OpenHW Group CORE-V Family

Open Source HW IP for high-volume production SoCs

Rick O'Connor rickoco@openhwgroup.org

T @rickoco T @openhwgroup

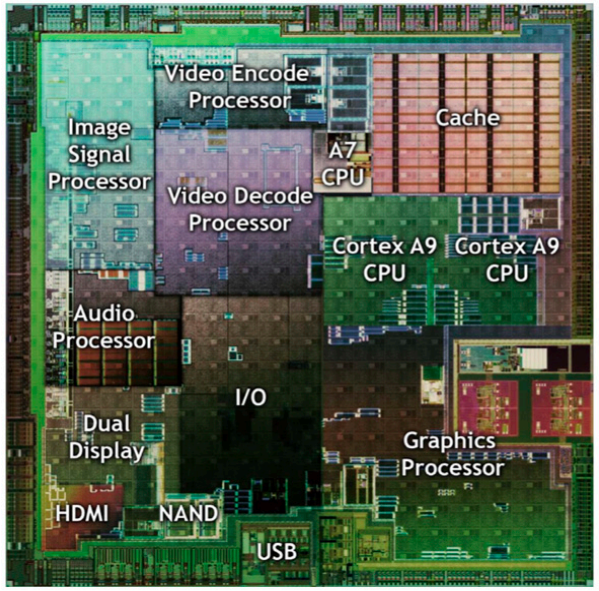
www.openhwgroup.org

Outline

- RISC-V Introduction
  - Free & Open Instruction Set Architecture
- Challenges with SoC design and Open Source IP
- OpenHW Group
  - CORE-V Family of open source RISC-V cores
  - CORE-V Chassis SoC
  - OpenHW Working Groups & Task Groups
- Summary

Most SoCs have many CPUs with many ISAs

- Applications processor (usually ARM)
- Graphics processors
- Image processors
- Radio DSPs
- Audio DSPs
- Security processors
- Power-management processor
- ….



NVIDIA Tegra SoC

- Apps processor ISA too large for base accelerator ISA
- IP bought from different places, each proprietary ISA
- Home-grown ISA cores
- Over a dozen ISAs on some SoCs – each with unique software stack

Why so Many ISAs?

Must they be proprietary?

What if there was one free and open ISA everyone could use across all computing devices?

RISC-V Background

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, 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 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
  - More extensions being defined: P – packed SIMD; B – Bit Manipulation; V – vector instructions; etc.
- All the above are a fairly standard RISC encoding in a fixed 32-bit instruction format

RV32I

RISC-V ①

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`

②

+14
Privileged

+ 8 for M

+ 11 for A

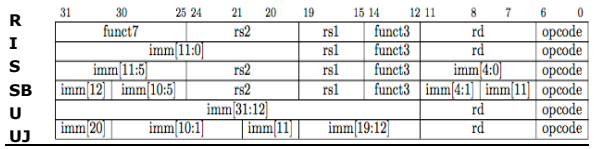
+ 34
for F, D, Q

+ 46 for C

③ **RISC-V Reference Card ④**

CORE-V™

32-bit Instruction Formats



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

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

CORE-V™

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}

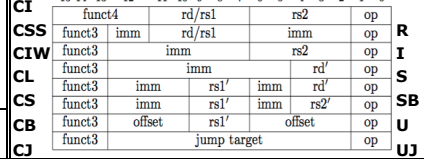
Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)	
Load	Load	I	FL{W,D,Q}	rd,rs1,imm
Store	Store	S	FS{W,D,Q}	rs1,rs2,imm
Arithmetic	ADD	R	FADD.{S|D|Q}	rd,rs1,rs2
	SUBtract	R	FSUB.{S|D|Q}	rd,rs1,rs2
	MULtiply	R	FMUL.{S|D|Q}	rd,rs1,rs2
	DIVide	R	FDIV.{S|D|Q}	rd,rs1,rs2
	SQuare RooT	R	FSQRT.{S|D|Q}	rd,rs1
Mul-Add	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 Type	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	FSFLAGS	rd,rs1
	Swap Rounding Mode Imm	I	FSRMI	rd,imm
	Swap Flags Imm	I	FSFLAGSI	rd,imm

