Risc-V : Introduction to the open era of computing

Anand Venkitasubramani , Programmable Solutions Group, Intel

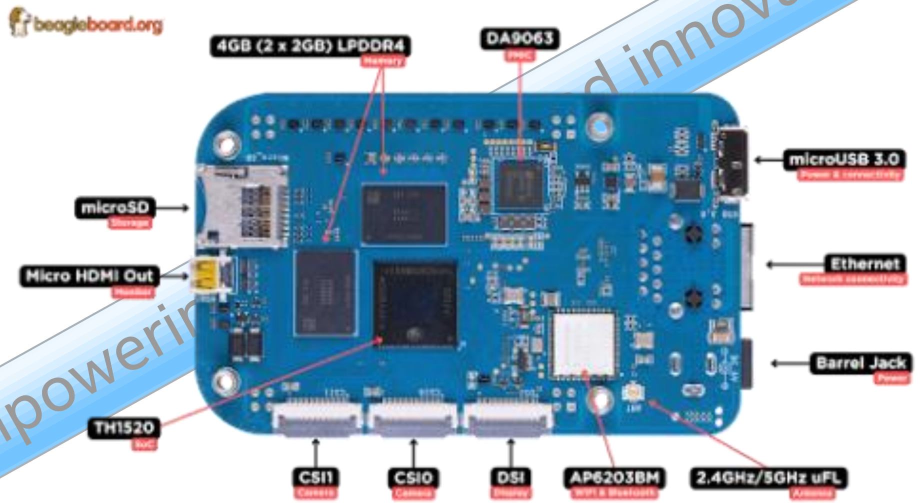
04 November 2023

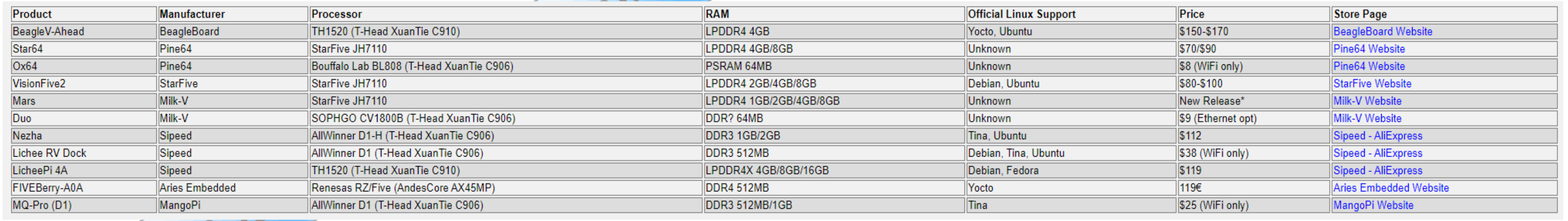
intel®

Agenda

- Risc-V overview
- Risc-V in Intel
- Risc-V opportunities at Intel
- Summary

Ice breaker





Product	Manufacturer	Processor	RAM	Official Linux Support	Price	Store Page
BeagleV-Ahead	BeagleBoard	TH1520 (T-Head XuanTie C910)	LPDDR4 4GB	Yocto, Ubuntu	$150-$170	BeagleBoard Website
Star64	Pine64	StarFive JH7110	LPDDR4 4GB/8GB	Unknown	$70/$90	Pine64 Website
Ox64	Pine64	Bouffalo Lab BL808 (T-Head XuanTie C906)	PSRAM 64MB	Unknown	$8 (WiFi only)	Pine64 Website
VisionFive2	StarFive	StarFive JH7110	LPDDR4 2GB/4GB/8GB	Debian, Ubuntu	$80-$100	StarFive Website
Mars	Milk-V	StarFive JH7110	LPDDR4 1GB/2GB/4GB/8GB	Unknown	New Release*	Milk-V Website
Duo	Milk-V	SOPHGO CV1800B (T-Head XuanTie C906)	DDR? 64MB	Unknown	$9 (Ethernet opt)	Milk-V Website
Nezha	Sipeed	AllWinner D1-H (T-Head XuanTie C906)	DDR3 1GB/2GB	Tina, Ubuntu	$112	Sipeed - AliExpress
Lichee RV Dock	Sipeed	AllWinner D1 (T-Head XuanTie C906)	DDR3 512MB	Debian, Tina, Ubuntu	$38 (WiFi only)	Sipeed - AliExpress
LicheePi 4A	Sipeed	TH1520 (T-Head XuanTie C910)	LPDDR4X 4GB/8GB/16GB	Debian, Fedora	$119	Sipeed - AliExpress
FIVEBerry-A0A	Aries Embedded	Renesas RZ/Five (AndesCore AX45MP)	DDR4 512MB	Yocto	119€	Aries Embedded Website
MQ-Pro (D1)	MangoPi	AllWinner D1 (T-Head XuanTie C906)	DDR3 512MB/1GB	Tina	$25 (WiFi only)	MangoPi Website

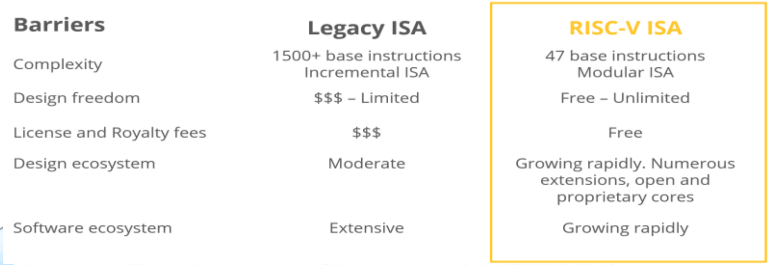
https://github.com/riscvarchive/riscv-cores-list

Risc-V : Overview

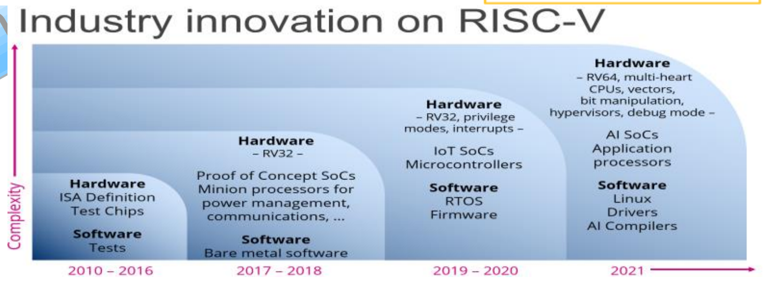
- Started as a research project in Berkeley in 2010
- Open, non-profit standard
- ISA and extensions are royalty-free
- Can build solutions and services on top of it
- Open-ness leads to more contribution and innovation
- Rapid adoption by IP providers, SoC companies, OEMs

