Joint ICTP-IAEA School on Detector Signal Processing and Machine Learning for Scientific Instrumentation and Reconfigurable Computing

The Open Standard RISC-V Architecture

Fernando Rincón
University of Castilla-La Mancha
fernando.rincon@uclm.es

The Abdus Salam International Centre for Theoretical Physics

IAEA International Atomic Energy Agency

unesco United Nations Educational, Scientific and Cultural Organization

UCLM

Contents

- What is RISC-V?
- Why RISC-V?
- RISC-V International
- RISC-V Modularity and ISA standards
- RISC-V Extensions
- RISC-V & AMD Xilinx: Microblaze V
- Beyond Microblaze V

What is RISC-V? (And What is it Not?)

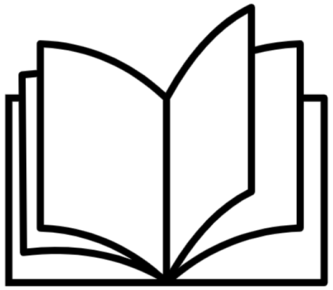
RISC-V is

- An Instruction Set Architecture (ISA)
- A Specification or Blueprint
- A Standard (like AXI)
- The language SW uses to talk to HW

RISC-V is not

- A processor
- An implementation or IP core
- A company (like ARM or Intel)
- A chip (like a cortex)

RISC-V standard



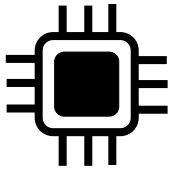
add x1, x2, x3

lw x4, 0(x5)

Memory Model

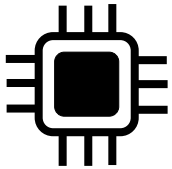
Extensions (M, C, F...)

Implemented as



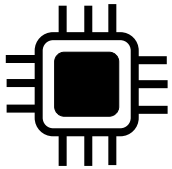
AMD/Xilinx Microblaze V

Small 5-stages RV32IMC



SiFive P550

High-performance 13-stages RV64G



Your own vhdl core

Custom 3-stage RV32I + custom insns.

What is an ISA?

- Contract between hardware (HW) and software (SW)

- Has profound implications:
  - 99% of laptops/desktop/servers based on AMD x86-64 ISA
    - CPUs built by Intel or AMD
  - 99% of mobile phones and tablets based on ARM v7/v8 ISA
  - Intel and ARM have tried to penetrate the other markets
  - Limited or no success. Can guess why?

The Old World (Proprietary ISAs)

x86 (Intel/AMD)

Complex (CISC)

High-performance

Licensed

ARM Cortex

Simpler (RISC)

scalable

Licensed (royalties, architecture fees)

MicroBlaze (Classic):

Microcontroller

Configurable soft-core

Licensed

The RISC-V Proposition

- A free, open-standard ISA.
  - Same kind of freedom as in free software
- Simple
  - Far smaller than most commercial ISAs
- Clean design
  - Clear separation between user and privileged ISA
  - No microarchitecture or technology-dependent features
- Modular
  - Small standard base ISA
  - Multiple standard (and optional) extensions
- Extensible and customizable by design
- Stable
  - Base and standard extensions are frozen

A Little History

- May 2010: The Beginning
  - A team at UC Berkeley starts a 3-month project to create a new ISA for their undergraduate and graduate courses.
  - Existing options (x86, Arm) were too complex and proprietary for teaching.
- 2010 - 2014: From Project to Movement
  - The "3-month project" turns into 4 years of intensive research, publications, and several chip tape-outs.
  - The team discovers a large following in industry and academia who are actively using and relying on the spec.
- August 2015: The Foundation is Born
  - Realizing the project was bigger than the university, the RISC-V Foundation was created to steward the open standard.
- the "V" refers to the 5th generation of RISC from UC Berkeley.

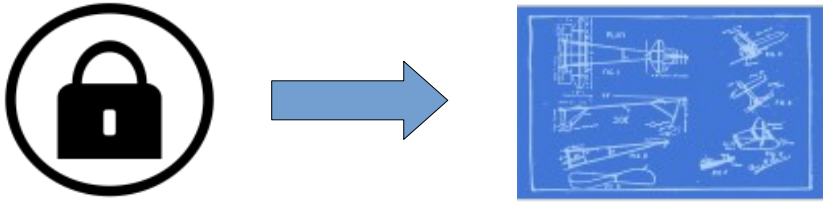
RISC-V International

- The non-profit organization established in 2015
- Based in Switzerland (neutrality).
- Role of the foundation:
  - Lead future development of ISA
  - Formal verification of the ISA
  - Community development
  - Ecosystem development

The Rise of RISC-V

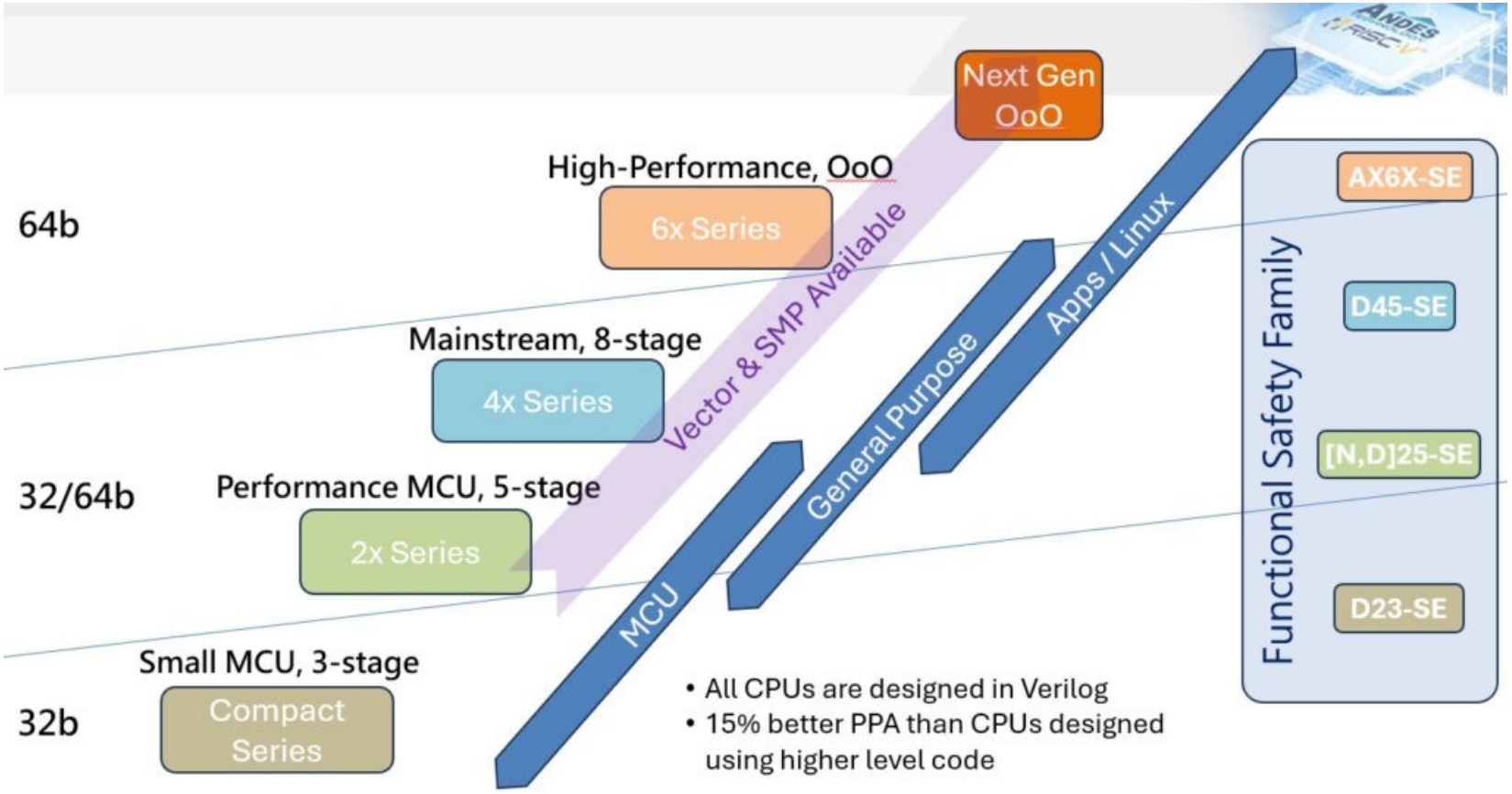
- Market & Geopolitical Drivers:
  - US monopoly on CPU technology (Intel, AMD, NVIDIA)
  - Seeking for technological sovereignty:
    - especially in the EU and China
    - The potential NVIDIA-Arm deal accelerates this.

