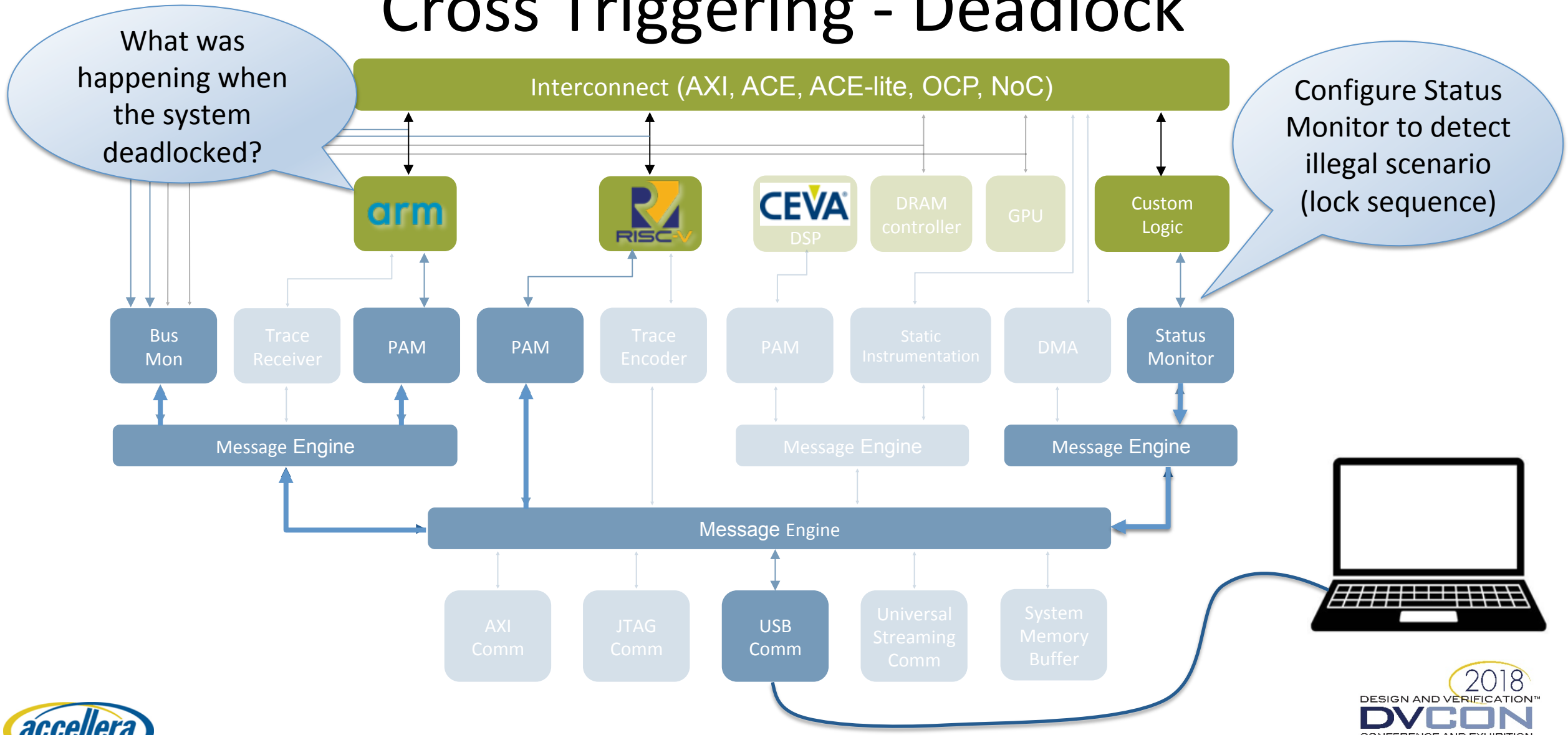
Configure Bus Monitor to report Min/Max/Avg bandwidths to DRAM controller from 2 CPUs & DSP