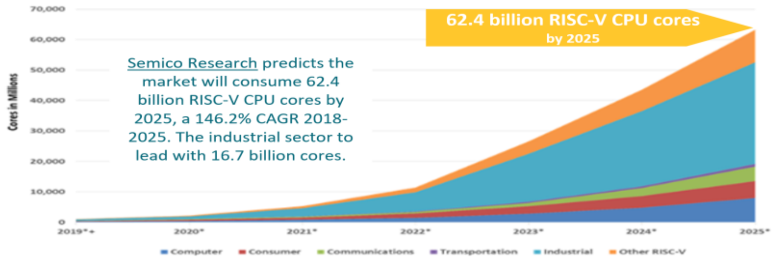
- Modular ISA with small base ISA and several optional extensions
- IEEE-754 floating point support
- 32 and 64bit address space
- Supports multicores
- Compressed instruction set
- Allows custom extensions
- 4 Privilege levels
- 32 registers for base ISA

- Source https://riscv.org/



Barriers	Legacy ISA	RISC-V ISA
Complexity	1500+ base instructions Incremental ISA	47 base instructions Modular ISA
Design freedom	$$$ – Limited	Free – Unlimited
License and Royalty fees	$$$	Free
Design ecosystem	Moderate	Growing rapidly. Numerous extensions, open and proprietary cores
Software ecosystem	Extensive	Growing rapidly





Semico Research predicts the market will consume 62.4 billion RISC-V CPU cores by 2025, a 146.2% CAGR 2018-2025. The industrial sector to lead with 16.7 billion cores.

Risc-V benefits

•Its open-standard nature, which allows collaboration and innovation across the industry

•Common ISA, which helps make software development easier since all processors could potentially use the same architecture.

•Availability of smaller, energy-efficient, and modular options

•Security features, which are available through open-source reference designs, software composition analysis tools, and security extensions.

Risc-V applications

•**Wearables, Industrial, IoT, and Home Appliances.** RISC-V processors are ideal for meeting the power requirements of space-constrained and battery-operated designs.

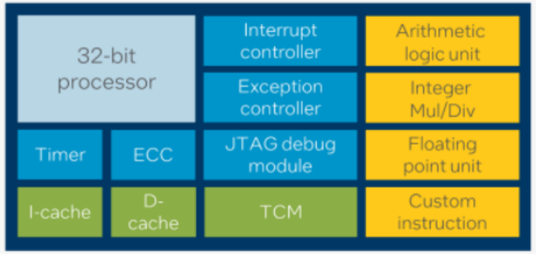
•**Smartphones.**

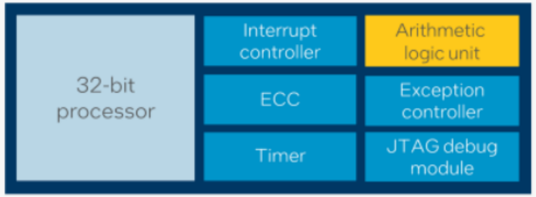
•**Automotive, High-Performance Computing (HPC), and Data Centers.**

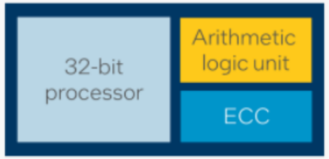
•**Aerospace and Government.** RISC-V offers high reliability and security for these use applications.

•**List is endless**

Risc-V in Intel FPGA







Nios® V/g General Purpose Processor

- RV32IMA(F)Zicsr_Zicbom
- Highest performance Nios V processor
- Supports RTOS embedded system

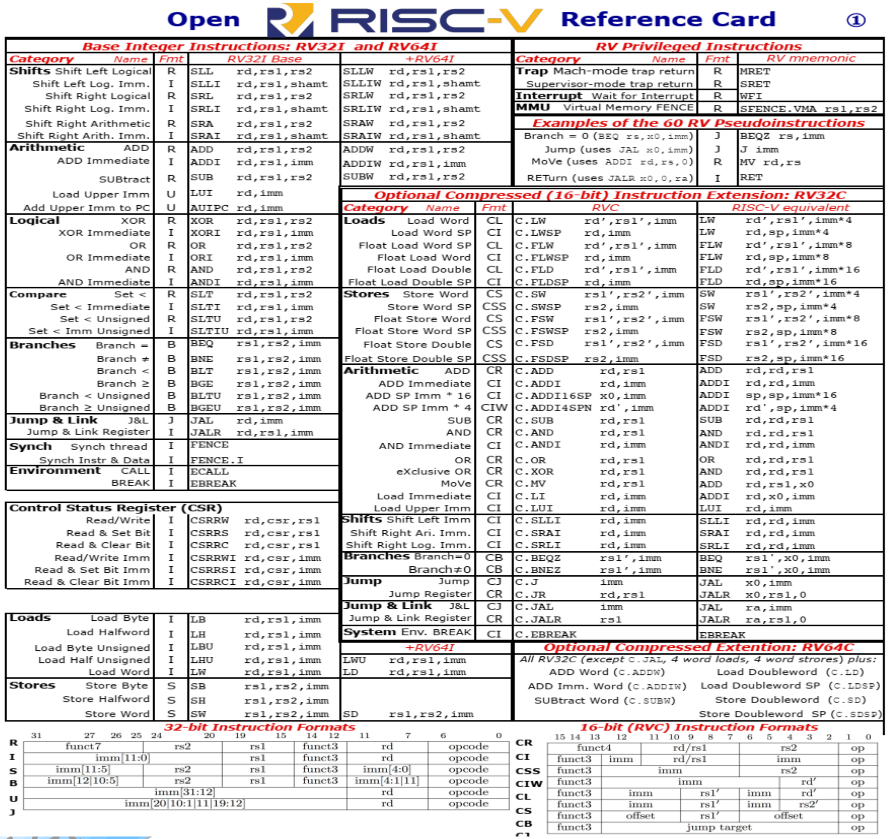
Nios® V/m Microcontroller

- RV32IAZicsr (Pipelined) & RV32IZicsr (Non-Pipelined)
- Supports RTOS embedded system
- Balanced for performance and size

Nios® V/c Compact Microcontroller

- RV32I
- No debug
- Smallest Nios V processor for non-interrupt-driven control application

Risc-V ISA



Open RISC-V Reference Card ①

Base Integer Instructions: RV32I and RV64I

Category	Name	Fmt	RV32I Base	+RV64I
Shifts	Shift Left Logical	R	SLL rd,rs1,rs2	SLLW rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI rd,rs1,shamt	SLLIW rd,rs1,shamt
	Shift Right Logical	R	SRL rd,rs1,rs2	SRLW rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI rd,rs1,shamt	SRLIW rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRAW rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI rd,rs1,shamt	SRAIW rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADDW rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDIW rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUBW 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 =	B	BEQ rs1,rs2,imm	
	Branch ≠	B	BNE rs1,rs2,imm	
	Branch <	B	BLT rs1,rs2,imm	
	Branch ≥	B	BGE rs1,rs2,imm	
	Branch < Unsigned	B	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	B	BGEU rs1,rs2,imm	
Jump & Link	J&L	J	JAL rd,imm	
	Jump & Link Register	I	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	