- The Open-Source Advantage:
  - Democratizes CPU Design:
    - open-source nature
  - Fosters Innovation & Ecosystem:
    - Strong community (industry + academia)
    - rich software ecosystem (compilers, operating systems, tools)
  - Customization:
    - custom processors for specific workloads (AI, HPC, IoT, storage controllers).



From Micro to Macro

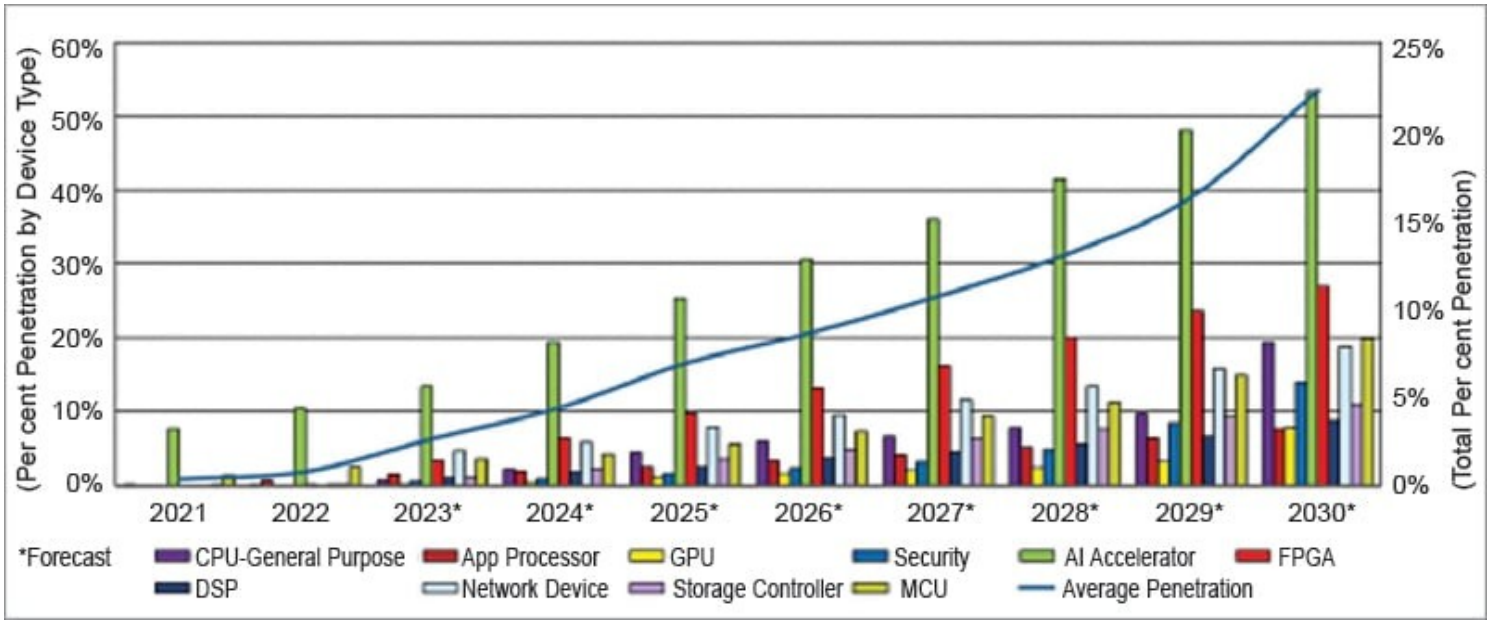
- RISC-V has been design to build from small microcontrollers to HPC multiprocessors
  - eg. The AndesCore RISC-V families



Source: Andes

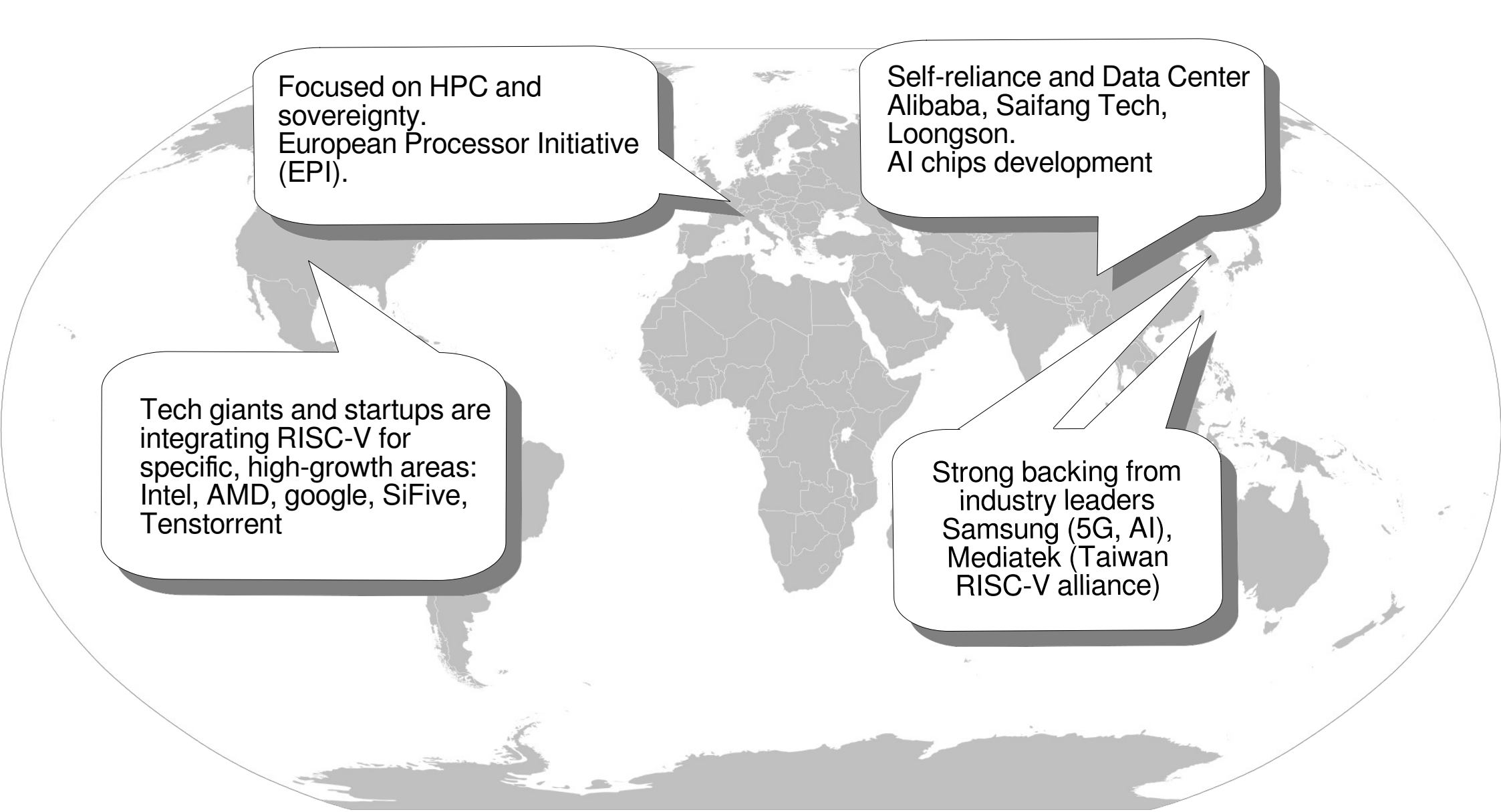
RISC-V Market Forecast

- Royalty revenues (paid per-unit-shipped) are forecast to surpass licensing fees (paid for access to the IP) around 2027.
  - Shift from design revenues to to shipping RISC-V based products
- Royalties expected from $16.7M in 2022 to $968.1M by 2030.
  - 60.2% Compound Annual Growth Rate (CAGR)