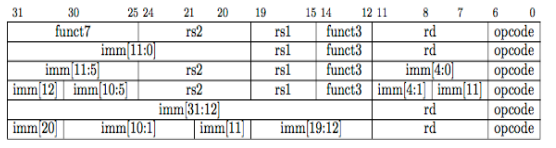
+6 for 64{F|D|Q}/ 128{F|D|Q}

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





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

RISC-V "Green Card"

RISC-V Reference Card ①②③④

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 Type	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	FSFLAGS rd,rs1
	Swap Rounding Mode Imm	I	FSRMI rd,imm
	Swap Flags Imm	I	FSFLAGSI rd,imm

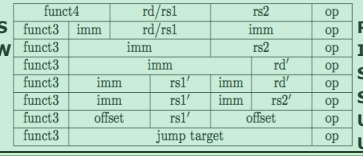
3 Optional FP Extensions: RV{64|128}{F|D|Q}

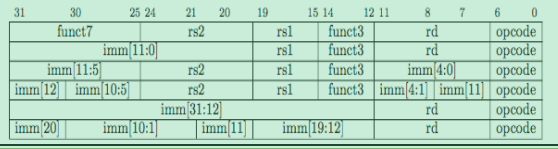
Category	Name	Fmt	RV{F|D|Q} (HP/SP,DP,QP)
Move	Move from Integer	R	FMV.{D|Q}.X rd,rs1
	Move to Integer	R	FMV.X.{D|Q} rd,rs1
Convert	Convert from Int	R	FCVT.{S|D|Q}.{L|T} rd,rs1
	Convert from Int Unsigned	R	FCVT.{S|D|Q}.{L|T}U rd,rs1
	Convert to Int	R	FCVT.{L|T}.{S|D|Q} rd,rs1
	Convert to Int Unsigned	R	FCVT.{L|T}U.{S|D|Q} rd,rs1

Optional Compressed Instructions: RVC

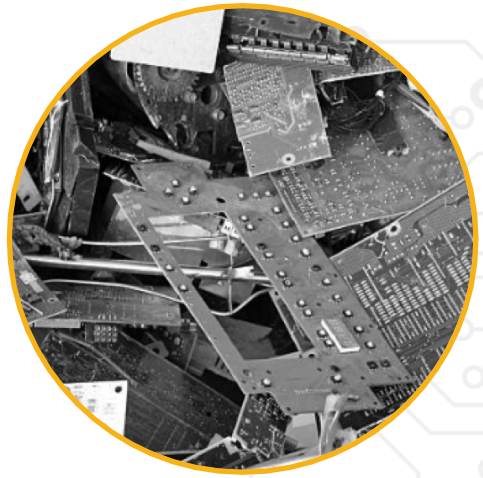
Category	Name	Fmt	RVC
Loads	Load Word	CL	C.LW rd',rs1',imm
	Load Word SP	CI	C.LWSP rd,imm
	Load Double	CL	C.LD rd',rs1',imm
	Load Double SP	CI	C.LWSP rd,imm
	Load Quad	CL	C.LQ rd',rs1',imm
	Load Quad SP	CI	C.LQSP rd,imm
	Load Byte Unsigned	CL	C.LBU rd',rs1',imm
	Float Load Word	CL	C.FLW rd',rs1',imm
	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