Control Status Register (CSR)

	Name	Fmt	RV32I Base
	Read/Write	I	CSRRW rd,csr,rs1
	Read & Set Bit	I	CSRRS rd,csr,rs1
	Read & Clear Bit	I	CSRRC rd,csr,rs1
	Read/Write Imm	I	CSRRWI rd,csr,imm
	Read & Set Bit Imm	I	CSRRSI rd,csr,imm
	Read & Clear Bit Imm	I	CSRRCI rd,csr,imm

Category	Name	Fmt	RV32I Base	+RV64I
Loads	Load Byte	I	LB rd,rs1,imm	
	Load Halfword	I	LH rd,rs1,imm	
	Load Byte Unsigned	I	LBU rd,rs1,imm	LWU rd,rs1,imm
	Load Half Unsigned	I	LHU rd,rs1,imm	LD rd,rs1,imm
	Load Word	I	LW rd,rs1,imm	
Stores	Store Byte	S	SB rs1,rs2,imm	
	Store Halfword	S	SH rs1,rs2,imm	
	Store Word	S	SW rs1,rs2,imm	SD rs1,rs2,imm

RV Privileged Instructions

Category	Name	Fmt	RV mnemonic
Trap	Mach-mode trap return	R	MRET
	Supervisor-mode trap return	R	SRET
Interrupt	Wait for Interrupt	R	WFI
MMU	Virtual Memory FENCE	R	SFENCE.VMA rs1,rs2

Examples of the 60 RV Pseudoinstructions

Branch = 0 (BEQ rs,x0,imm)	J	BEQZ rs,imm
Jump (uses JAL x0,imm)	J	J imm
MoVe (uses ADDI rd,rs,0)	R	MV rd,rs
RETurn (uses JALR x0,0,ra)	I	RET

Optional Compressed (16-bit) Instruction Extension: RV32C

Category	Name	Fmt	RVC	RISC-V equivalent
Loads	Load Word	CL	C.LW rd′,rs1′,imm	LW rd′,rs1′,imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Float Load Word SP	CL	C.FLW rd′,rs1′,imm	FLW rd′,rs1′,imm*8
	Float Load Word	CI	C.FLWSP rd,imm	FLW rd,sp,imm*8
	Float Load Double	CL	C.FLD rd′,rs1′,imm	FLD rd′,rs1′,imm*16
	Float Load Double SP	CI	C.FLDSP rd,imm	FLD rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1′,rs2′,imm	SW rs1′,rs2′,imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Float Store Word	CS	C.FSW rs1′,rs2′,imm	FSW rs1′,rs2′,imm*8
	Float Store Word SP	CSS	C.FSWSP rs2,imm	FSW rs2,sp,imm*8
	Float Store Double	CS	C.FSD rs1′,rs2′,imm	FSD rs1′,rs2′,imm*16
	Float Store Double SP	CSS	C.FSDSP rs2,imm	FSD rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm	ADDI rd',sp,imm*4
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
	AND	CR	C.AND rd,rs1	AND rd,rd,rs1
	AND Immediate	CI	C.ANDI rd,imm	ANDI rd,rd,imm
	OR	CR	C.OR rd,rs1	OR rd,rd,rs1
	eXclusive OR	CR	C.XOR rd,rs1	AND rd,rd,rs1
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
	Shift Right Ari. Imm.	CI	C.SRAI rd,imm	SRAI rd,rd,imm
	Shift Right Log. Imm.	CI	C.SRLI rd,imm	SRLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1′,imm	BEQ rs1',x0,imm
	Branch≠0	CB	C.BNEZ rs1′,imm	BNE rs1',x0,imm
Jump	Jump	CJ	C.J imm	JAL x0,imm
	Jump Register	CR	C.JR rd,rs1	JALR x0,rs1,0
Jump & Link	J&L	CJ	C.JAL imm	JAL ra,imm
	Jump & Link Register	CR	C.JALR rs1	JALR ra,rs1,0
System	Env. BREAK	CI	C.EBREAK	EBREAK

Optional Compressed Extention: RV64C

All RV32C (except C.JAL, 4 word loads, 4 word strores) plus:

ADD Word (C.ADDW)	Load Doubleword (C.LD)
ADD Imm. Word (C.ADDIW)	Load Doubleword SP (C.LDSP)
SUBtract Word (C.SUBW)	Store Doubleword (C.SD)
	Store Doubleword SP (C.SDSP)

32-bit Instruction Formats

	31–27	26–25	24–20	19–15	14–12	11–7	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
B	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
J	imm[20|10:1|11|19:12]					rd	opcode

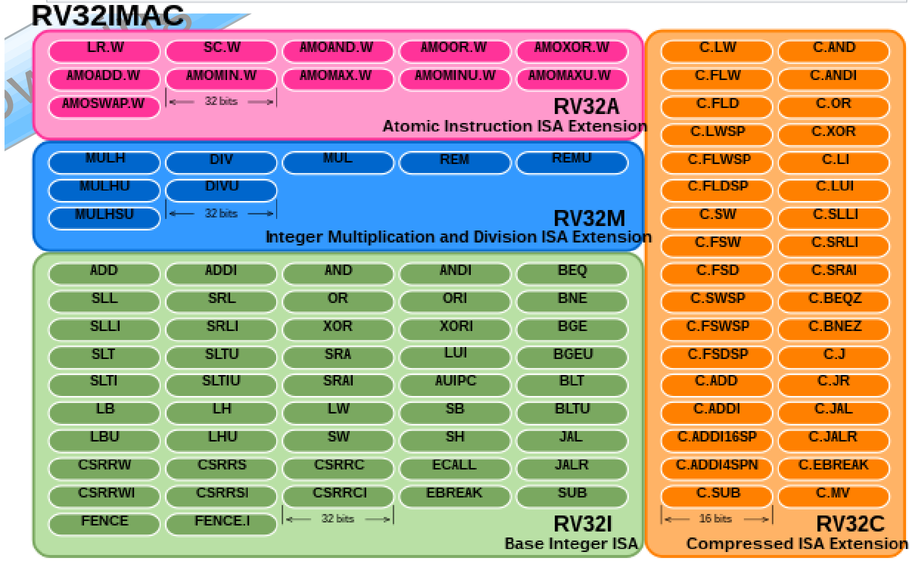
16-bit (RVC) Instruction Formats

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

32-bit RISC-V instruction formats

Format	Bit																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

- *opcode* (7 bits): Partially specifies which of the 6 types of *instruction formats*.
- *funct7* (7 bits) and *funct3* (3 bits): These two fields extend the *opcode* field to specify the operation to be performed.
- *rs1*, *rs2*, or *rd* (5 bits): Specifies, by index, the register, resp., containing the first operand (i.e., source register), second operand, and destination register to which the computation result will be directed.



