国立大学法人
電気通信大学
The University of Electro-Communications

Synopsys & SCDC
Training Of Trainers
29/12/2022

VLSI Lab @ UEC: RISC-V Research

Trong-Thuc HOANG and Cong-Kha PHAM

OUTLINE

1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

OUTLINE

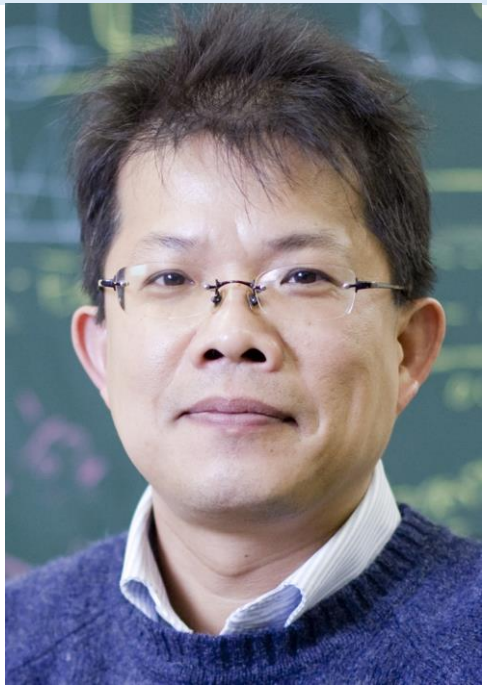
1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

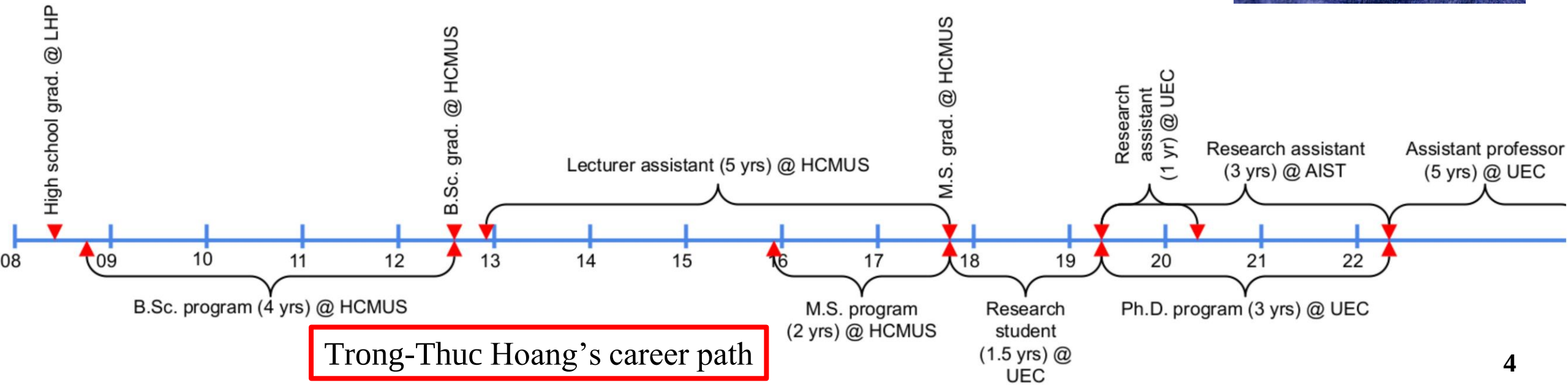
1. Introduction (1/4)

Trong-Thuc Hoang
Assistant Professor (UEC)
hoangtt@uec.ac.jp

https://thuchoang90.github.io/

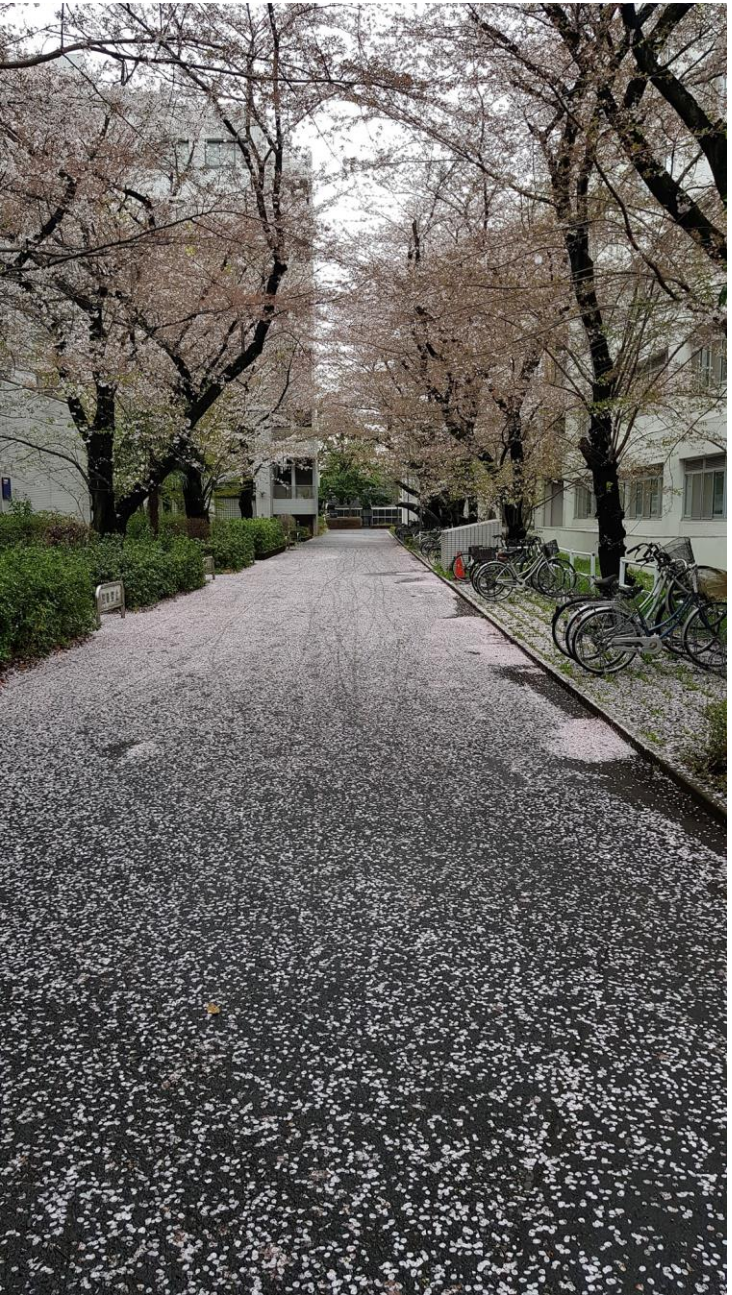
Cong-Kha Pham
Professor (UEC)
phamck@uec.ac.jp