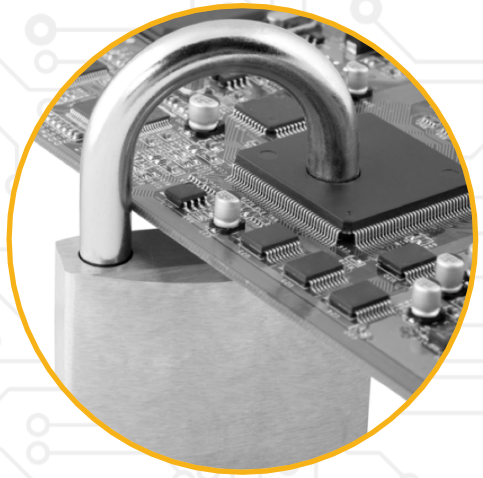




New workloads require architecture flexibility – a new innovation frontier



Legacy ISAs Are Decades Old

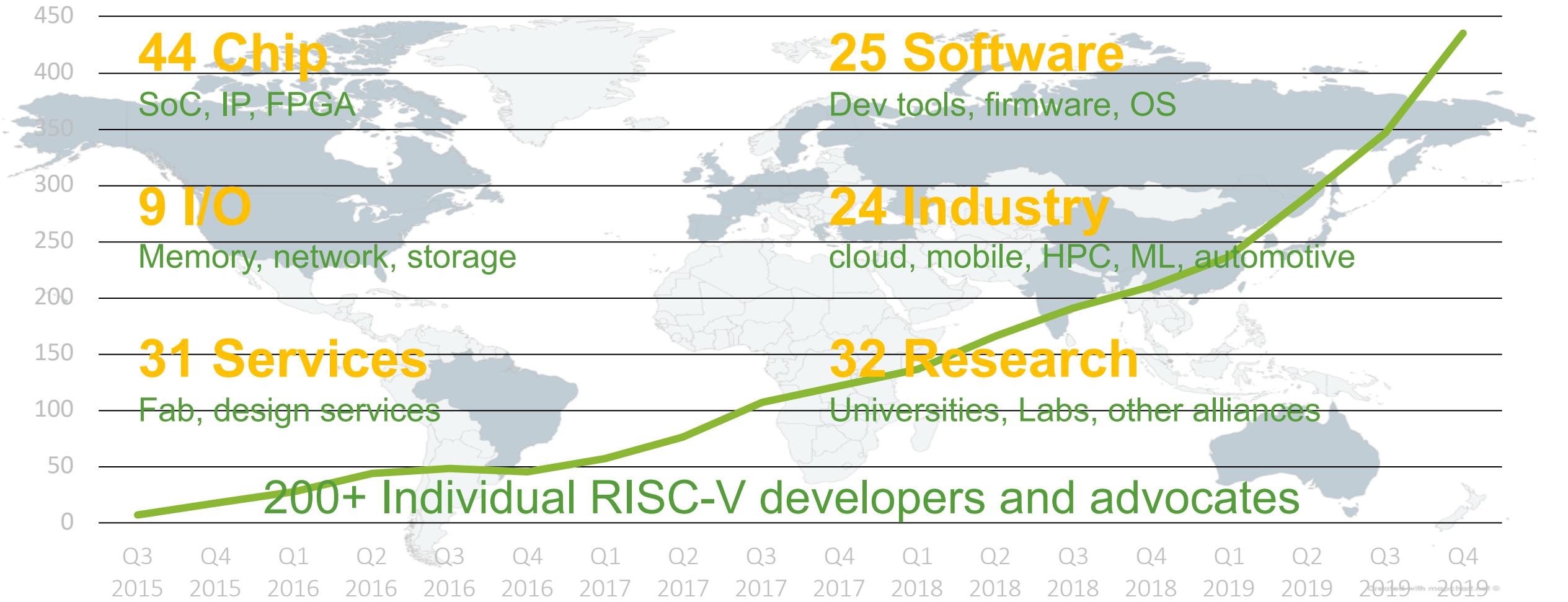


RISC-V Unlocks the Architecture & Enables Innovation

RISC-V ISA is Free, Open and Transparent

Source: RISC-V Foundation

More than 435 RISC-V Members
across 33 Countries Around the World



Source: RISC-V Foundation

Source: RISC-V Foundation

Outline

- RISC-V Introduction
  - Free & Open Instruction Set Architecture
- Challenges with SoC design and Open Source IP
- OpenHW Group
  - CORE-V Family of open source RISC-V cores
  - CORE-V Chassis SoC
  - OpenHW Working Groups & Task Groups
- Summary

SoC Development Cost Drivers

- Software, RTL design, Verification and Physical design account for ~90% of overall SoC development costs
- For highly differentiated IP blocks and functions, this investment is warranted
- For general purpose CPU cores an effective open-source model can drive down these development costs and increase re-use across the industry