Risc-V Instructions

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	-
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5	t0	Temporary / alternate link register	Caller
x6-7	t1-2	Temporaries	Caller
x8	s0/fp	Saved-register / frame-pointer	Callee
x9	s1	Saved register	Callee
x10-11	a0-1	Function arguments / return values	Caller
x12-17	a2-7	Function arguments	Caller
x18-27	s2-11	Saved registers	Callee
x28-31	t3-6	Temporaries	Caller
Floating-Point Registers			
f0-7	ft0-7	FP temporaries	Caller
f8-9	fs0-1	FP saved registers	Callee
f10-11	fa0-1	FP arguments / return values	Caller
f12-17	fa2-7	FP arguments	Caller
f18-27	fa2-11	FP saved registers	Callee
f28-31	ft8-11	FP temporaries	Caller

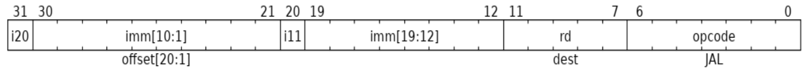
Assembly	C	Description
`add x1,x2,x3`	`a = b + c`	a=x1, b=x2, c=x3
`sub x3,x4,x5`	`d = e - f`	d=x3, e=x4, f=x5
`add x0,x0,x0`	NOP	Writes to `x0` are always ignored
`add x3,x4,x0`	`f = g`	f=x3, g=x4
`addi x3,x4,-10`	`f = g - 10`	f=x3, g=x4

`lw x10,12(x13) # 12 = 3x4`
`add x11,x12,x10`
	`int A[100];`
	`g = h + A[3];`
		Reg `x10` gets A[3]
		g=x11, h=x12

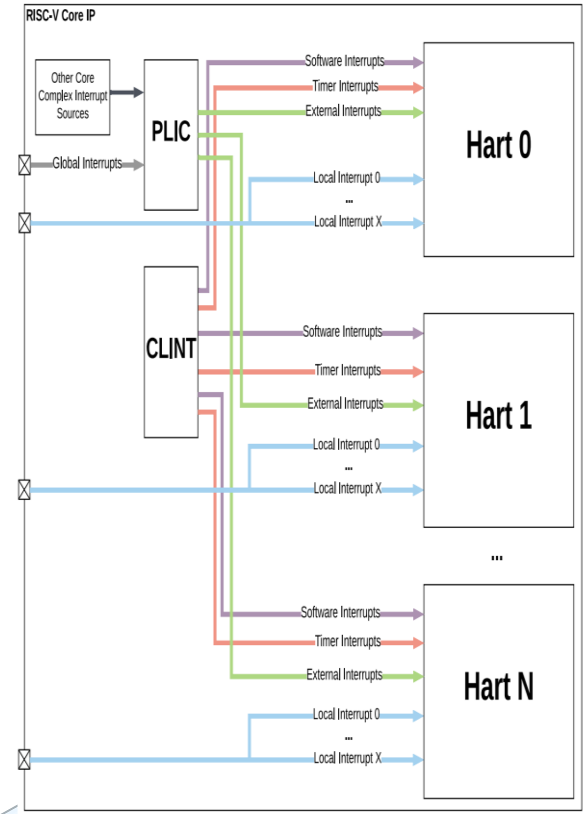
`lw x10,12(x13) # 12 = 3x4`
`add x10,x12,x10`
`sw x10,40(x13) # 40 = 10x4`
	`int A[100];`
	`A[10] = h + A[3];`
		Reg `x10` gets A[3]
		h=x12
		Reg `x10` gets h + A[3]

`  bne x13,x14,done`
`  add x10,x11,x12`
`done:`
	`if (i == j)`
	`  f = g + h;`
		f=x10, g=x11, h=x12, i=x13, j=x14

`  bne x10,x14,else`
`  add x10,x11,x12`
`  j done`
`else: sub x10,x11,x12`
`done:`
	`if (i == j)`
	`  f = g + h;`
	`else`
	`  f = g - h;`
		f=x10, g=x11, h=x12, i=x13, j=x14



Risc-V Interrupts



mtvec + (4 * X)
...
mtvec + 0x40
mtvec + 0x3C
mtvec + 0x38
mtvec + 0x34
mtvec + 0x30
mtvec + 0x2C
mtvec + 0x28
mtvec + 0x24
mtvec + 0x20
mtvec + 0x1C
mtvec + 0x18
mtvec + 0x14
mtvec + 0x10
mtvec + 0x0C
mtvec + 0x08
mtvec + 0x04
mtvec + 0x00

CLINT Machine Mode Interrupt Vector Table

Local Interrupts, ID: 16...X

External Interrupt, ID: 11

Timer Interrupt, ID: 7

Software Interrupt, ID: 3

Reserved

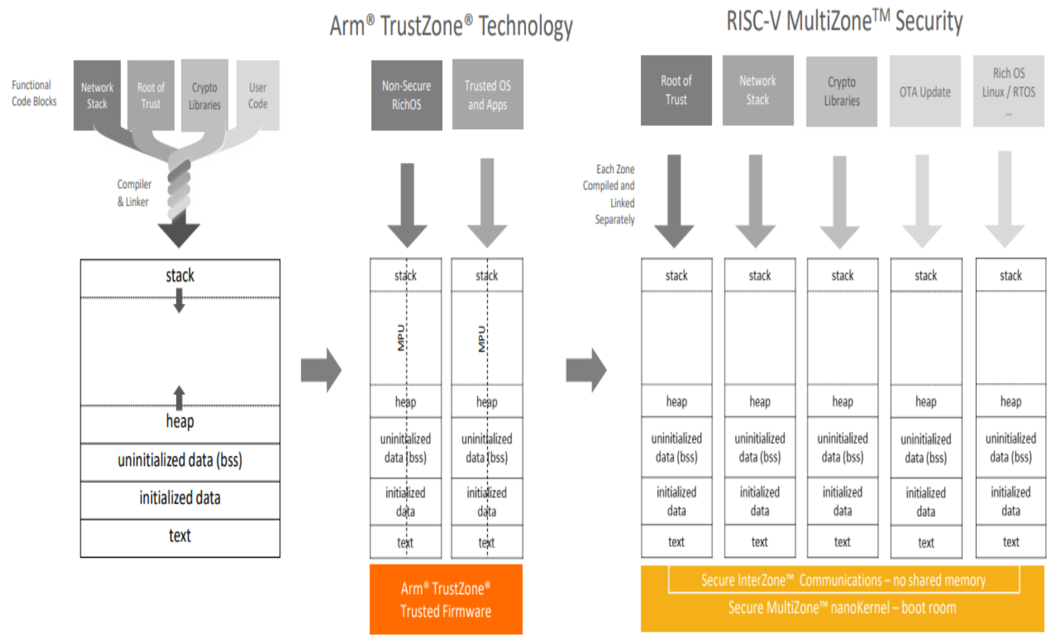
Vector Table Base Address
mtvec + (4 * Interrupt ID)