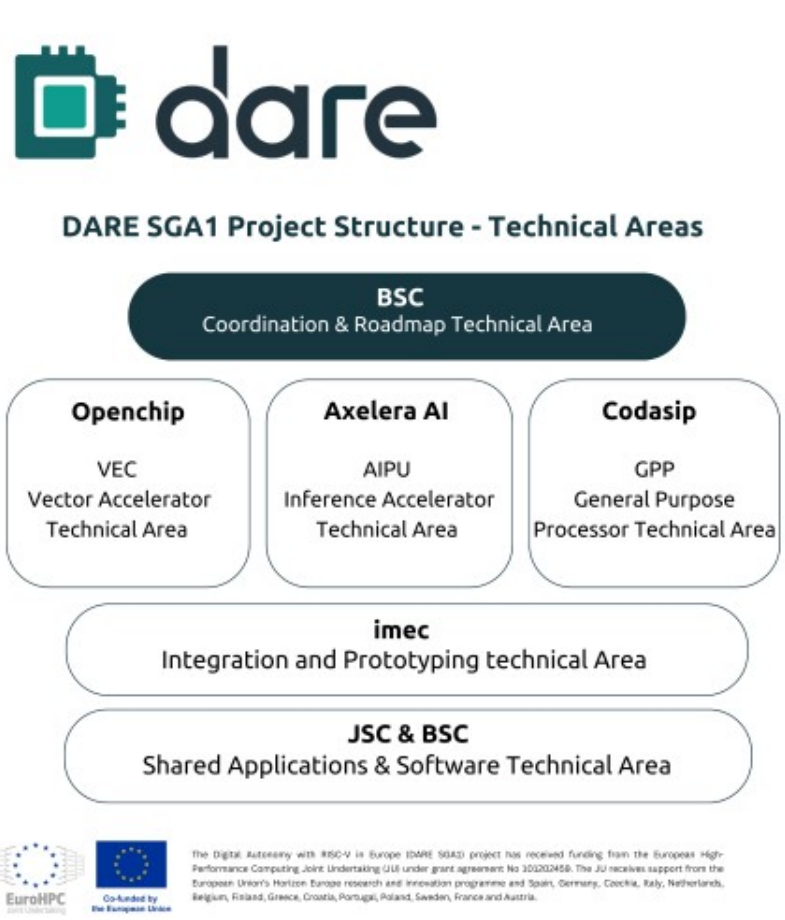
Source: The SHD Group, January 2024

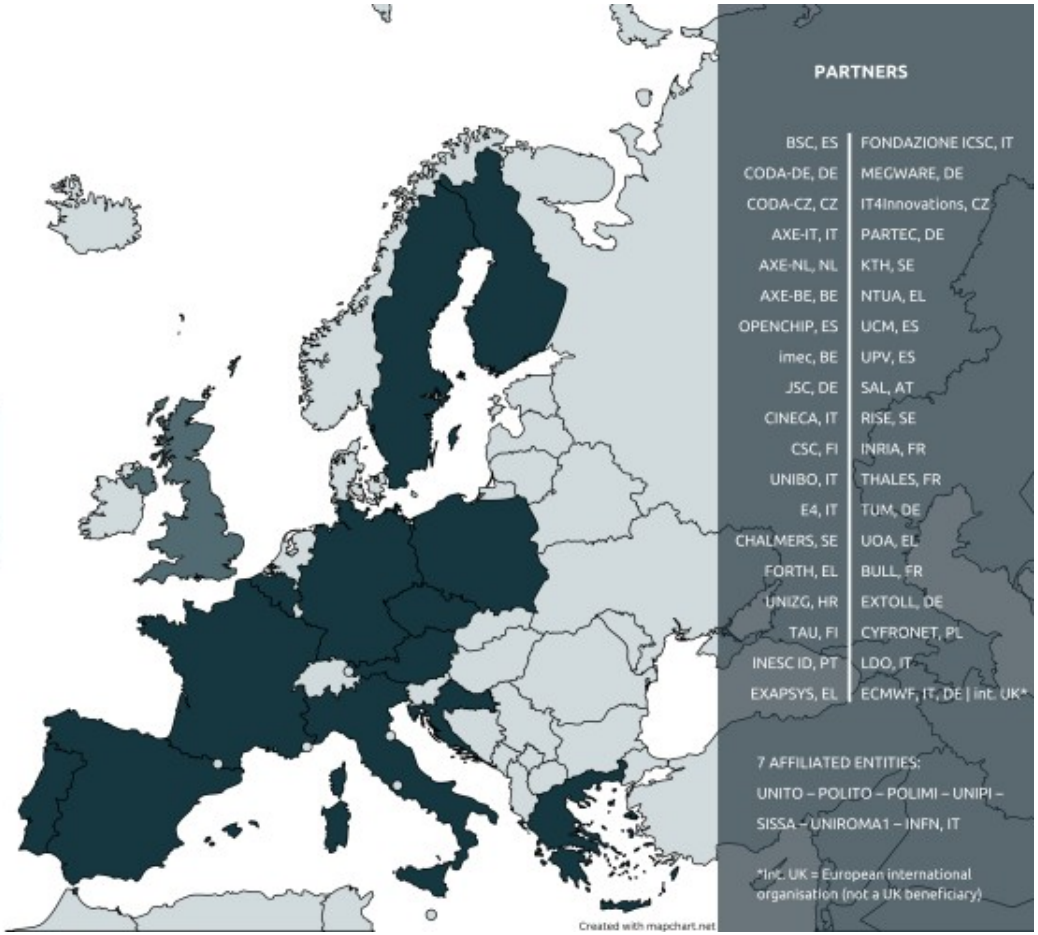
Global Adoption: Who is Building on RISC-V?