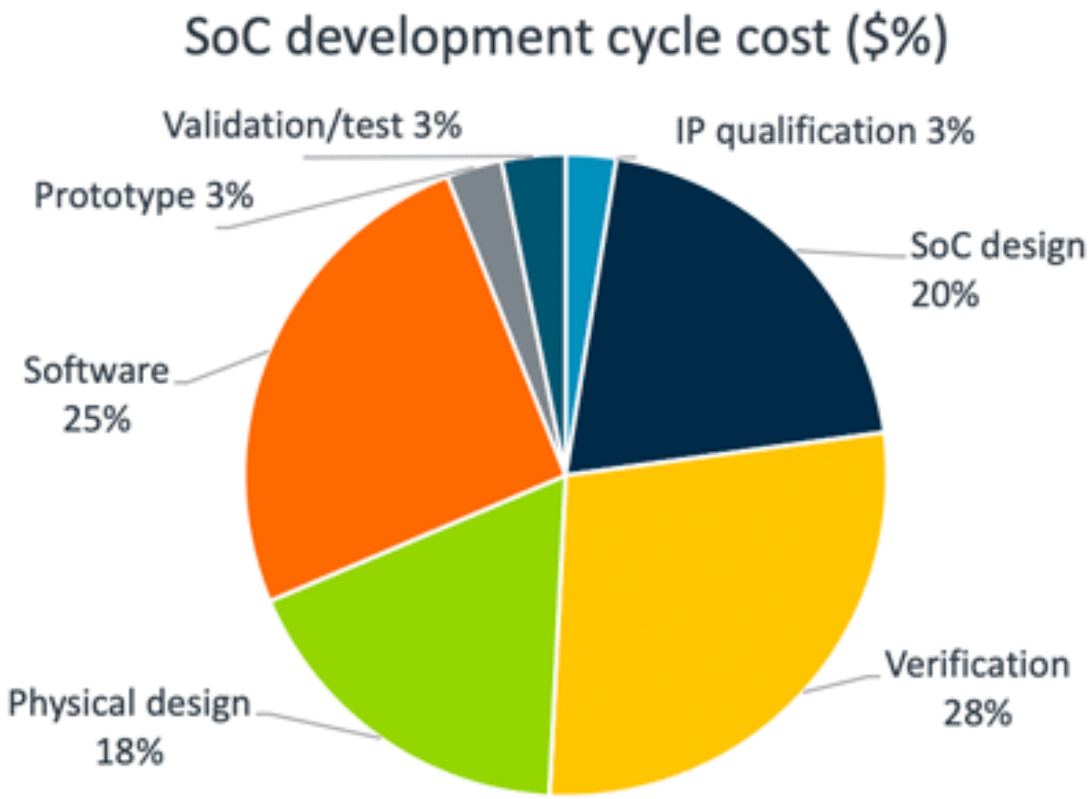


Image Source: Arm Ecosystem Blog

Barriers to adoption of open-source IP

- IP quality
  - harness community best-in-class design and verification methods and contributions
- Ecosystem
  - ensure availability of IDE, RTOS / OS ports, physical design etc. and create a roadmap of cores covering a range of PPA metrics
- Permissive use
  - permissive open-source licensing and processes to minimize business and legal risks

RISC-V ISA Brings Open Source Paradigm to CPU Design

- The free and open RISC-V ISA unleashes a new frontier of processor design and innovation
- How many open source processor implementations do we need as an industry?
  - Open cores are great from a pedagogical teaching perspective, but how many is too many for widespread industry adoption?
- How does the industry, ecosystem, community organize to ensure open core success?
  - How do we establish critical mass around a handful of open cores?

Many RISC-V Open Source Cores...
...and counting....

(source: riscv.org)

Name	Maintainer	Links	User spec	License
rocket	SiFive, UCB Bar	GitHub	2.3-draft	BSD
freedom	SiFive	GitHub	2.3-draft	BSD
Berkeley Out-of-Order Machine (BOOM)	Esperanto, UCB Bar	GitHub	2.3-draft	BSD
ORCA	VectorBlox	GitHub	RV32IM	BSD
RI5CY	ETH Zurich, Università di Bologna	GitHub	RV32IMC	Solderpad Hardware License v. 0.51
Zero-riscy	ETH Zurich, Università di Bologna	GitHub	RV32IMC	Solderpad Hardware License v. 0.51
Ariane	ETH Zurich, Università di Bologna	Website,GitHub	RV64GC	Solderpad Hardware License v. 0.51
Riscy Processors	MIT CSAIL CSG	Website,GitHub		MIT
RiscyOO	MIT CSAIL CSG	GitHub	RV64IMAFD	MIT
Lizard	Cornell CSL BRG	GitHub	RV64IM	BSD