Trong-Thuc Hoang's career path

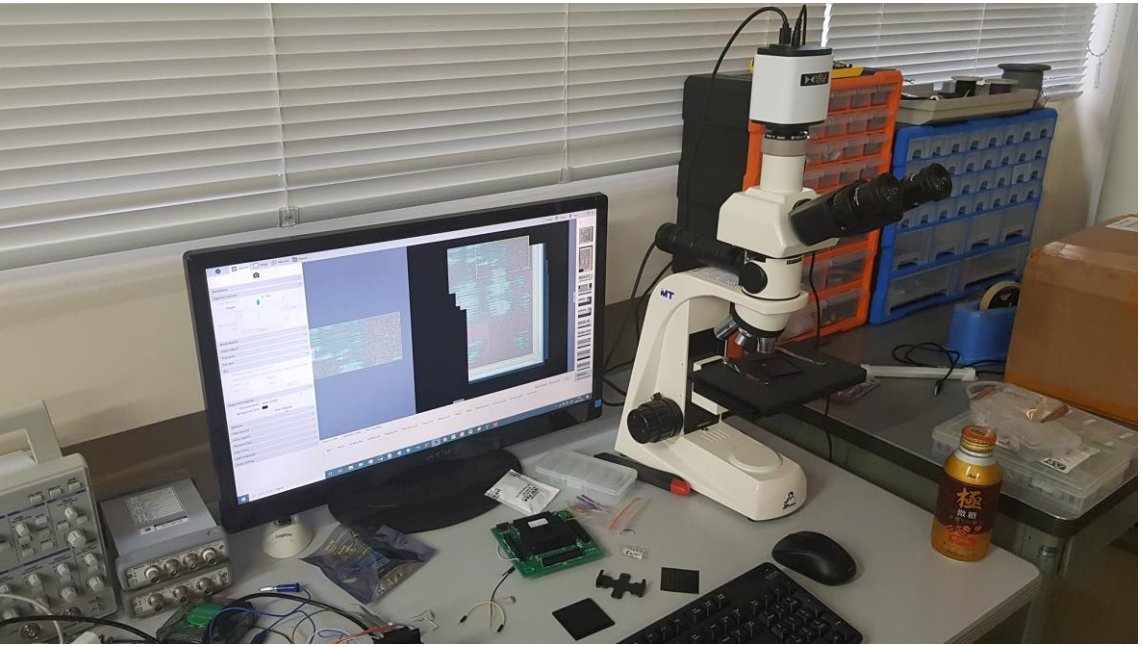
1. Introduction (2/4) University



国立大学法人
電気通信大学
The University of Electro-Communications

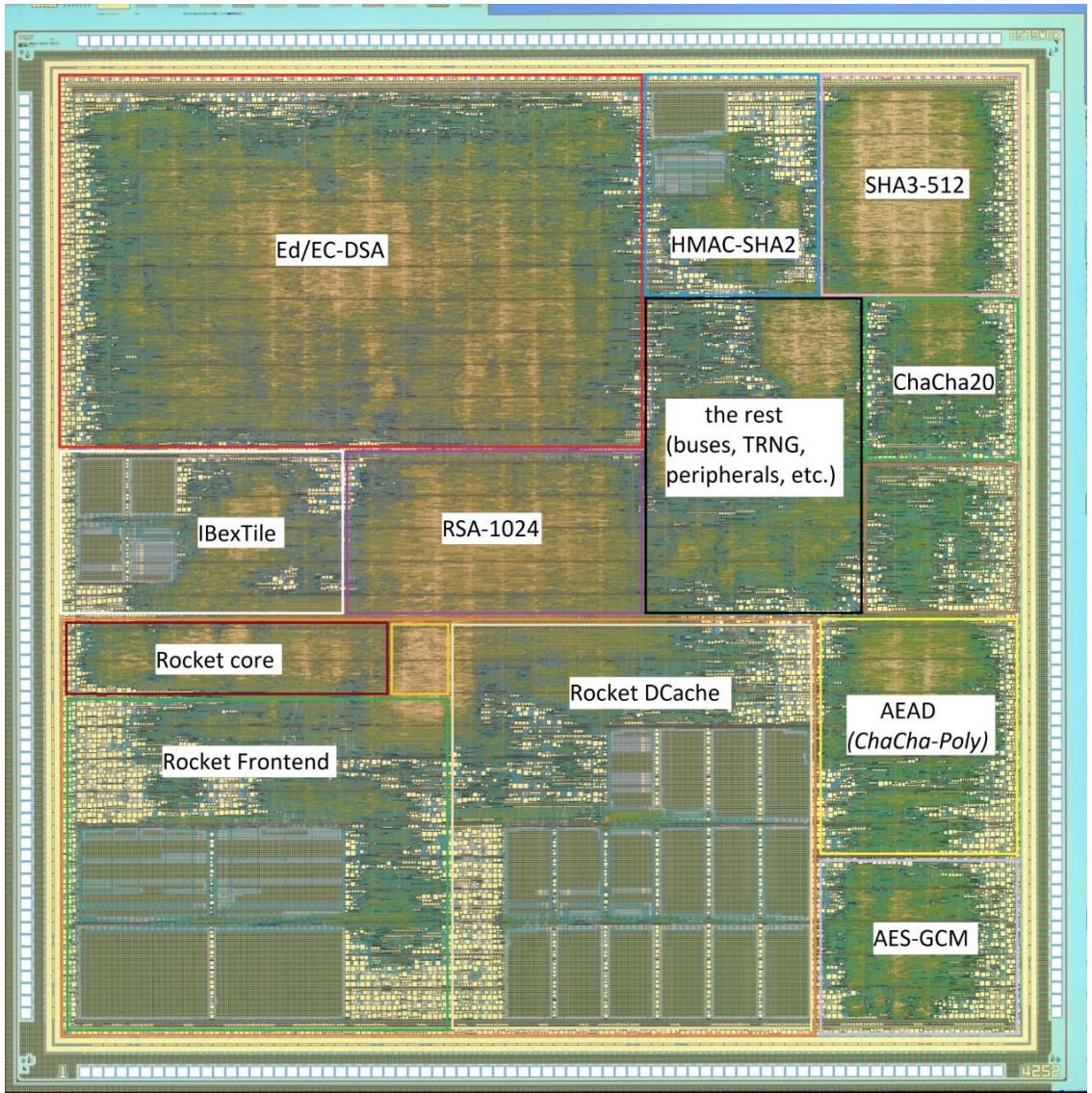
https://www.uec.ac.jp/

1. Introduction (3/4) Laboratory



http://vlsilab.ee.uec.ac.jp/

Pham Laboratory
Integrated circuit design laboratory



1. Introduction (4/4) Member

Member

(as in 2022)