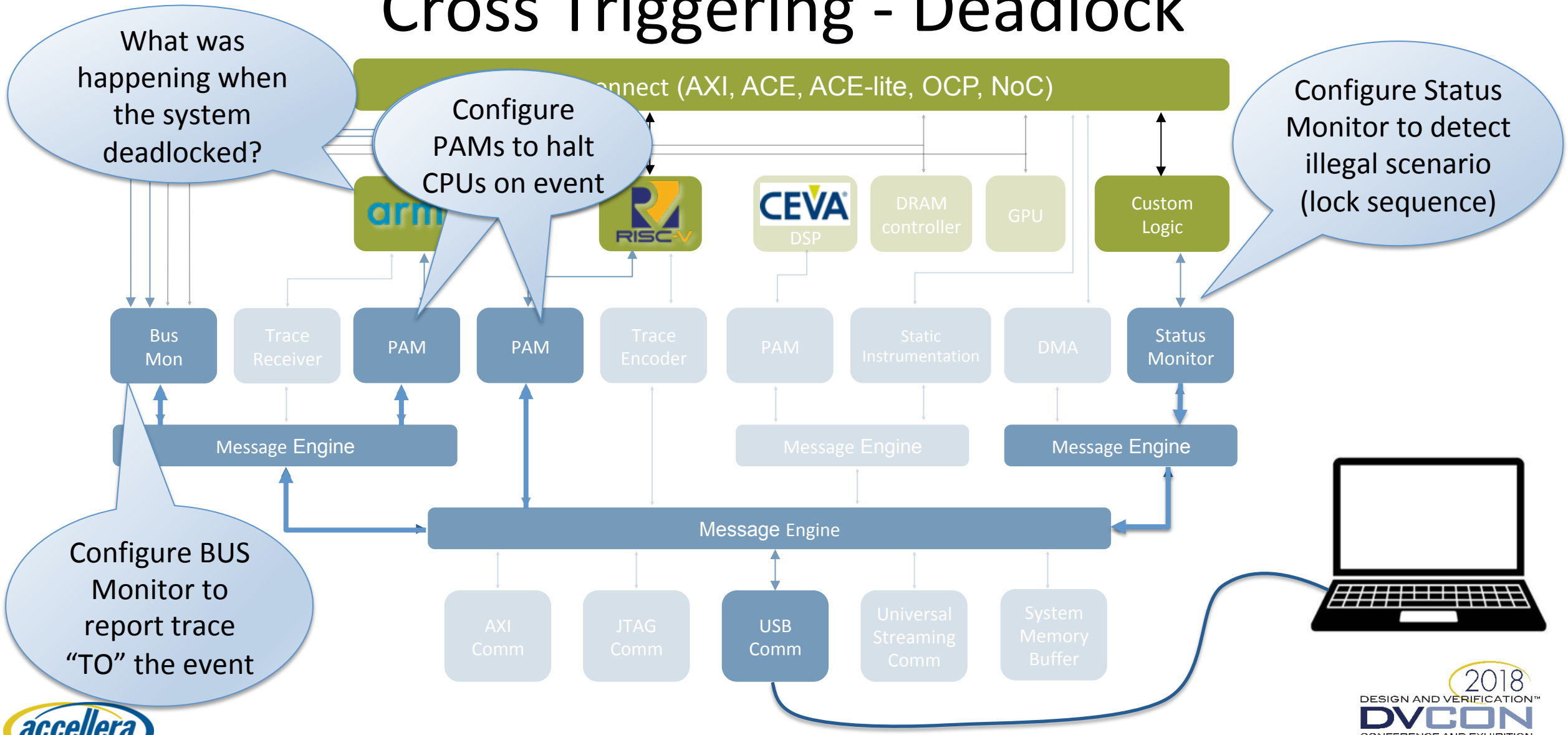

# Cross Triggering - Deadlock



# Cross Triggering - Deadlock



# Cross Triggering - Deadlock



# Cross triggering - Deadlock

What was happening when the system deadlocked?

Configure PAMs to halt CPUs on event

(AXI, ACE, ACE-lite, OCP, NoC)

Configure Status Monitor to detect illegal scenario (lock sequence)

Bus Mon

Trace Receiver

PAM

Message Engine

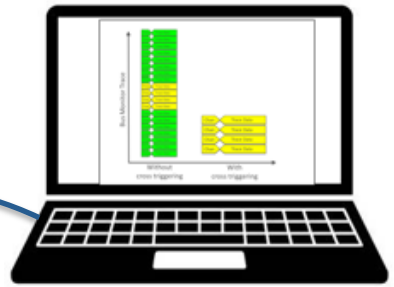
Configure BUS Monitor to report trace "TO" the event