Name	Maintainer	Links	User spec	License
Minerva	LambdaConcept	GitHub	RV32I	BSD
OPenV/mriscv	OnChipUIS	GitHub	RV32I(?)	MIT
VexRiscv	SpinalHDL	GitHub	RV32I[M][C]	MIT
Roa Logic RV12	Roa Logic	GitHub	2.1	Non-Commercial License
SCR1	Syntacore	GitHub	2.2, RV32I/E[MC]	Solderpad Hardware License v. 0.51
Hummingbird E200	Bob Hu	GitHub	2.2, RV32IMAC	Apache 2.0
Shakti	IIT Madras	Website,GitLab	2.2, RV64IMAFDC	BSD
ReonV	Lucas Castro	GitHub		GPL v3
PicoRV32	Clifford Wolf	GitHub	RV32I/E[MC]	ISC
MR1	Tom Verbeure	GitHub	RV32I	Unlicense
SERV	Olof Kindgren	GitHub	RV32I	ISC
SweRV EH1	Western Digital Corporation	GitHub	RV32IMC	Apache 2.0
Reve-R	Gavin Stark	GitHub	RV32IMAC	Apache 2.0

Outline

- RISC-V Introduction
  - Free & Open Instruction Set Architecture
- Challenges with SoC design and Open Source IP
- OpenHW Group
  - CORE-V Family of open source RISC-V cores
  - CORE-V Chassis SoC
  - OpenHW Working Groups & Task Groups
- Summary

- OpenHW Group is a not-for-profit, global organization driven by its members and individual contributors where HW and SW designers collaborate in the development of open-source cores, related IP, tools and SW such as the CORE-V Family of open-source RISC-V cores
- OpenHW Group provides an infrastructure for hosting high quality open-source HW developments in line with industry best practices
- Strong ecosystem support with 40+ members and partners worldwide

Industry Ecosystem
40+ Members & Partners

Research Ecosystem
40+ Members & Partners

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ETH zürich

uOttawa

UNIVERSITY OF TORONTO

THE UNIVERSITY OF UTAH®

Partner Ecosystem
40+ Members & Partners

Legal, Accounting, Banking

CORE-V™ Family of RISC-V Cores

- Initial contribution of open source RISC-V cores from ETH Zurich PULP Platform
  - Very popular, industry adopted cores
- OpenHW Group becomes the official committer for these repositories

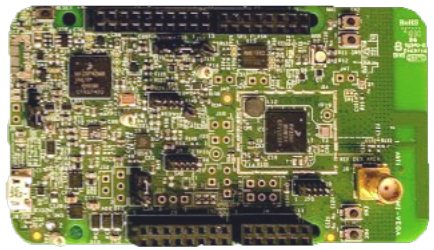
Core	Bits/Stages	Description
CV32 (RI5CY)	32bit / 4-stage	A 4-stage core that implements, the RV32IMFCXpulp, has an optional 32-bit FPU supporting the F extension and instruction set extensions for DSP operations, including hardware loops, SIMD extensions, bit manipulation and post-increment instructions.
CV64 (Ariane)	64bit / 6-stage	A 6-stage, single issue, in-order CPU implementing RV64GC extensions with three privilege levels M, S, U to fully support a Unix-like (Linux, BSD, etc.) operating system. It has configurable size, separate TLBs, a hardware PTW and branch-prediction (branch target buffer, branch history table and a return address stack).