- Ph.D. students: 4
- Master students: 6
- Bachelor students: 3
- Researchers: 3

Total: 16

From

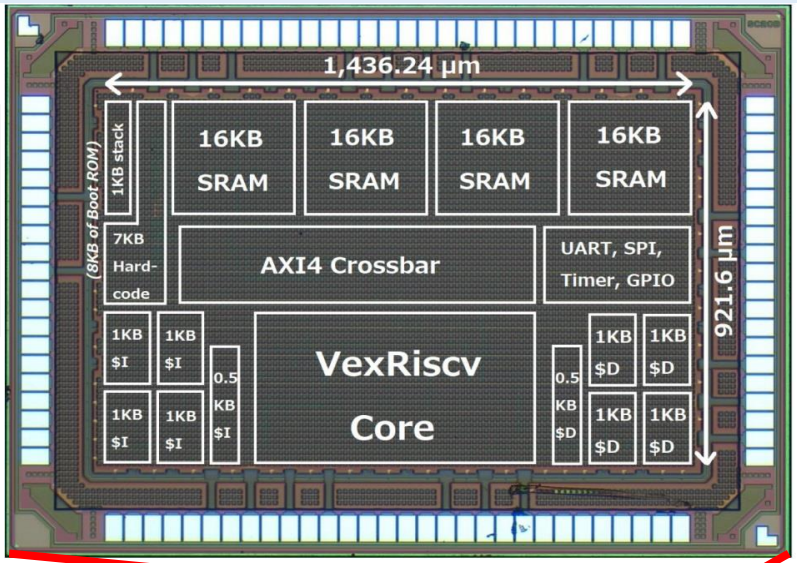
- Japan: 8
- Vietnam: 6
- Columbia: 2

OUTLINE

1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

2. Achievement highlight (1/8) Crypto-SoC timeline

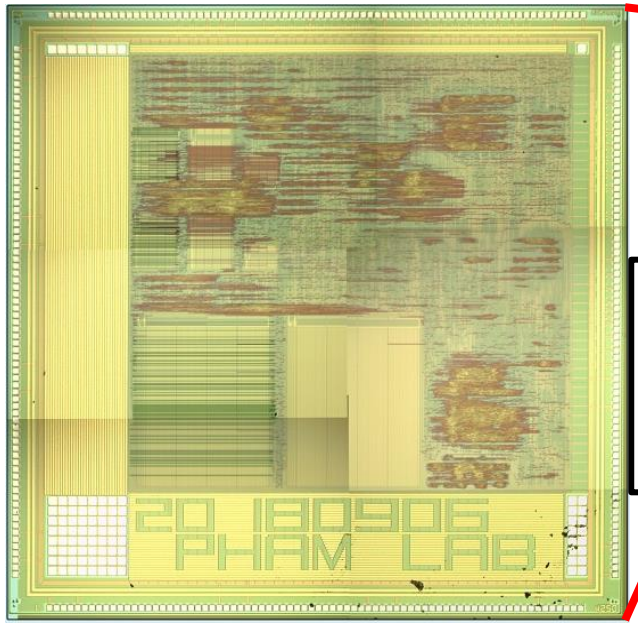
Crypto-SoC



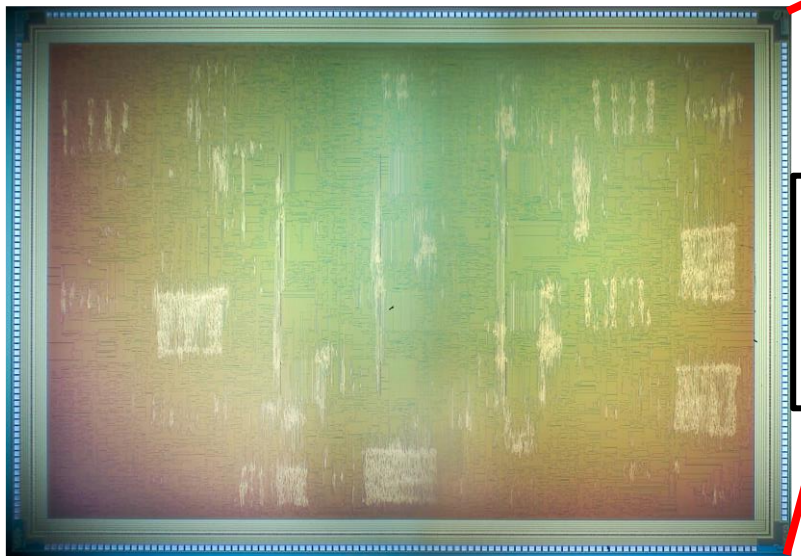
VexRiscv32(×1)
SOTB-65nm
2.0×1.5-mm2

2018 **2019** **2020**

09 **08** **10**



Rocket64(×1)
ROHM-180nm
5.0×5.0-mm2

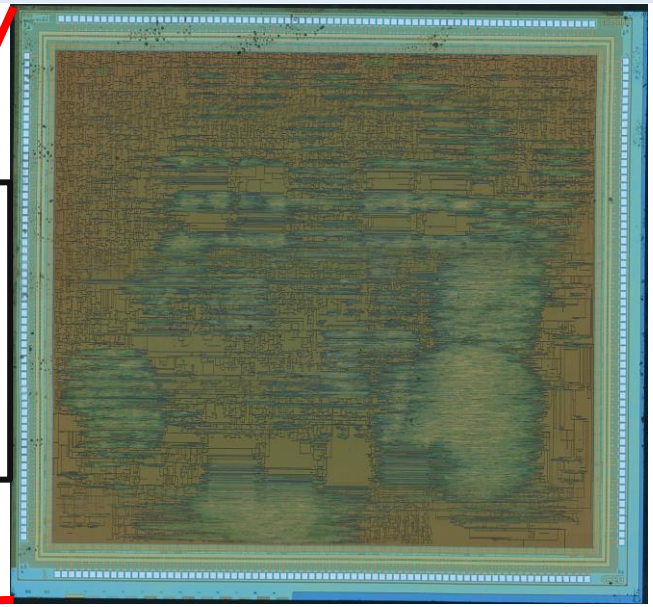


Rocket64(×4)
ROHM-180nm
5.0×7.5-mm2

2. Achievement highlight (2/8) Crypto-SoC timeline

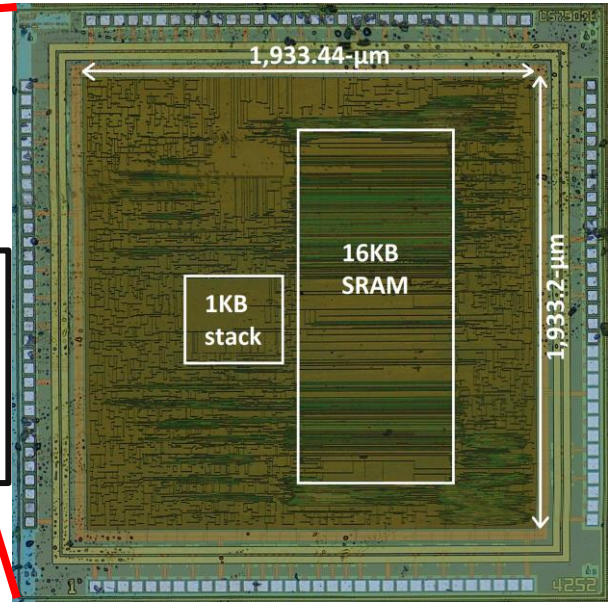
Crypto-SoC

Rocket64(×2)
+ Crypto-cores
ROHM-180nm
5.0×5.0-mm2



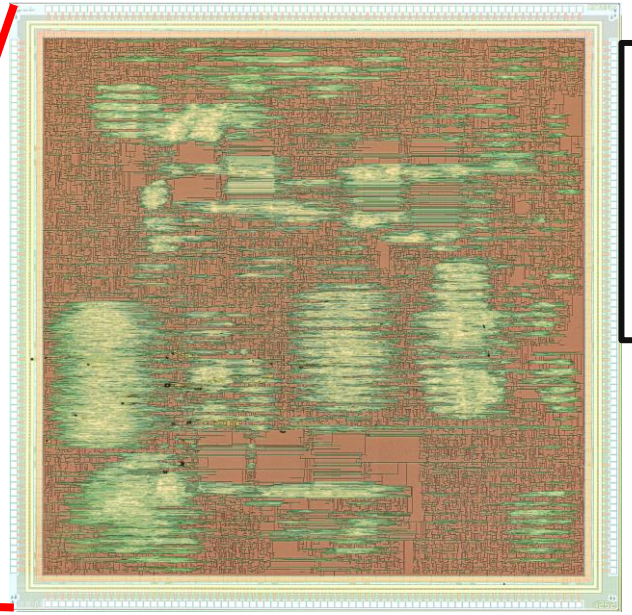
2020

01



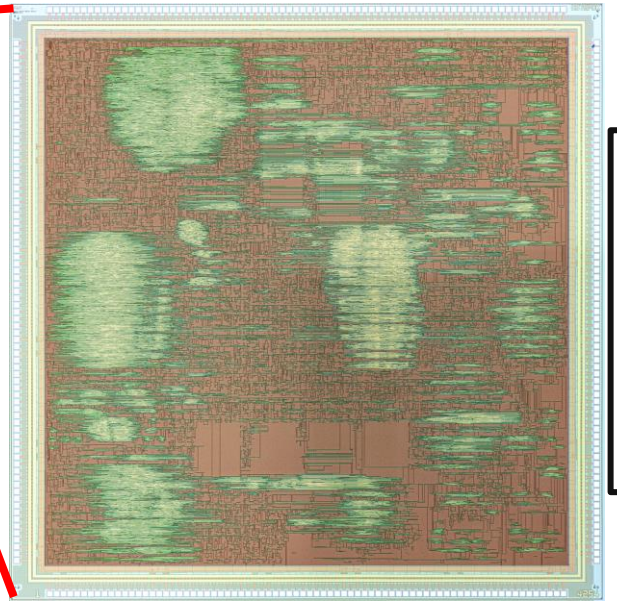
VexRiscv32(×1)
ROHM-180nm
2.5×2.5-mm2

06



Boom64(×1)
+ Crypto-cores
ROHM-180nm
5.0×5.0-mm2

2021

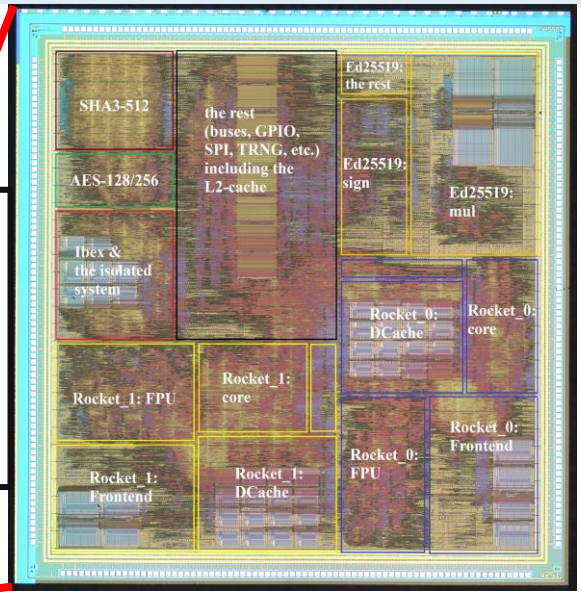


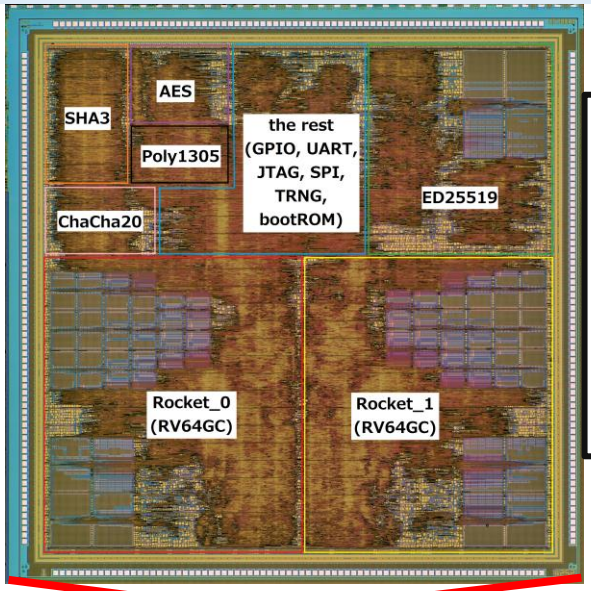
Rocket32(×1)
+ Boom32(×1)
+ Crypto-cores
ROHM-180nm
5.0×5.0-mm2