Bus Monitor Trace

Chan Trace Data

Without cross triggering

With cross triggering



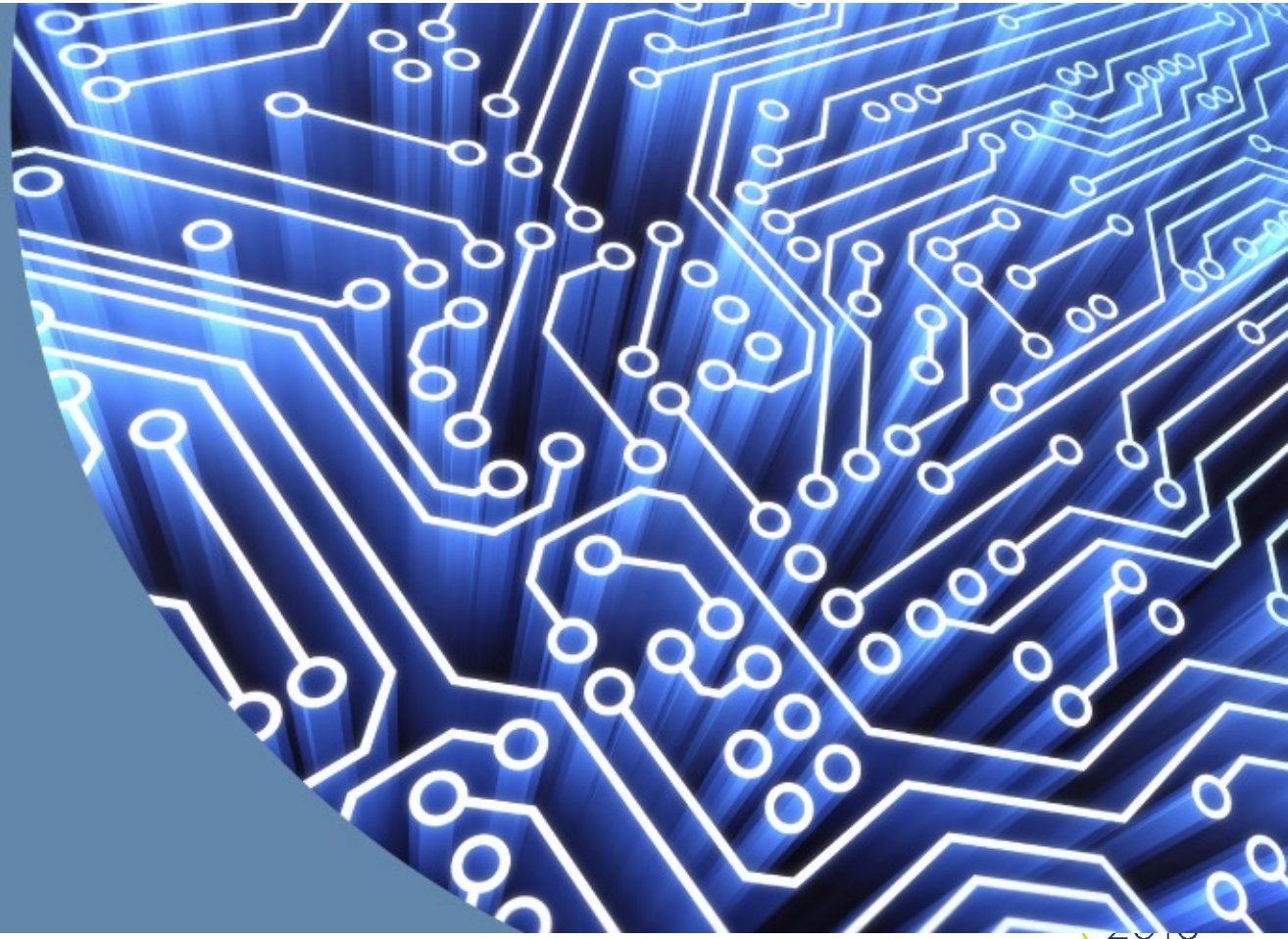


accellera SYSTEMS INITIATIVE

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# RISC-V Post Silicon Analytics Summary

- Heterogeneous architectures
- Non-intrusive, wire-speed
- High-speed analytics over USB/SerDes
- Debug, forensics, optimization
  - pre-silicon & post-silicon
- In-life system monitoring
- Cooperation within RISC-V community is critical



## Contact details:

Peter Shields
peter.shields@ultrasoc.com

www.ultrasoc.com
@UltraSoC

# Questions