OpenHW Group 2020 BHAG

BIG HARRY AUDACIOUS GOAL

CORE-V™ Chassis

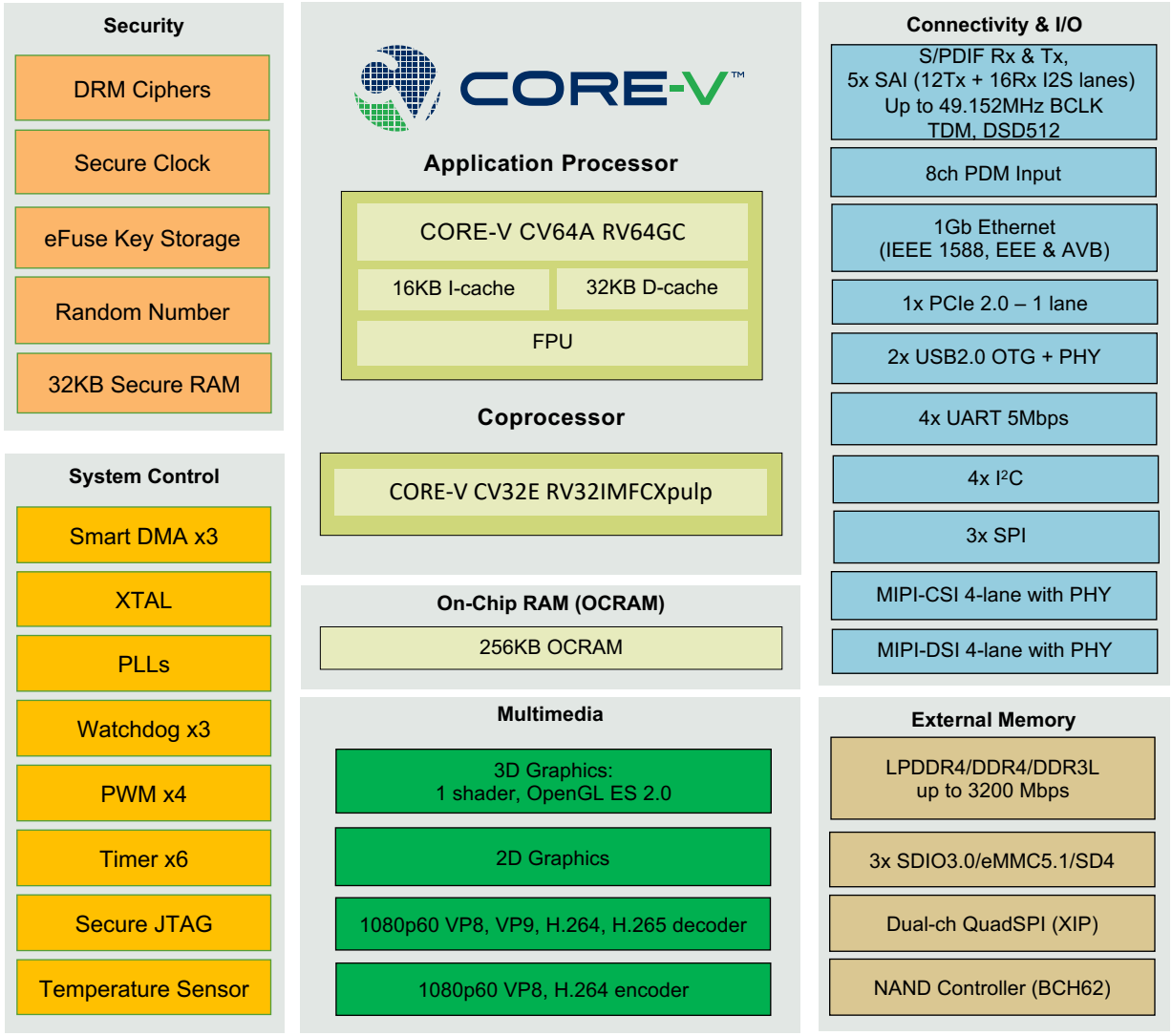
- production ready,
- CORE-V CV64A & CV32E cores,
- deep sub-micron SoC,
- on an eval board,
- running Linux / Zephyr
- Tapeout 2H, 2020



CORE-V™ Chassis – tapeout 2H 2020

CORE-V Chassis based on NXP iMX Platform CORE-V™

- Project announced with NXP at RISC-V Summit Dec 2019
- Linux capable 1.5GHz CV64A host CPU and CV32E coprocessor
- X32/x16 (LP)DDR4, DDR3L memory
- 3D / 2D GPUs with OpenGL support
- MIPI-DSI / CSI display / camera controllers
- Security: DRM Ciphers, key storage, random number generator, etc.
- GigE MAC
- PCIe 2.0 x1 port
- 2 USB 2.0 interfaces
- 3 SDIO interfaces for boot source, storage, etc