The European Processor Initiative

- develop cutting-edge HPC hardware and software based on RISC-V





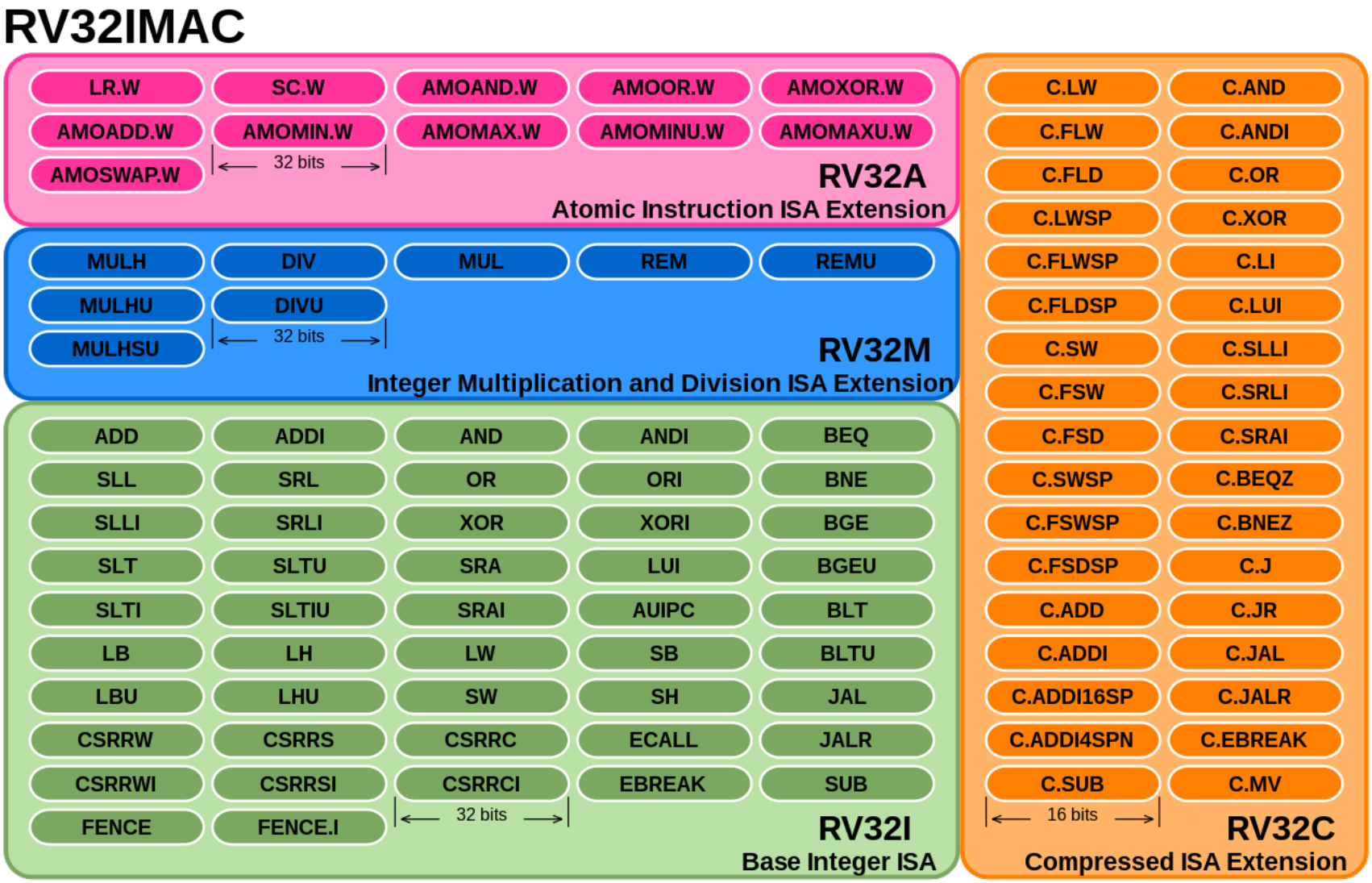
ISA

- Standard naming convention to describe the ISAs supported by a given implementation.
- ISA Name Format:
  - **RV[###][abc.....xyz] RV**
    - RISC-V architecture.
  - **[###]: {32, 64, 128}**
    - Width of the integer register file and the size of the user address space.
  - **[abc...xyz]**
    - set of extensions supported by an implementation.
- The Base ISAs:
  - **RV32I**: The 32-bit base integer instruction set. (The "I" is for Integer).
  - **RV64I**: The 64-bit base integer instruction set.
  - **RV32E**: A tiny (16-register) base for the smallest embedded cores.

Standard Extensions: G set

- **M**: Integer Multiply/Divide
- **A**: Atomics
  - For locking, semaphores, OS support
- **F**: Single-Precision Floating Point
- **D**: Double-Precision Floating Point
- **C**: Compressed Instructions
  - 16-bit instructions
- **G** = IMAFD:
  - The common "General-purpose" set

Standard Extensions



Other Extensions

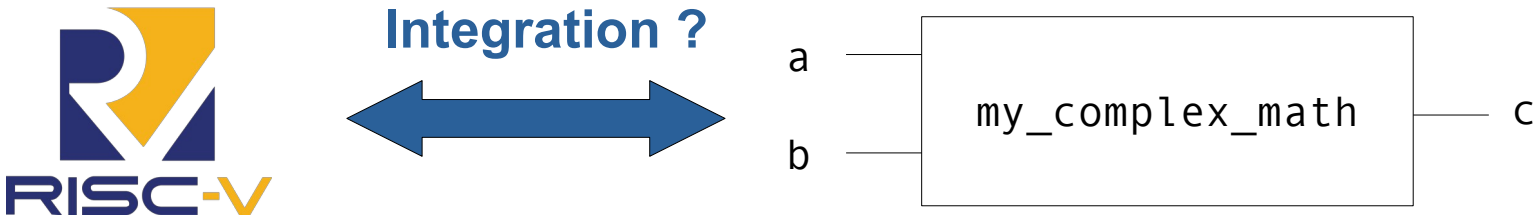
- **P**: Packed SIMD
- **V**: Vector Extension
  - for HPC, AI/ML
- **B**: Bit Manipulation
  - crypto, graphics
- **T**: Transactional memory
- The "Killer Feature": **Custom Extensions!**
  - You can add your own instructions to the (Z) extensions, and it's part of the spec!

RISC-V Core extensions: the problem

- Your code is slow

```
for (i = 0; i < 1_000_000; i++) {
  // This one line is 95% of the runtime!
  result = my_complex_math(a, b, c);
}
```

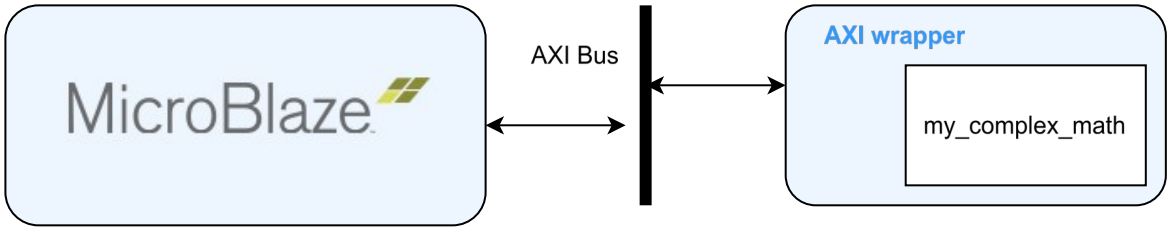
- But Hw implementation of the complex math function could take a pair of cycles



- 2 options:
  - External accelerator
  - Custom instruction: insert extra hw in the CPU pipeline

External acceleration

- Wrap your logic into a bus peripheral:
  - ej. AXI accelerator



HW flow

- Build logic
- Wrap as AXI IP core
- Connect to the AXI Bus

Sw flow

- CPU writes into mem-mapped address
  - Data and trigger operation
- CPU polls (or int) for completion
- CPU reads result from mem-mapped adress

External acceleration: pros & cons

Pros

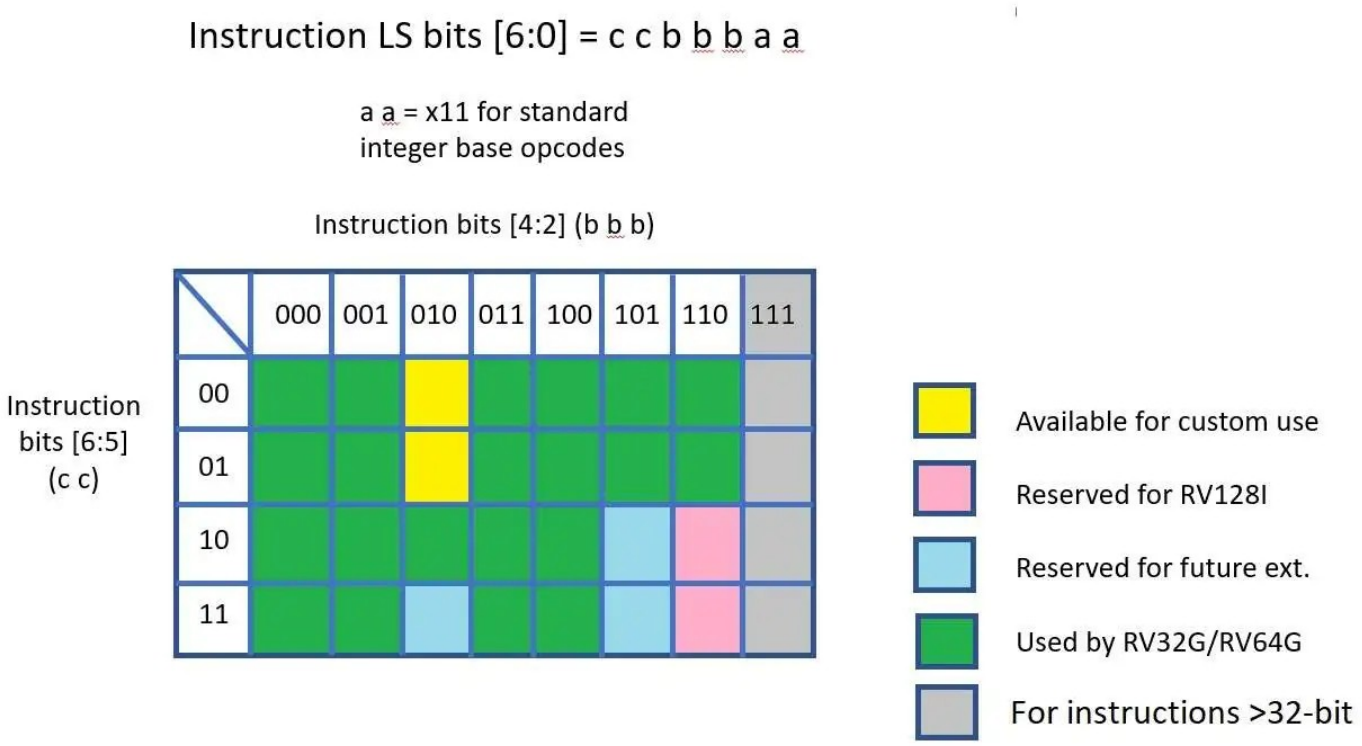
- Standard & Well-Supported: This is the default Vivado/Vitis flow. HLS generates this.
- Asynchronous: Your IP can run for a long time (e.g., process a whole video frame) while the CPU does other tasks.
- High Throughput: Perfect for large data. AXI-Stream and DMA are designed to move massive blocks of data efficiently.