CSR	Description
`mcause`	Contains the cause value of the exception/interrupt. See Section 8.7.5 for more description.
`mepc`	Contains the `pc` where the exception occurs.
`mtval`	If the cause is a load/store fault, this register has the value of the problematic address. If it is an invalid instruction, it provides the instruction that the core tried to execute.
`mstatus`	Contains the interrupt enables, privilege modes, and general status of execution. See Section 8.7.1 for more description.
`mtvec`	Contains the vector that the core will jump to when an exception occurs. If this is not a valid executable value, you may get a double exception when jumping to the exception handler, so it is important to look at all these registers when the exception FIRST occurs. See Section 8.7.2 for more description.

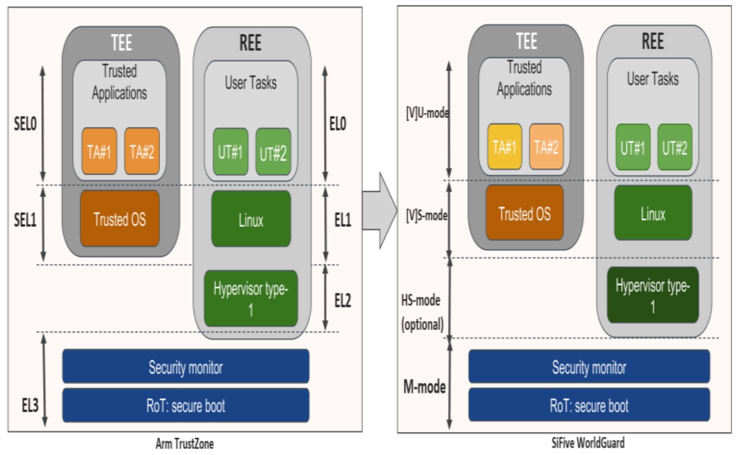
Priority	Exception Code	Description
Highest	3	Instruction address breakpoint
	12	Instruction page fault
	1	Instruction access fault
	2	Illegal instruction
	0	Instruction address misaligned
	8, 9, 11	Environment call
	3	Environment break
	3	Load/Store/AMO address breakpoint
	6	Store/AMO address misaligned
	4	Load address misaligned
	15	Store/AMO page fault
	13	Load page fault
Lowest	7	Store/AMO access fault
	5	Load access fault

Security, access control

Security Through Separation



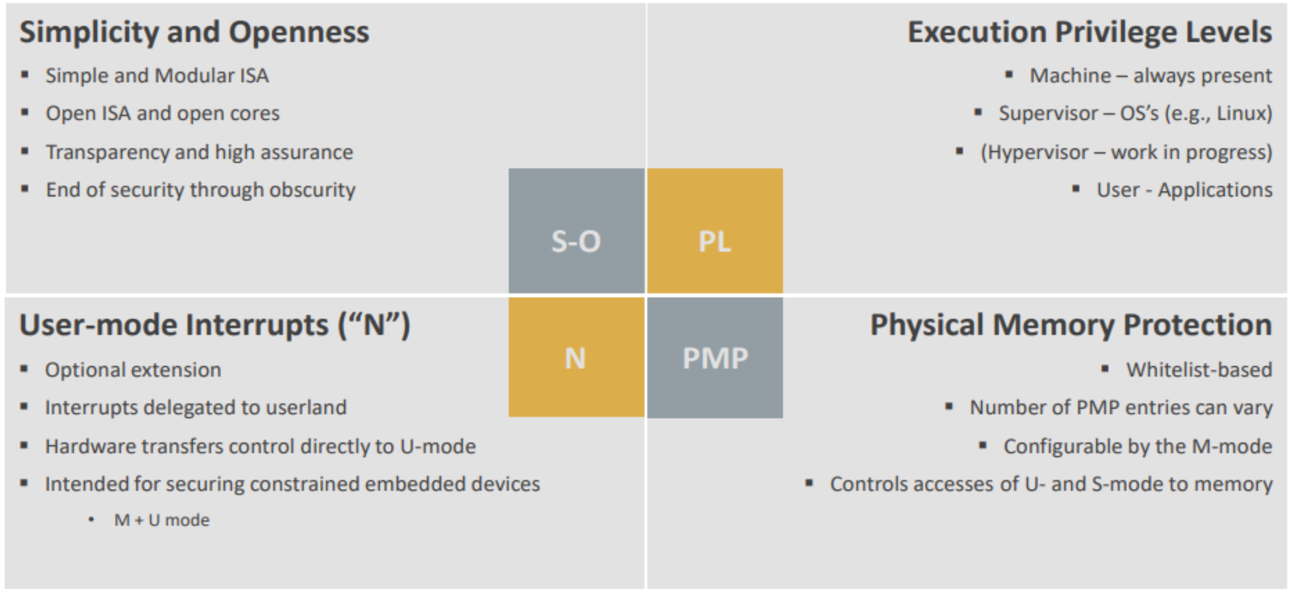
TrustZone (Armv8-A) to WorldGuard



	Arm TrustZone (Armv8-A)		
	EL	TZ Mode	Priv. Mode
RoT/Secure Boot	EL3	Secure	Privileged
Security monitor		Secure	Privileged
Trusted OS	SEL1	Secure	Privileged
Trusted App #1	SEL0	Secure	Non Privileged
Trusted App #2		Secure	Non Privileged
Hypervisor	EL2	Non Secure	Privileged
Linux	EL1	Non Secure	Privileged
Untrusted App #2	EL0	Non Secure	Non Privileged
Untrusted App #2		Non Secure	Non Privileged

	SiFive WorldGuard	
	WID	Priv. Mode
RoT/Secure Boot	6	M
Security monitor	6	M
Trusted OS	2	S
Trusted App #1	4	U
Trusted App #2	5	U
Hypervisor	1	HS
Linux	3	VS
Untrusted App #2	3	VU
Untrusted App #2	3	VU