CORE-V™ Chassis SoC Ecosystem

CV32 & CV64 Cores

System Verilog RTL

RTL Simulation
Formal Methods
Stimulus

Cloud Based Verification

Device Under Test

GCC / LLVM & OS ports

MPW Layout & Fab

CORE-V Chassis

Protos

Eval Boards

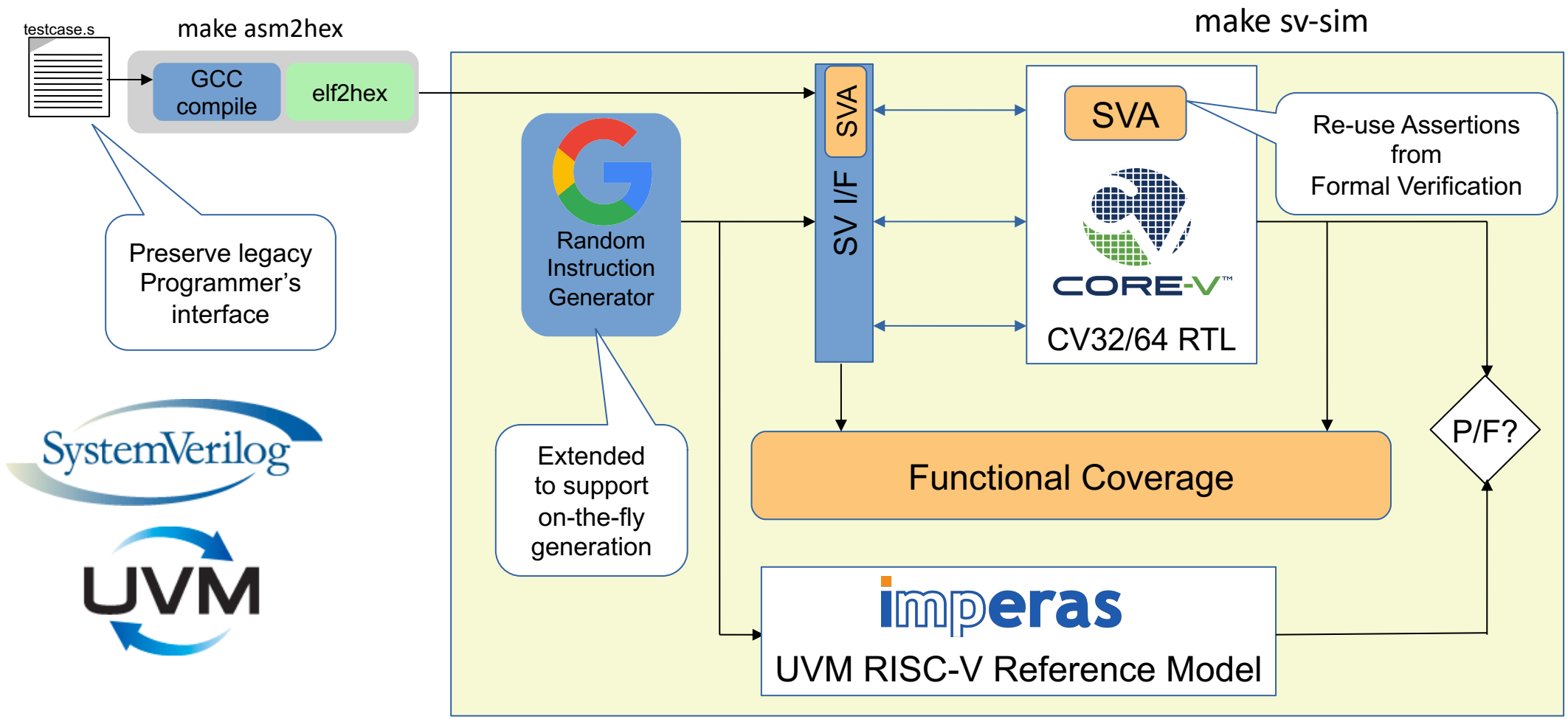
Outline

- RISC-V Introduction
  - Free & Open Instruction Set Architecture
- Challenges with SoC design and Open Source IP
- OpenHW Group
  - CORE-V Family of open source RISC-V cores
  - CORE-V Chassis SoC
  - OpenHW Working Groups & Task Groups
- Summary