2. Achievement highlight (3/8) Crypto-SoC timeline

Crypto-SoC

Rocket64(×2)
+ Crypto-cores+TRNG
+ Secure boot
ROHM-180nm, 5.0×5.0-mm2





Rocket64(×2)
+ Crypto-cores
+ Rnd Freq
ROHM-180nm
5.0×5.0-mm2

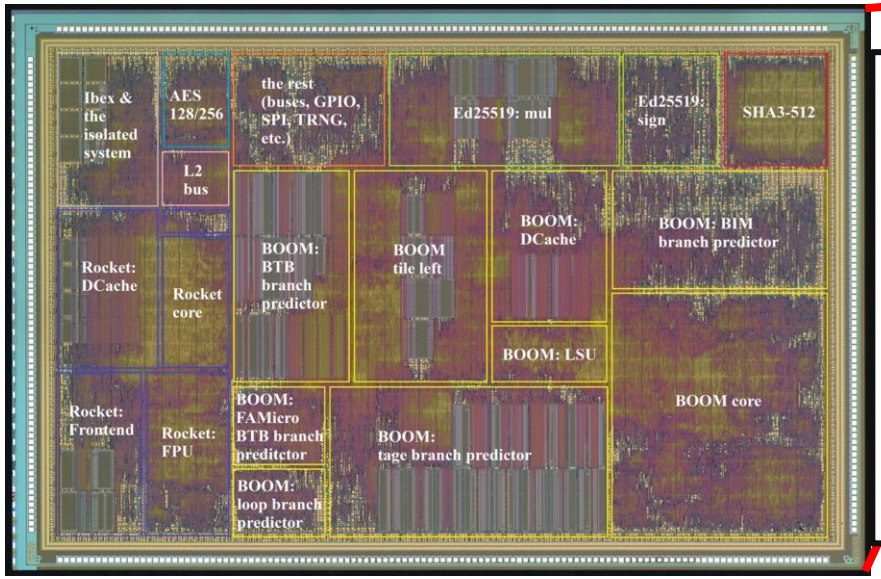
2021

2022

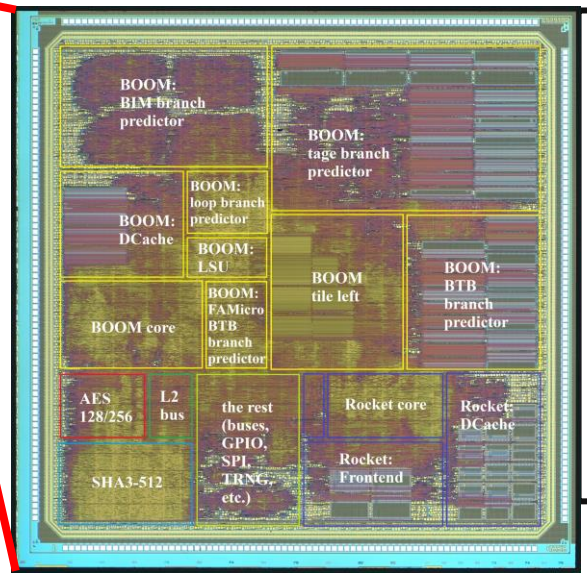
02

06

09



Rocket64(×1)
+ Boom64(×1)
+ Crypto-cores
+ TRNG
+ Secure boot
ROHM-180nm
5.0×7.5-mm2



Rocket32(×1)
+ Boom32(×1)
+ Crypto-cores
+ TRNG
+ Secure boot
ROHM-180nm
5.0×5.0-mm2

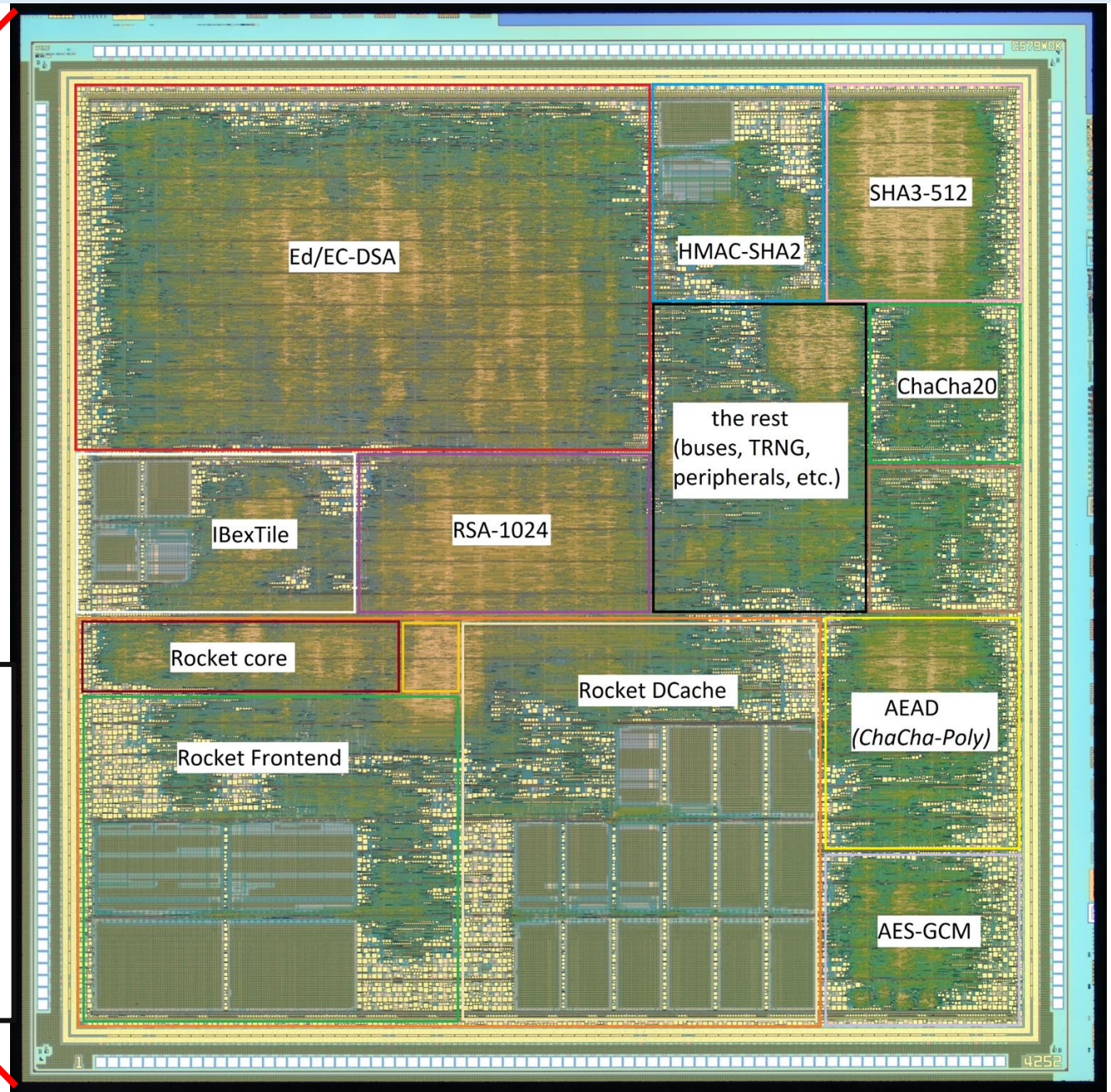
2. Achievement highlight (4/8) Crypto-SoC timeline

Crypto-SoC

2022

02

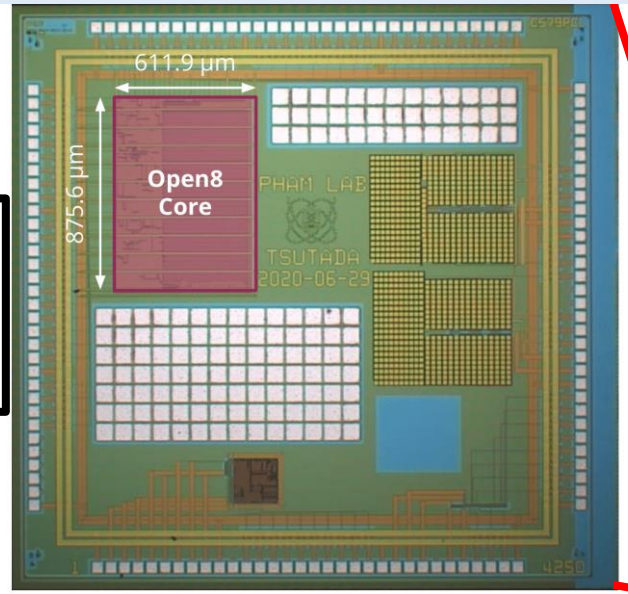
Rocket32(×1)
+ TLS-1.3 Crypto-cores
+ TRNG
+ Secure boot
ROHM-180nm
5.0×5.0-mm2