Security, access control in Risc-V



Simplicity and Openness

- Simple and Modular ISA
- Open ISA and open cores
- Transparency and high assurance
- End of security through obscurity

Execution Privilege Levels

- Machine – always present
- Supervisor – OS's (e.g., Linux)
- (Hypervisor – work in progress)
- User - Applications

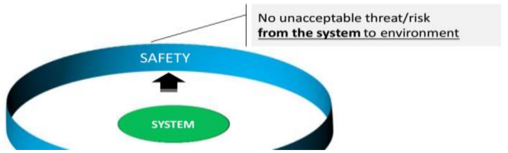
User-mode Interrupts ("N")

- Optional extension
- Interrupts delegated to userland
- Hardware transfers control directly to U-mode
- Intended for securing constrained embedded devices
  - M + U mode

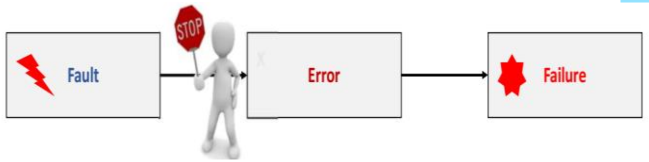
Physical Memory Protection

- Whitelist-based
- Number of PMP entries can vary
- Configurable by the M-mode
- Controls accesses of U- and S-mode to memory

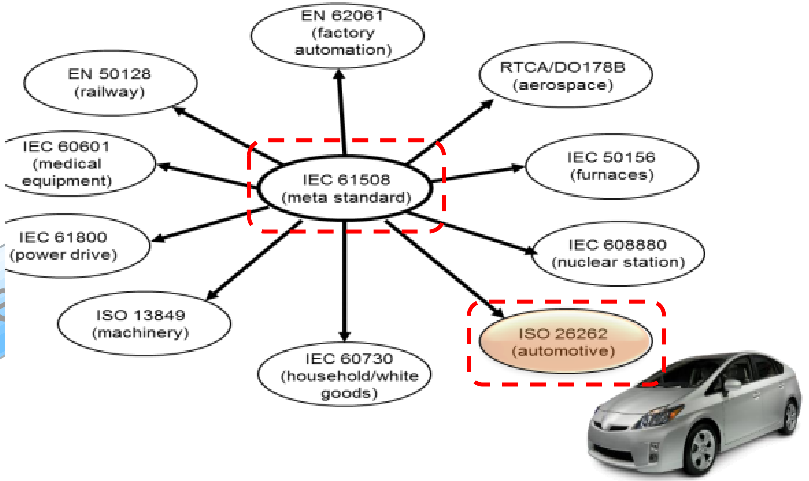
Functional Safety


No unacceptable threat/risk
from the system to environment
SAFETY
SYSTEM

Meeting FuSa is an *art* of finding *possibilities* to achieve safety at a holistic level



Functional safety standards


EN 62061 (factory automation)
EN 50128 (railway)
RTCA/DO178B (aerospace)
IEC 60601 (medical equipment)
IEC 61508 (meta standard)
IEC 50156 (furnaces)
IEC 61800 (power drive)
IEC 608880 (nuclear station)
ISO 13849 (machinery)
IEC 60730 (household/white goods)
ISO 26262 (automotive)

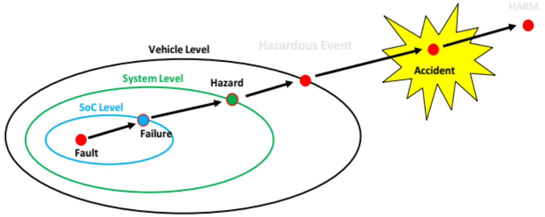
Definition of Functional Safety

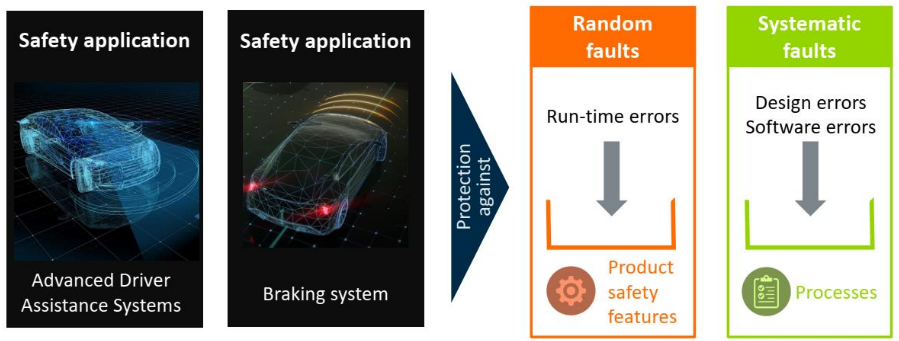
Risk judged to be unacceptable in certain context according to valid societal moral concept

Potential source of harm

Absence of ***unreasonable risk*** due to ***hazards*** caused by **malfunctioning behaviour** of Electrical/Electronic systems.