Cons

- High Latency & Overhead: Every. Single. Transaction. is slow. You have bus arbitration, address decoding, and multiple clock cycles for one register read.
- Bad for "Micro-Tasks": If your task only takes 3 cycles, but the AXI bus transaction to start it takes 100 cycles, you've lost.
- Software Complexity: You must write (or generate) a driver. You have to manage base addresses, register offsets, and polling loops.

Custom Instructions

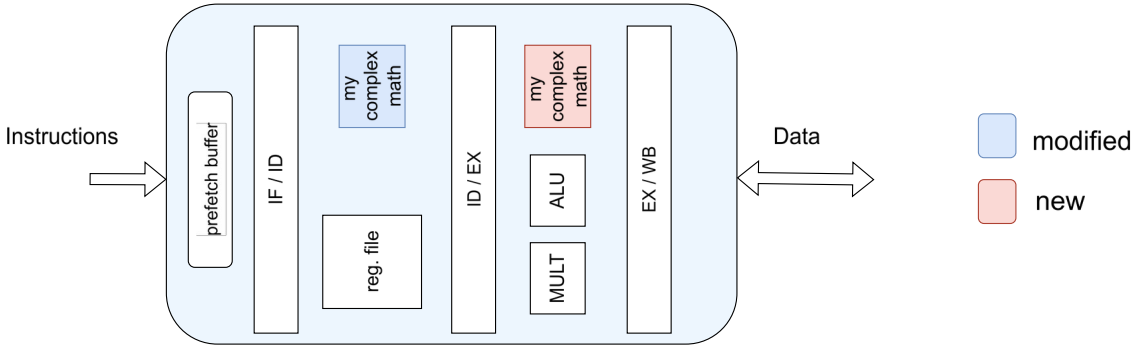
- RISC-V is architected to enable user-defined custom instructions
- 4 major opcodes reserved for use (out of 32)
  - And 2 more in the case of RV32C



Source: https://www.edn.com/creating-a-custom-processor-with-risc-v/

Custom Instructions

- Custom logic is part of the pipeline



HW flow

- Build logic
- Connect to reg. File and insert between stages
- Extend decoder

Sw flow

- CPU fetches a single instruction, e.g., MY_CUSTOM_OP x1, x2, x3.
- CPU pipeline sends values from registers x2 and x3 to my_complex_math unit.
- Computation is completed in 2 cycles.
- Result is written back to register x1, just like a normal ADD instruction.

Custom Instructions: pros & cons

Pros

- Ultra-Low Latency: As fast as the CPU's own ADD instruction (1-2 cycles).
- Zero Software Overhead: It's not a "driver." It's a single, atomic instruction.
- Perfect for "Micro-Tasks": Ideal for accelerating the "inner loop" of an algorithm (e.g., custom math, bit manipulation, crypto).

Cons

- Synchronous: The CPU stalls (waits) until your instruction is finished. Your logic must be fast (1-10 cycles).
- Tightly Coupled: You can't have a 10,000-cycle instruction; it would stall the whole processor.
- Limited Data: You are limited to the data in the CPU's registers (e.g., two 32-bit inputs, one 32-bit output). Not for streaming large data.
- HW Complexity: You are modifying the processor, not just connecting to a bus.
- SW Complexity: has an impact in the toolchain

Software Stack Implications

AXI Accelerator

- Implies writing a driver:
  - Using Xil_In32 and Xil_Out32, primitives
- Doesn't affect the compiler

my_driver.c

```
#include "my_driver.h"

MyAccel_SetData(...);
MyAccel_Start(...);
while(!MyAccel_IsDone(…));
result = MyAccel_GetData(...);
```

Custom Instruction

- Requires the extension of the compiler with the new instruction
- Usage through inline assembly macros

my_complex_math.h

```
#define my_complex_math(a, b) ({ \
int res; \
  asm volatile ("myop %0, %1, %2" \
  : "=r"(res) \
  : "r"(a), "r"(b)); \
   Res; \
})
```

my_program.c

```
#include “my_complex_math.h”

Result = my_complex_math(a, b);
```

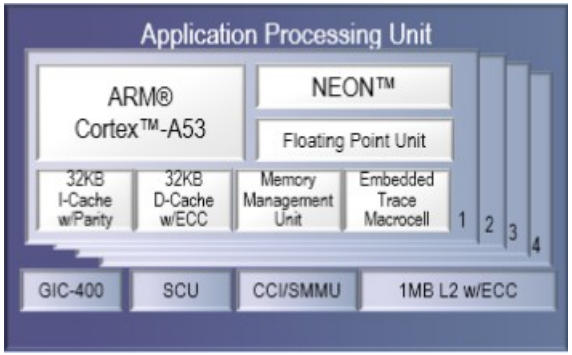
RISC-V Core extensions: Which One To Use?

- Use accelerators when:
  - Dealing with large blocks of data (e.g., an image, a large vector).
    - And move data through DMAs, not through CPU buses
  - The task takes a long time (e.g., an FFT, a video encoder).
  - You want the CPU to do other work while the accelerator runs.
- Use a Custom Instruction when:
  - The task is small and very repetitive
  - The task operates on register-sized data (e.g., a*b+c)
  - You need the absolute lowest possible latency

The Processor Spectrum on Xilinx

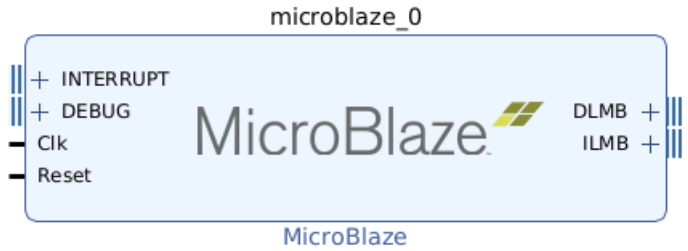
Hard-core processors

- Physical Hw core on the die
- Pros
  - Max. performance and power efficiency
  - Full-featured: MMUs, caches, complex pipelines
  - No use of FPGA resources
- Cons
  - Fixed and not configurable neither extensible



Soft-core procesor (Microblaze)