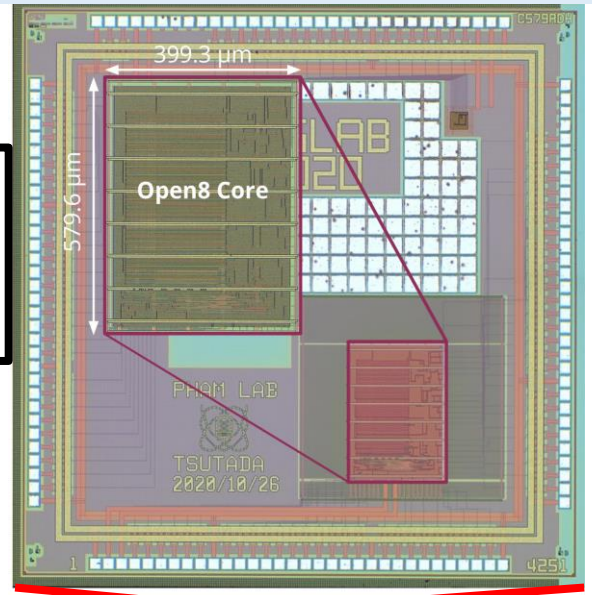
2. Achievement highlight (5/8) ULP-SoC timeline