Failure or unintended behavior of an item with respect to its design intent

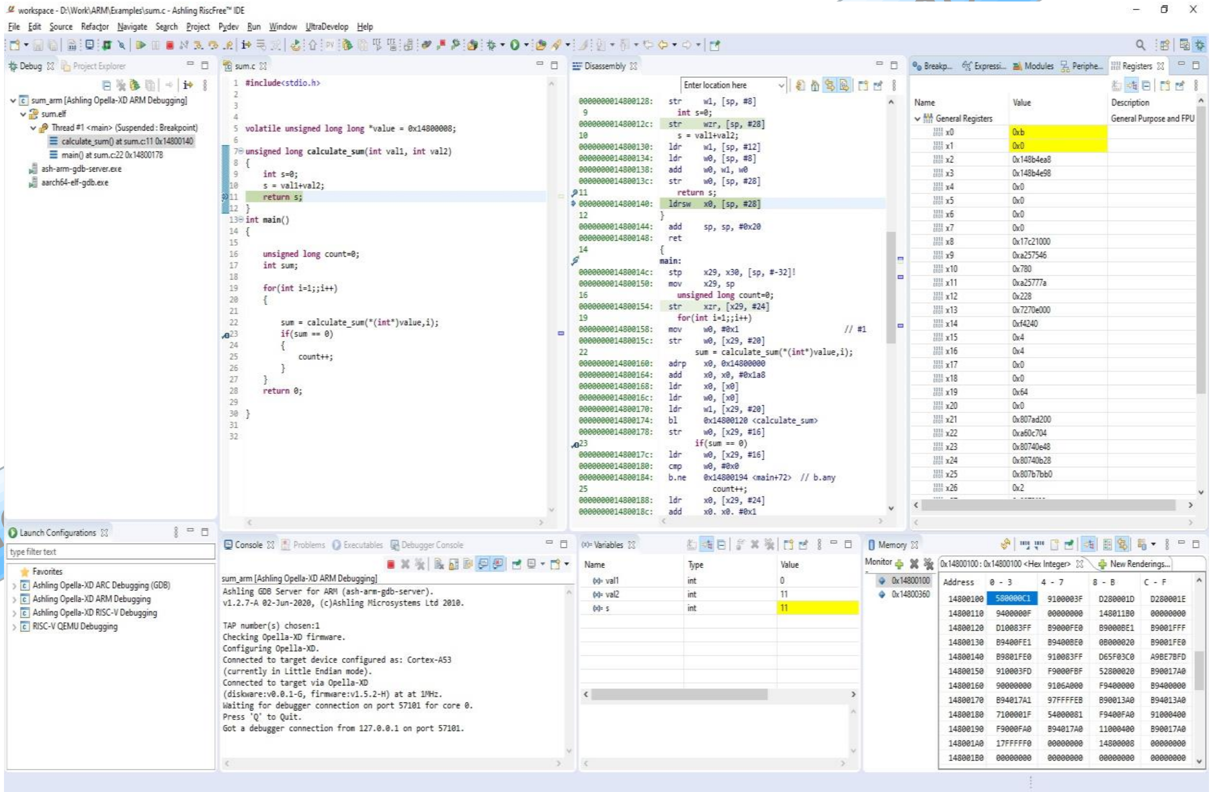




"Absence of unreasonable risk due to hazards caused by malfunctions"

Risc-V Debug, Trace

- Getting competitive with Coresight and external debugger/trace HW, solutions
- Nexus is the trace format



```
sum_arm [Ashling Opella-XD ARM Debugging]
  sum.elf
    Thread #1 <main> (Suspended : Breakpoint)
      calculate_sum() at sum.c:11 0x14800140
      main() at sum.c:22 0x14800178
  ash-arm-gdb-server.exe
  aarch64-elf-gdb.exe
```

```c
#include<stdio.h>



volatile unsigned long long *value = 0x14800008;

unsigned long calculate_sum(int val1, int val2)
{
    int s=0;
    s = val1+val2;
    return s;
}
int main()
{

    unsigned long count=0;
    int sum;

    for(int i=1;;i++)
    {

        sum = calculate_sum(*(int*)value,i);
        if(sum == 0)
        {
            count++;
        }
    }
    return 0;

}
```

```
0000000014800128:  str     w1, [sp, #8]
 9                   int s=0;
000000001480012c:  str     wzr, [sp, #28]
10                   s = val1+val2;
0000000014800130:  ldr     w1, [sp, #12]
0000000014800134:  ldr     w0, [sp, #8]
0000000014800138:  add     w0, w1, w0
000000001480013c:  str     w0, [sp, #28]
11                   return s;
0000000014800140:  ldrsw   x0, [sp, #28]
12                 }
0000000014800144:  add     sp, sp, #0x20
0000000014800148:  ret
14                 {
                   main:
000000001480014c:  stp     x29, x30, [sp, #-32]!
0000000014800150:  mov     x29, sp
16                   unsigned long count=0;
0000000014800154:  str     xzr, [x29, #24]
19                   for(int i=1;;i++)
0000000014800158:  mov     w0, #0x1                    // #1
000000001480015c:  str     w0, [x29, #20]
22                       sum = calculate_sum(*(int*)value,i);
0000000014800160:  adrp    x0, 0x14800000
0000000014800164:  add     x0, x0, #0x1a8
0000000014800168:  ldr     x0, [x0]
000000001480016c:  ldr     w0, [x0]
0000000014800170:  ldr     w1, [x29, #20]
0000000014800174:  bl      0x14800120 <calculate_sum>
0000000014800178:  str     w0, [x29, #16]
23                       if(sum == 0)
000000001480017c:  ldr     w0, [x29, #16]
0000000014800180:  cmp     w0, #0x0
0000000014800184:  b.ne    0x14800194 <main+72>  // b.any
25                           count++;
0000000014800188:  ldr     x0, [x29, #24]
000000001480018c:  add     x0. x0. #0x1
```

Name	Value	Description
General Registers		General Purpose and FPU
x0	0xb	
x1	0x0	
x2	0x148b4ea8	
x3	0x148b4e98	
x4	0x0	
x5	0x0	
x6	0x0	
x7	0x0	
x8	0x17c21000	
x9	0xa257546	
x10	0x780	
x11	0xa25777a	
x12	0x228	
x13	0x7270e000	
x14	0xf4240	
x15	0x4	
x16	0x4	
x17	0x0	
x18	0x0	
x19	0x64	
x20	0x0	
x21	0x807ad200	
x22	0xa60c704	
x23	0x80740e48	
x24	0x80740b28	
x25	0x807b7bb0	
x26	0x2	

```
Favorites
Ashling Opella-XD ARC Debugging (GDB)
Ashling Opella-XD ARM Debugging
Ashling Opella-XD RISC-V Debugging
RISC-V QEMU Debugging
```

```
sum_arm [Ashling Opella-XD ARM Debugging]
Ashling GDB Server for ARM (ash-arm-gdb-server).
v1.2.7-A 02-Jun-2020, (c)Ashling Microsystems Ltd 2010.

TAP number(s) chosen:1
Checking Opella-XD firmware.
Configuring Opella-XD.
Connected to target device configured as: Cortex-A53
(currently in Little Endian mode).
Connected to target via Opella-XD
(diskware:v0.0.1-G, firmware:v1.5.2-H) at at 1MHz.
Waiting for debugger connection on port 57101 for core 0.
Press 'Q' to Quit.
Got a debugger connection from 127.0.0.1 on port 57101.
```

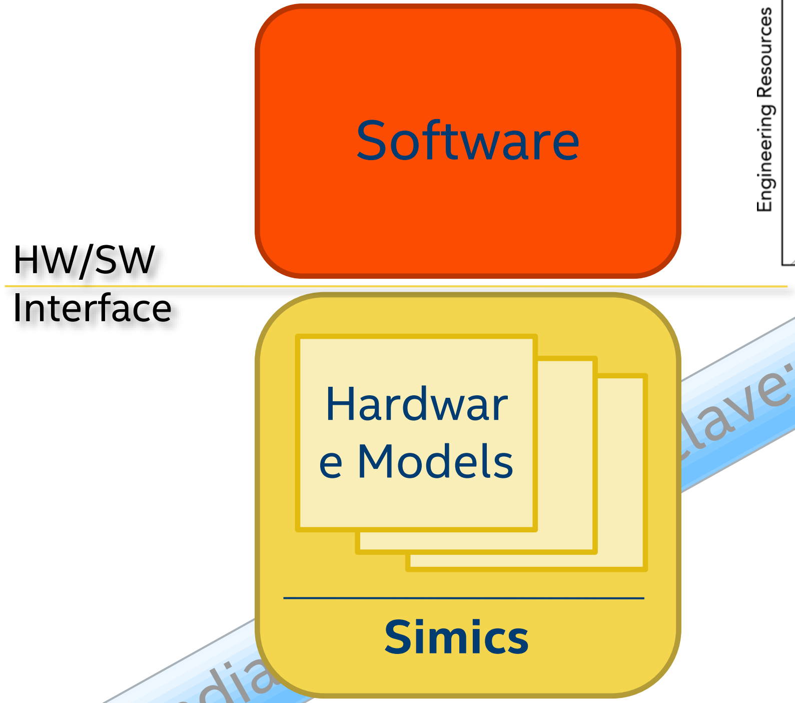
Name	Type	Value
val1	int	0
val2	int	11
s	int	11