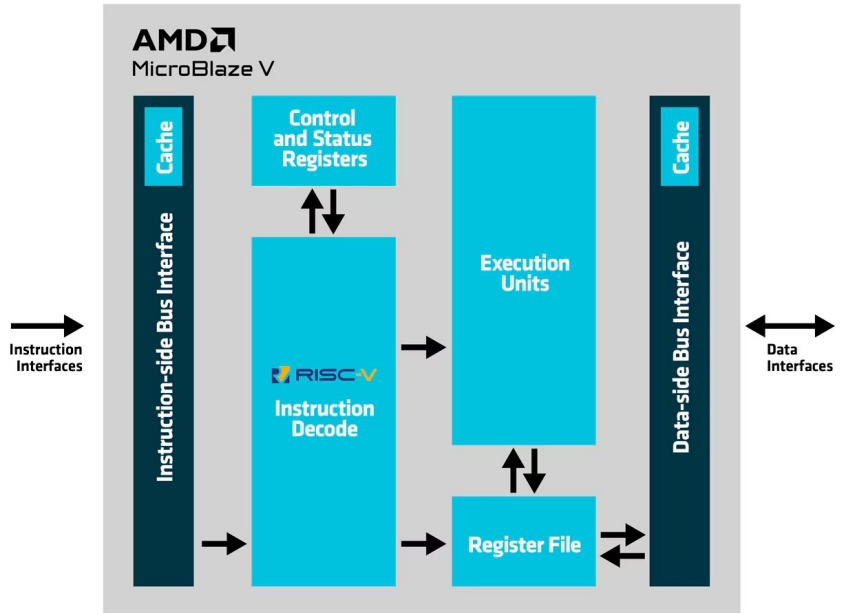
- Built on FPGA resources
- Pros
  - Flexible and Scalable
  - Well integrated in the flow
- Cons
  - Worse performance
  - Consumes FPGA resources
  - Proprietary



From Microblaze to Microblaze V

- Xilinx (AMD) became a member of RISC-V International in 2020
- Microblaze has been superseeded by Microblaze V
- Total redesign
- 32-bit soft-core fully compatible with RISC-V ISA
- No royalties
- Highly flexible
- Smaller for equivalent configs.

MicroBlaze V Overview Illustration



Microblaze V at a glance

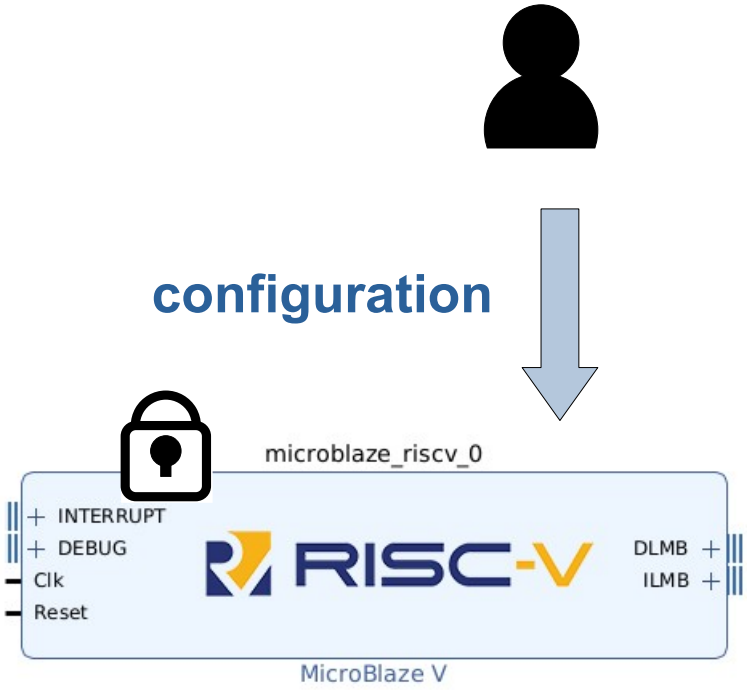
- ISA: RV32IMC
  - RV32I: The base 32-bit Integer instruction set.
  - M: Includes the Multiply/Divide extension
    - synthesis tools will infer DSPs for this!
  - C: Includes the Compressed instruction extension.
- Why is the 'C' Extension so important for FPGAs?
  - It mixes 16-bit and 32-bit instructions.
  - This makes your program's binary ~30% smaller.
    - ~30% less Block RAM (BRAM) is needed to store the program
- Pipeline: 5-stage, in-order (The classic RISC pipeline).
- AXI4 Interfaces
  - seamless integration with the Xilinx IP ecosystem
- Same exact Design flow and tool support:
  - You won’t notice the difference

Microblaze vs Microblaze V

Feature	Microblaze	Microblaze V
ISA	Propietary Xilinx	RISC-V (RV32IMC)
Ecosystem	Xilinx-only	Global RISC-V (gcc, etc)
Resources	baseline	~30% smaller (fewer LUTs)
Performance	baseline	"~1.4x Faster (CoreMark)
OS Support	FreeRTOS, Linux (w/ MMU)	FreeRTOS (No MMU yet)
Tool Flow	Vitis (GCC)	Vitis (GCC)
Debug	Vitis Debugger (GDB)	Vitis Debugger (GDB)

Microblaze V extensions

- Microblaze V is configurable, but not modifiable
- Delivered as a encrypted pre-compiled IP
- No access to code and pipeline
- Configuration
  - Enable/disable Instruction and Data Caches
  - Enable/disable the M (Multiply/Divide) extension
  - Enable the Interrupt Controller
  - Configure Debug levels
- No true Custom Instructions
- Bus-based approach, but 2 alternatives
  - AXI accelerator
  - AXI stream coprocessor



Microblaze V extensions

AXI accelerator

- Standard memory-mapped peripheral
- Sw flow:
  - Standard MMIO

```
Xil_Out32(BASE_ADDR, data);
```

  -
```
while(Xil_In32(STATUS_ADDR) != DONE);
```

- Loosely-coupled from CPU
  - Asynchronous execution
  - CPU triggers and waits for interrupt

AXI stream coprocessor

- Dedicated AXIS ports on the core
  - stream links
- Sw flow:
  - Special builtin coprocessor instructions

```
putf(my_coproc_id, data);
```

```
getf(my_coproc_id, &result);
```

- Tight-coupled with CPU
  - Synchronous communication
    - Putf / getf may stall the pipeline
  - Asynchronous execution

Beyond Microblaze V

- MicroBlaze V is the easy, supported path
- But with RISC-V, you are not limited to it
- There are numerous open-source alteratives
  - Look at https://github.com/riscv/learn?tab=readme-ov-file#open-risc-v-implementations

Open RISC-V Implementations

Explore open RISC-V implementations for hands-on learning.

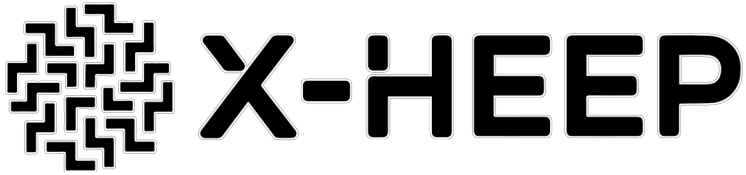
Name	Description	Access	Date Added
AUK-V-Aethia	AUK-V RV32I CPU.	Github	2024-18-10
CVA6	The CORE-V CVA6 is an Application class 6-stage RISC-V CPU capable of booting Linux	Github	2024-18-10
CV32E40P	CV32E40P is an in-order 4-stage RISC-V RV32IMFCXpulp CPU based on RI5CY from PULP-Platform.	Github	2024-18-10
DarkRISCV	Small RV32-E / I soft-core CPU optimized for FPGAs.	GitHub	2024-18-10
Grande Risco-5	RISC-V RV32I multi-cycle processor with a 5-stage pipeline, designed for educational purposes.	Github	2024-06-11
Hazard3	3-stage RV32IMACZb* processor with debug	Github	2024-19-12