ULP-SoC

Open8: 8-bit
ROHM-180nm
2.5×2.5-mm2

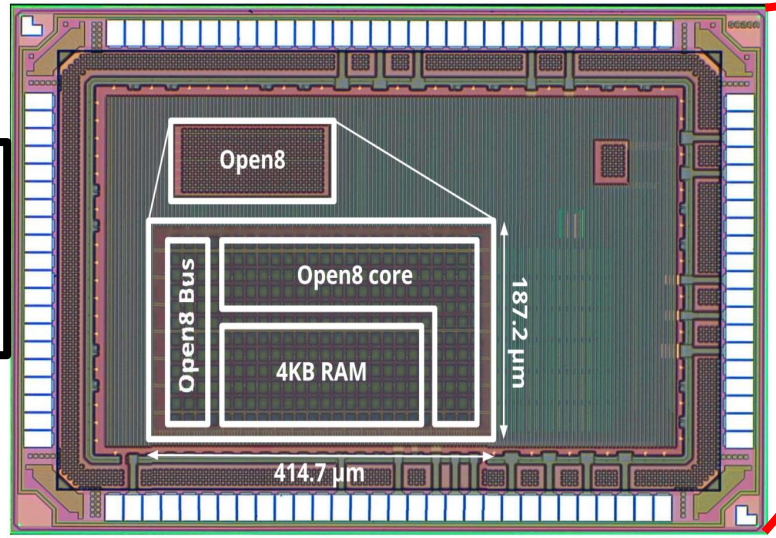


Open8: 8-bit
ROHM-180nm
2.5×2.5-mm2

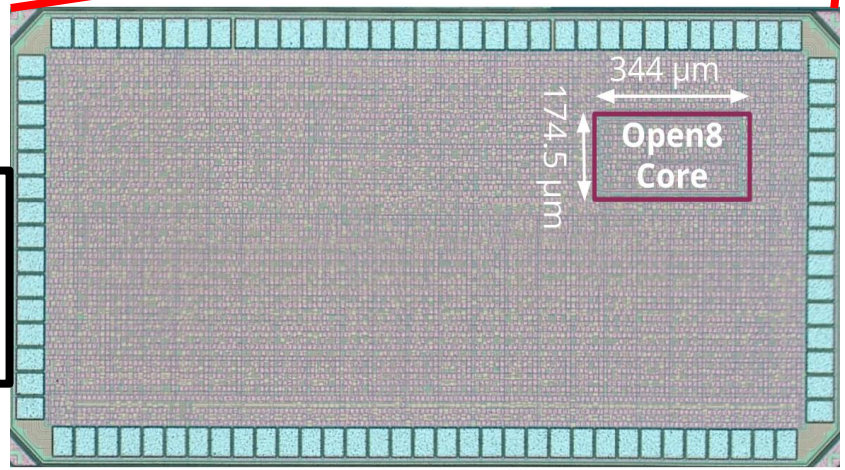


2020 — 06 — 08 — 10 — 2021 — 12

Open8: 8-bit
SOTB-65nm
1.5×2.0-mm2



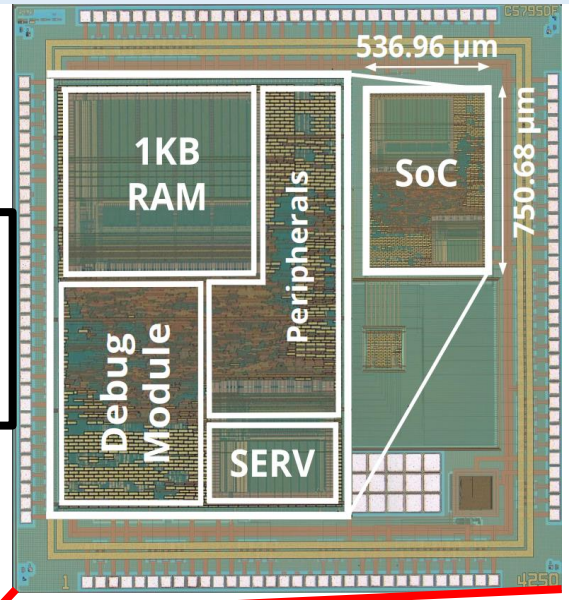
Open8: 8-bit
TSMC-65nm
1.0×2.0-mm2

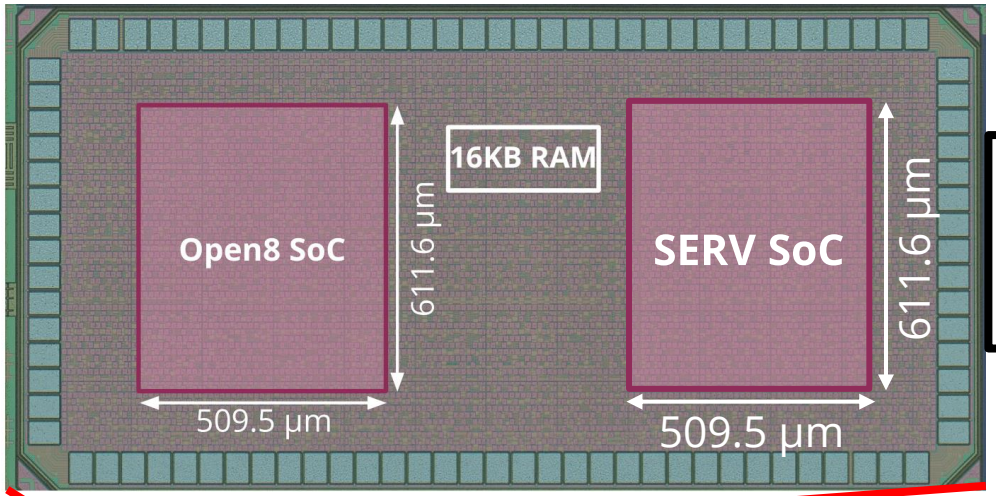


2. Achievement highlight (6/8) ULP-SoC timeline

ULP-SoC

SERV: 32-bit
ROHM-180nm
2.5×2.5-mm2





Open8+SERV
TSMC-65nm
1.0×2.0-mm2

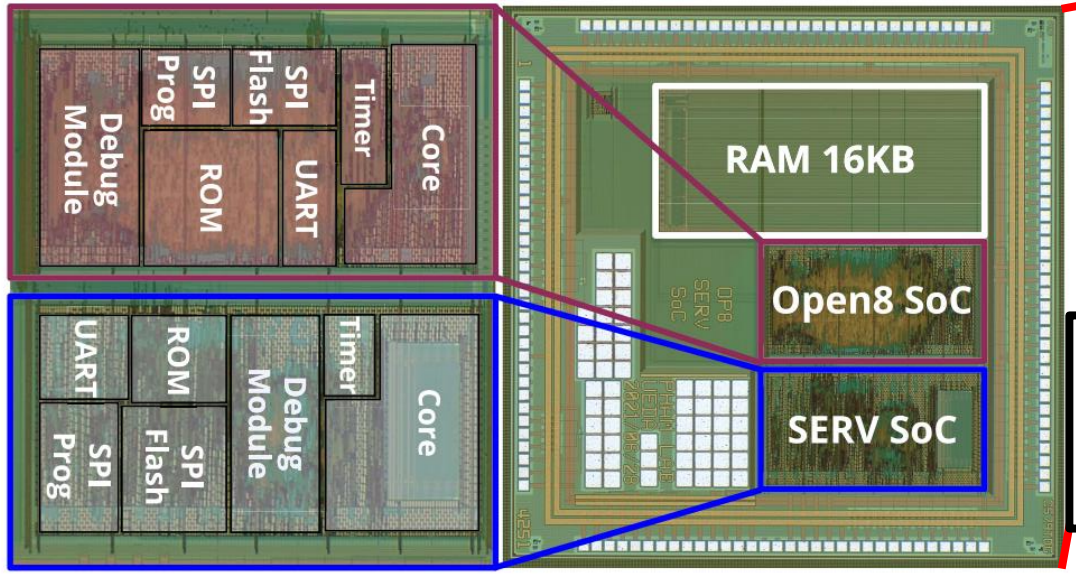
2021

2022

02

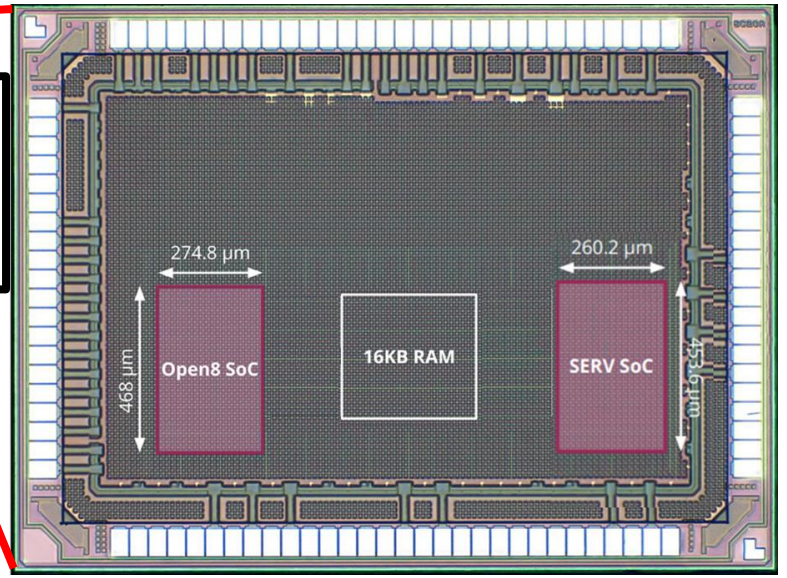
06

07



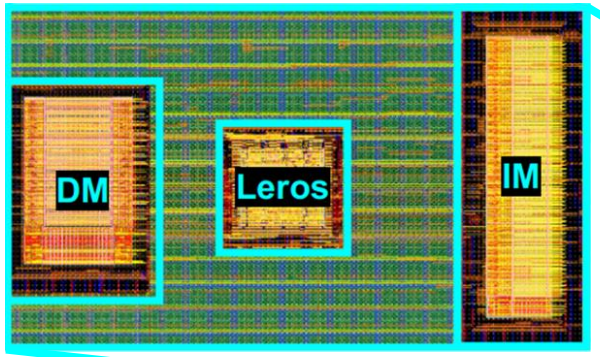
Open8+SERV
ROHM-180nm
2.5×2.5-mm2

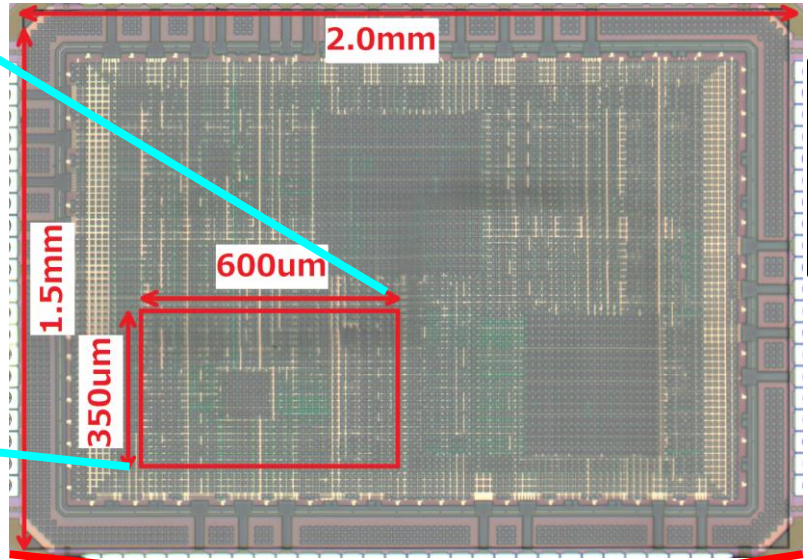
Open8+SERV
SOTB-65nm
1.5×2.0-mm2



2. Achievement highlight (7/8) ULP-SoC timeline

ULP-SoC

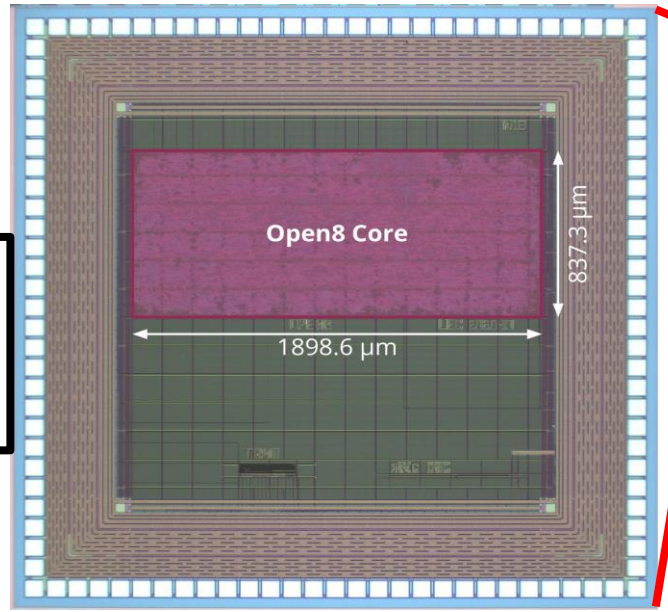




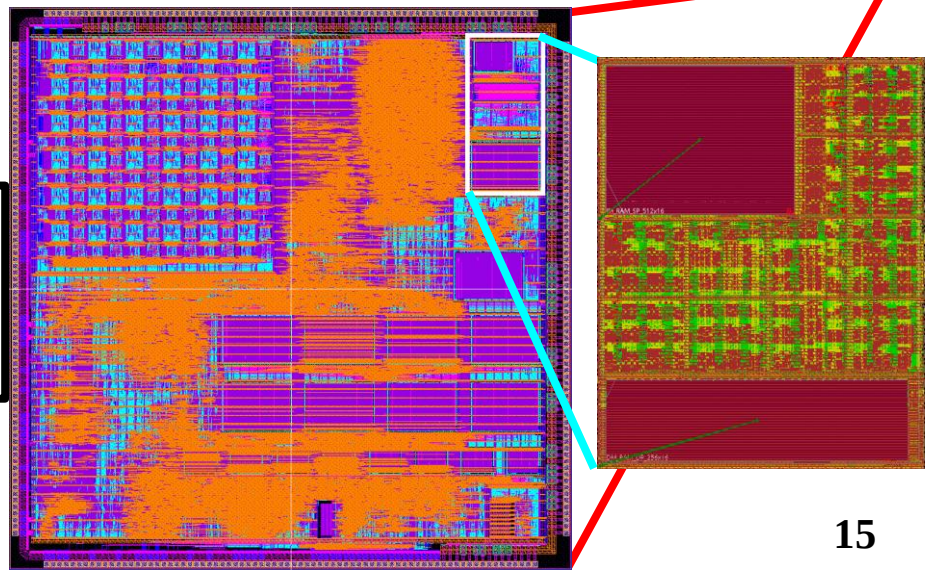
Leros: 16-bit
SOTB-65nm
1.5×2.0-mm2

2022

06 — 11 — 06 — 03



Open8: 8-bit
LAPIS-220nm
2.9×2.9-mm2



Leros: 16-bit
ROHM-180nm