Working Groups & Task Groups

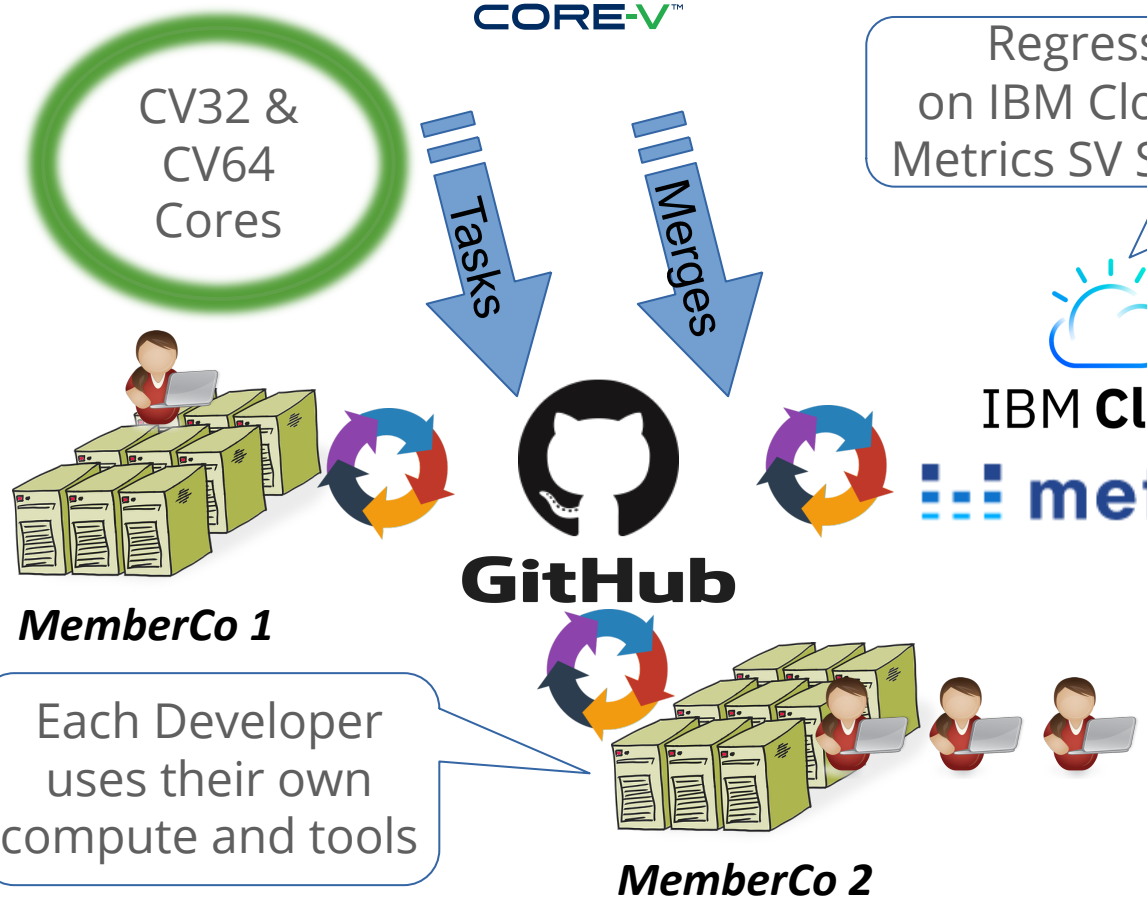
- Board appoints Chairs of ad-hoc working groups and has final approval of working group recommendations
  - Technical Working Group and Marketing Working Group will be standing working groups
- Together with internal OpenHW Group engineering staff, member company development engineers establish and execute OpenHW Group projects
- Technical Working Group https://www.openhwgroup.org/projects/
  - Cores Task Group
  - Verification Task Group
  - Platform Task Group
- Marketing Working Group
  - Content Task Group
  - Events Task Group

OpenHW Group Verification TG
UVM Environment (under development)



Verification TG Collaboration

https://github.com/openhwgroup/core-v-verif



- Most comprehensive and robust System Verilog RISC-V Verification Test Bench available across the industry
  - Multi-company contributed open-source verification stimulus

Outline

- RISC-V Introduction
  - Free & Open Instruction Set Architecture
- Challenges with SoC design and Open Source IP
- OpenHW Group
  - CORE-V Family of open source RISC-V cores
  - CORE-V Chassis SoC
  - OpenHW Working Groups & Task Groups
- Summary

- Open-source, not-for-profit corporation
  - International footprint with developers in North America, Europe and Asia
  - Strong support from industry, academia and individual contributors
- OpenHW Group & CORE-V Family of open-source RISC-V cores
  - Proven System Verilog CV64A and CV32E core designs, processor sub-system IP blocks, verification test bench, and reference designs
  - CORE-V Chassis project
  - Industry proven IDEs, a wide range of RTOS/OS ports and extensive libraries to build necessary software stacks
  - Validated EDA tool flows and proven PPA characteristics
  - Visit www.openhwgroup.org for further details
- Follow us on Social media
  - Twitter @openhwgroup and LinkedIn OpenHW Group
- Strong supporting testimonials from 40+ members & partners