27 projects

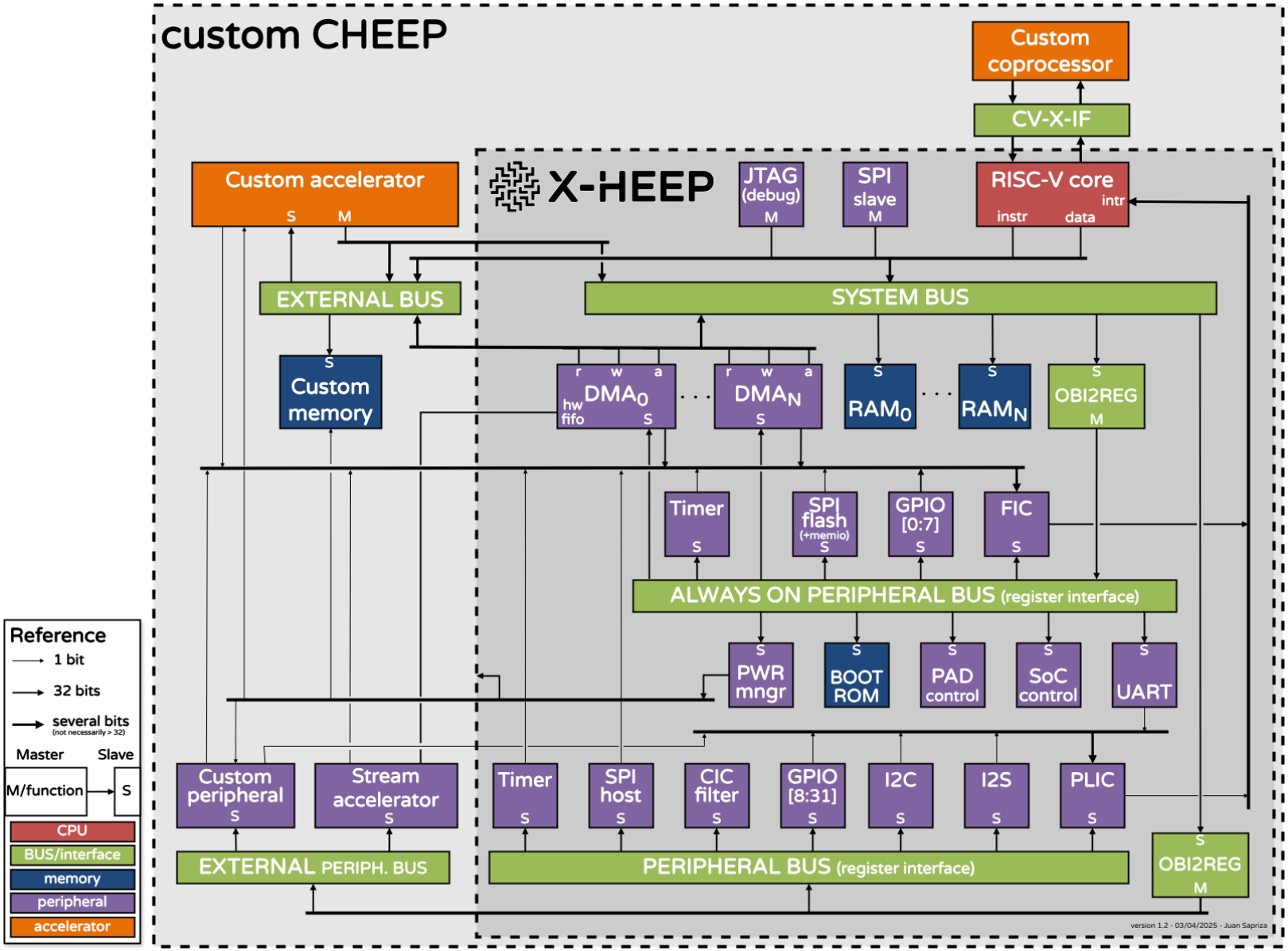
Currently listed

X-HEEP Project

- Stands for e**X**tensible **H**ardware **E**ngineering **E**ducation **P**latform
- Developed by EPFL (https://github.com/esl-epfl/x-heep)
- **Goal**:
  - design, verification, and deployment of RISC-V-based Systems-on-Chip
- provides a modular, pre-verified framework for building custom microcontrollers
- **Characteristics**:
  - Modular Architecture: Easy to add/remove peripherals and accelerators
  - Configurable: Choose your processor, memory, and bus structure
  - Open-Source: All hardware (Verilog/VHDL) and software are freely available
  - FPGA-First: Designed with FPGA implementation in mind
  - Some ASIC ports



X-HEEP Architecture



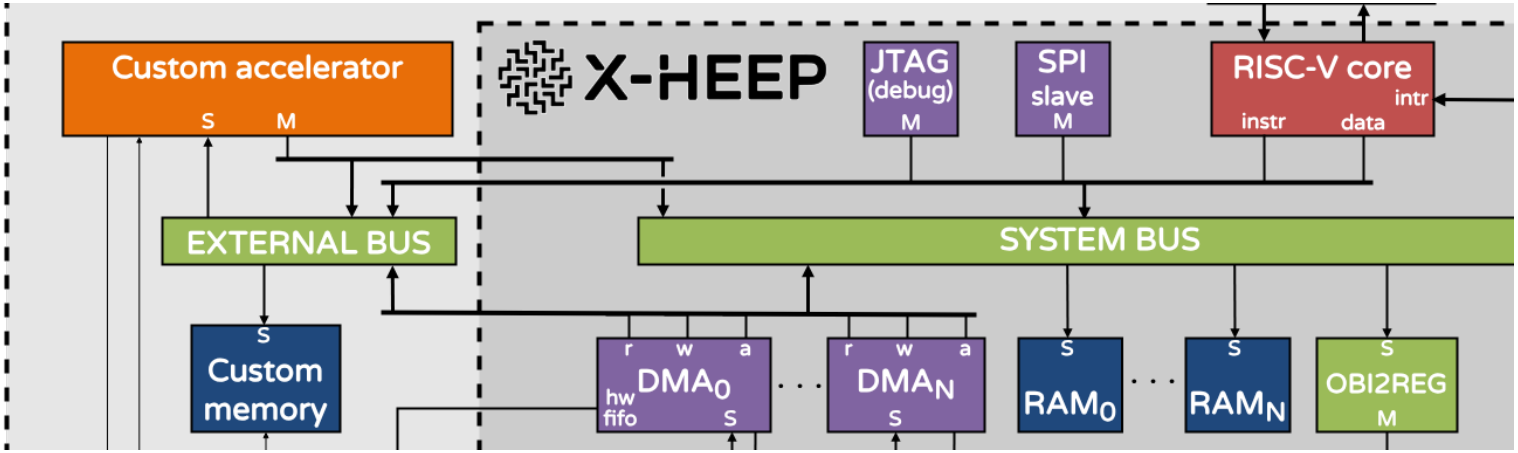
X-HEEP Base Processors & SoC Generation

- User choice
- By default based on the RISC-V OpenHW Group CPUs:
  - CVE2
  - CV32E40P / CV32E40PX
  - CV32E40X CPU

- SoC:
  - generator-based approach (Python scripts + fusesoc utils)
  - Configuration File: SoC definition in a simple YAML text file
  - Generation: The script reads your configuration and automatically:
    - Instantiates chosen RISC-V core.
    - Adds standard peripherals (UART, SPI, GPIO, Timers).
    - Configures memory controllers (SRAM, DDR).
    - Generates the top-level Verilog/VHDL for the entire SoC.
  - This output is then synthesizable directly for FPGA.