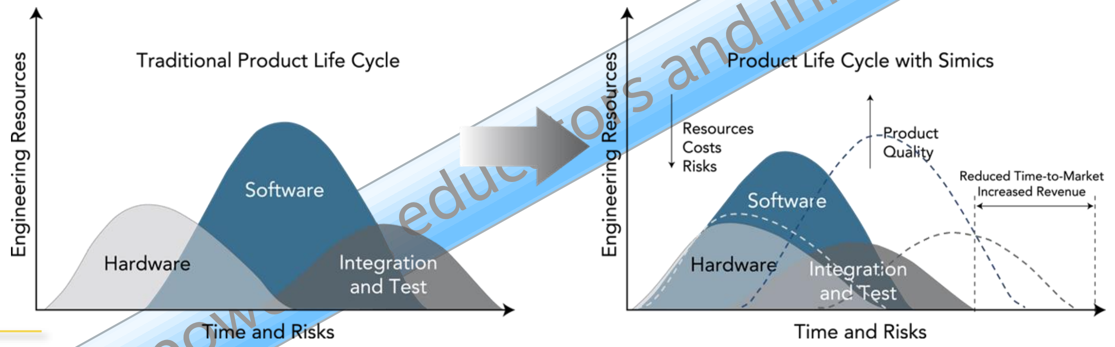
```
0x14800100
0x14800360
```

0x14800100 : 0x14800100 <Hex Integer>

Address	0 - 3	4 - 7	8 - B	C - F
14800100	580000C1	9100003F	D280001D	D280001E
14800110	9400000F	00000000	148011B0	00000000
14800120	D10083FF	B9000FE0	B9000BE1	B9001FFF
14800130	B9400FE1	B9400BE0	0B000020	B9001FE0
14800140	B9801FE0	910083FF	D65F03C0	A9BE7BFD
14800150	910003FD	F9000FBF	52800020	B90017A0
14800160	90000000	9106A000	F9400000	B9400000
14800170	B94017A1	97FFFFEB	B90013A0	B94013A0
14800180	7100001F	54000081	F9400FA0	91000400
14800190	F9000FA0	B94017A0	11000400	B90017A0
148001A0	17FFFFF0	00000000	14800008	00000000
148001B0	00000000	00000000	00000000	00000000

Modelling on Virtual Platforms and Simics





Fast, function-accurate, full-system virtual platform

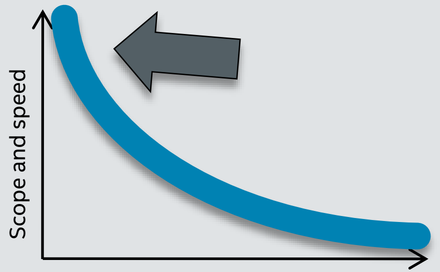
Simulate any computer-based system on a standard PC, server, or cloud server

Run the full software stack – firmware, boot code, OS, drivers, applications, middleware

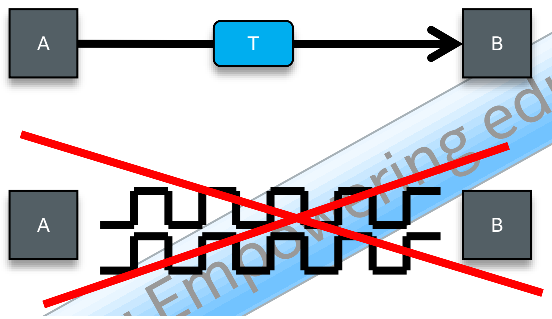
https://www.intel.com/content/www/us/en/developer/articles/technical/simics-simulator-technology.html

Simics Level of Abstraction

Goal: Fast & scalable simulation

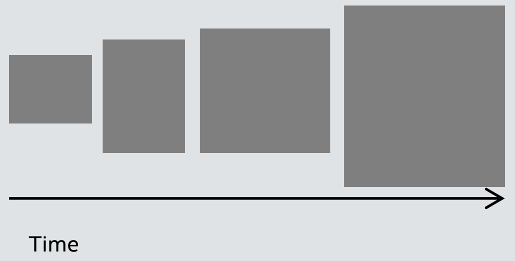


Transaction-level modeling (TLM)

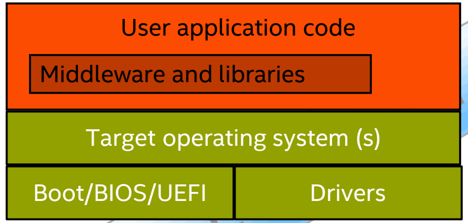


Lazy and agile modeling

Build out platform from core to all over time



Goal: run the real software



User application code

Middleware and libraries

Target operating system (s)

Boot/BIOS/UEFI | Drivers

Target model includes all software-visible functional aspects of hardware, such as processor instructions, supervisor modes, device registers, interrupts, etc.

Model function & basic timing

Processor instruction set	System memory map (not bus system)	Device register interface
Loose timing model	Packet-level models of networks	Event-driven simulation, not cycle-driven

Add timing and µarch when needed

Processor simulators from designers	Cycle-accurate hardware models	Cache model (timing)
Processor timing models	Power models	

Risc-V opportunities for students

- https://riscv.org/
- https://shakti.org.in/
- Risc-V simulators - QEMU
- Opportunities for students to work on Risc-V at Intel/PSG on
  - Design, Verification, Physical Design, FPGA emulation, Silicon validation …
- Short term internships – 6 months
- Long term internship – 1 year
- College graduate hiring

Summary

- Risc-V is open and is gaining fast and wide acceptance
    - With many vendors and many SoC products in the field
- Risc-V has limitless opportunities and potential for all types of computing.
- Intel is working on advanced products with Risc-V based CPUs with best-in-class PPA

Quiz

- What does V stand in Risc-V for ?
- How many registers in ISA ?
- What is r0's specialty ?