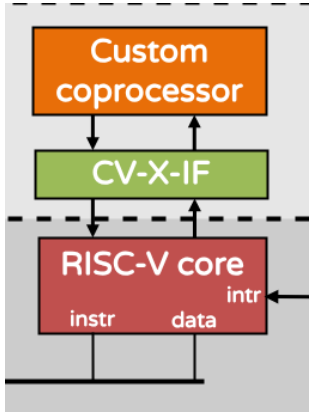
X-HEEP Extension

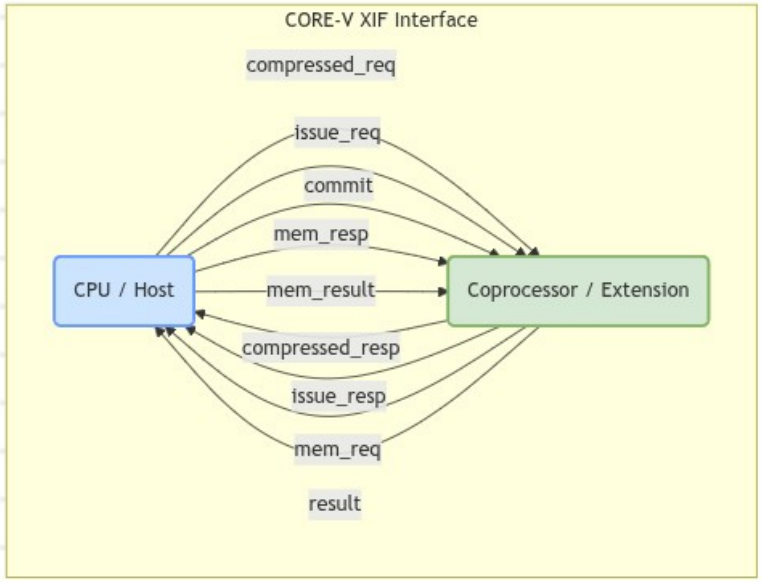
- Through external accelerators
- Similar to the MicroBlaze V's AXI peripherals.
- X-HEEP typically uses an AXI or Wishbone bus as its primary system interconnect.



X-HEEP Extension

- Through custom instructions (coprocessor)
- Based on the extension interface CV-X-IF
  - Available on cv32e40px or cv32e40x RISC-V CPUs
- CV-X-IF
  - Tightly-Coupled
  - The CPU forwards instructions not recognized
  - Streamlined Handshake:
    - Interface defined by signal groups that manage the entire "life" of a custom instruction
  - Internal pipeline stages similar to the CPU





RISC-Vfpga Course

- Comprehensive, open-source training course developed by **Imagination Technologies** in collaboration with RISC-V International.
- **Designed to teach** RISC-V ISA concepts, system-on-chip (SoC) design, and embedded programming using a real RISC-V core on a real FPGA board.
- F**PGA-Centric**:
  - Based on the Digilent Nexys A7 FPGA Board (Xilinx Artix-7 device)
- **Practical & Hands-On**:

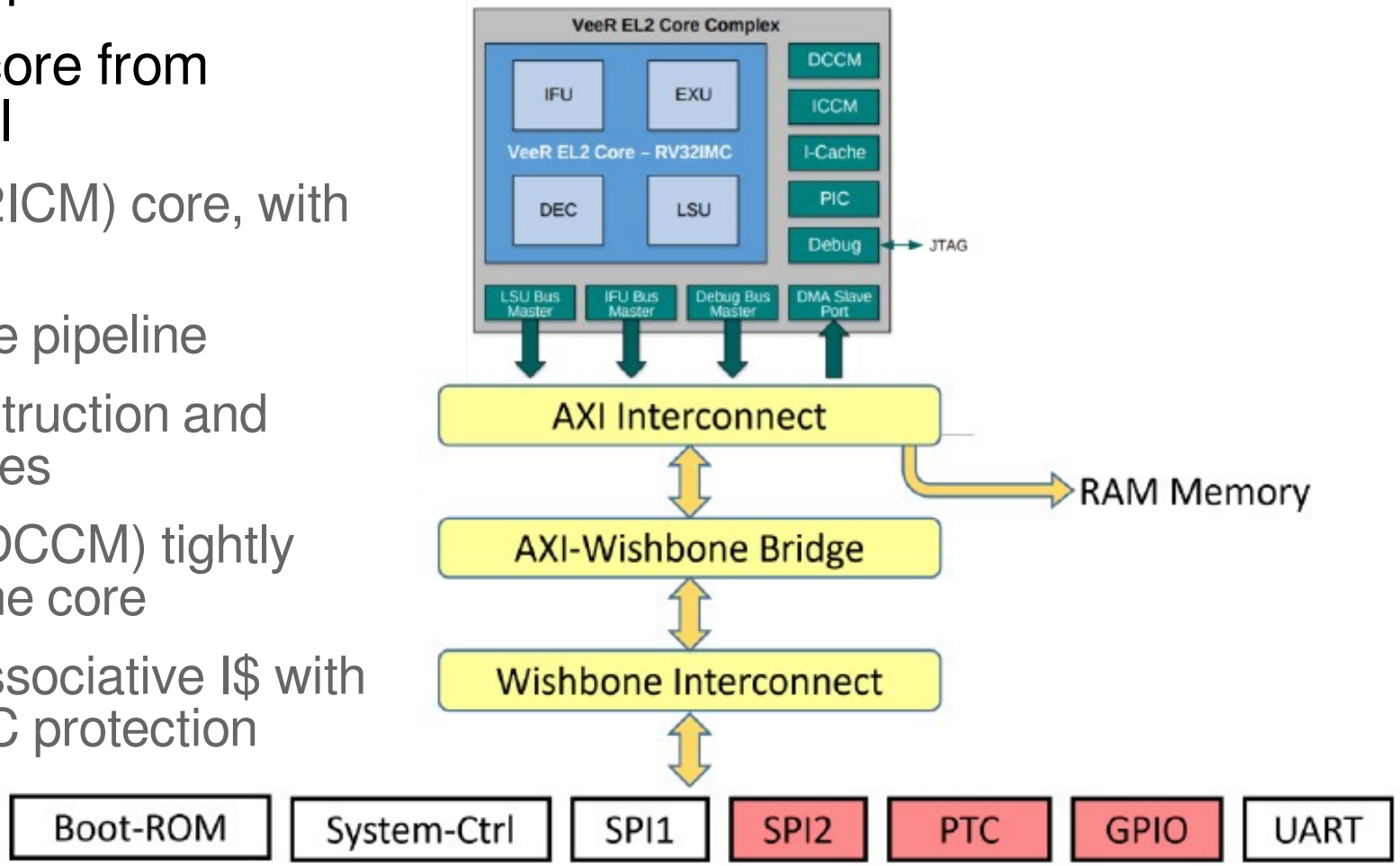
  It's not just theory; you'll be building, running, and debugging RISC-V code on hardware.
- **Open-Source Core**: The course utilizes the SweRV Core EH1, an open-source RISC-V core, allowing you to explore its Verilog source code.
- **System-Level Design**:
  - Integration of RISC-V core with peripherals (UART, GPIO, timers) to build a SoC.

RISC-Vfpga Course

- RVfpga: A first course about the RISC-V core, memory system, and peripherals.
- RVfpga-SoC: a second course that shows how to:
  - Build a RISC-V SoC from building blocks
  - Install the Zephyr RTOS (real-time operating system)
  - Run programs on Zephyr
  - Run simple Tensorflow programs
- Both courses (RVfpga and RVfpga-SoC) are available as separate downloads (free upon registration) at:
  - https://university.imgtec.com/rvfpga/

RISC-Vfpga Soc

- Open-source system-on-chip (SoC) from Chips Alliance
- Open-source core from Western Digital
  - 32-bit (RV32ICM) core, with single-
  - issue 4-stage pipeline
  - Separate instruction and data memories
  - (ICCM and DCCM) tightly coupled to the core
  - 4-way set-associative I$ with parity or ECC protection



References

- RISC-V foundation
  - Main site: https://riscv.org
  - Resources: https://riscv.org/members/resources/
  - training and certification: https://riscv.org/community/training/
- “RISC V ISA & Foundation Overview”. Youtube, uploaded by RISC-V International, May 20th 2018.
- “RISC-V Market Trends and Preddictions Till 2030”. Electronics for you, [June 7, 2024], www.electronicsforyou.biz/eb-specials/risc-v-market-trends-predictions-till-2030/
- "The World Is Getting Riskier." Omdia, [Day Month Year], omdia.tech.informa.com/om019360/the-world-is-getting-riscier.
- “MicroBlaze V Processor Reference Guide (UG1629)”, AMD Xilinx, 2025.
- X-Heep: https://github.com/esl-epfl/x-heephttps://github.com/esl-epfl/x-heep
- Rvfpga: https://university.imgtec.com/rvfpga/