2.5×2.5-mm2

2. Achievement highlight (8/8) Summary

- Since 2019 to now: **4 years**
- A core team of **five members**
- **20+** chips: FullChip designs
- **Linux** bootable SoC
- Two big *past* projects: **Crypto-SoC** and **ULP-SoC** *(and other small projects)*
- **16** Journals and **11** Conferences

All these achievements are thanks to **RISC-V**.

OUTLINE

1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

3. What is RISC-V (1/10) CISC vs. RISC

CISC

(**C**omplex **I**nstruction **S**et **C**omputer)

1) Emphasis on hardware
2) Includes multi-clock complex instructions
3) Memory-to-memory mindset
4) Small code sizes, high cycles/s
5) Transistors used for storing complex instructions

RISC

(**R**educed **I**nstruction **S**et **C**omputer)

1) Emphasis on software
2) Single-clock reduced instructions only
3) Register-to-register mindset
4) Large code sizes, low cycles/s
5) Spends more transistors, and most of them are used for storing data

RISC win CISC win

$$\textbf{Performance} = \frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycle}}{\text{instruction}} \times \frac{\text{instruction}}{\text{program}}$$

RISC-V simply means *RISC* architecture version *five*

Nowadays, almost all processors in the market are RISCs.

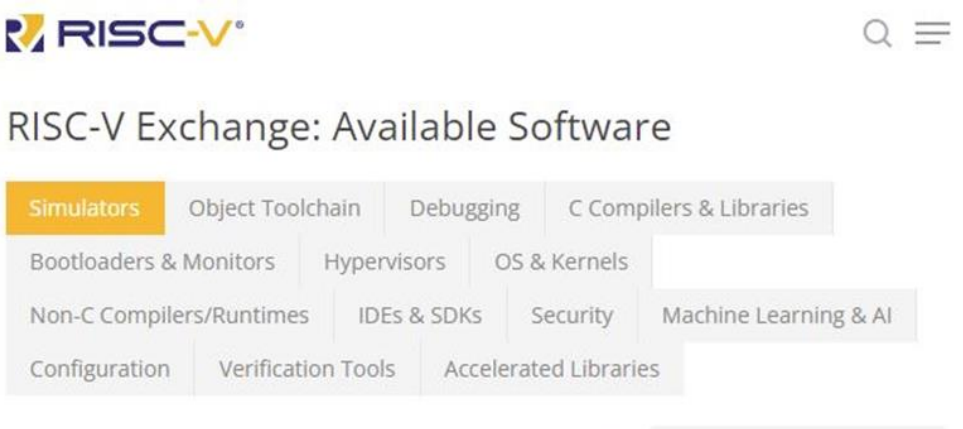
Economic reason: memory price is way down ↓ ↓ ↓

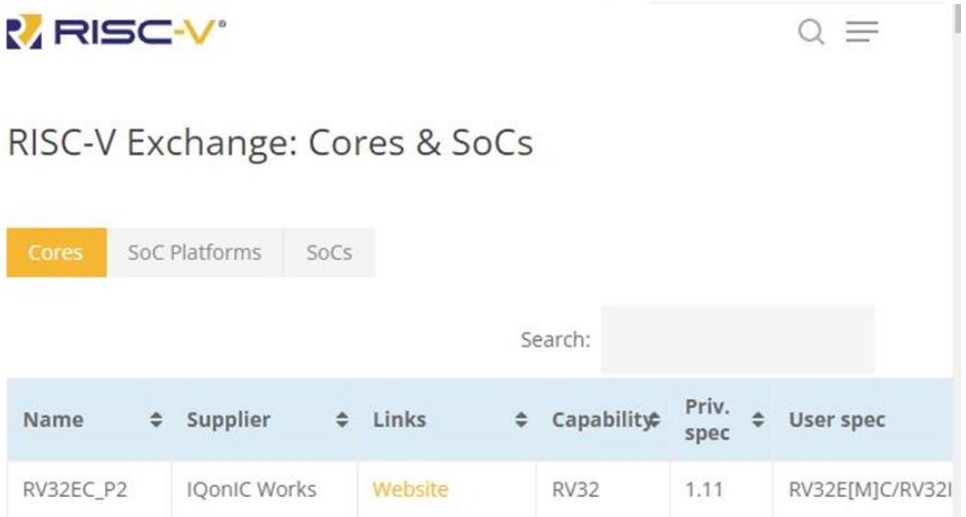
3. What is RISC-V (2/10) RISC-V?

Open-source **RISC-V** means open-source **ISA**, no more, no less.

(ISA: Instruction Set Architecture)

(some other common ISAs: i386, amd64, ARM 32/64, AVR, MIPS, NiosII, etc.)





RISC-V Foundation: https://riscv.org/

- Official released ISA specification
- Many cores, SoCs, & software are available
- Developers can reuse each other designs & tools
  → significantly reducing R&D time and effort

Licensed free:

- RISC-V ISA
- RISC-V toolchain

License depended on authors/developers:

- RISC-V processors
- RISC-V software applications
- RISC-V-related products

3. What is RISC-V (3/10) ISA?

ISA means **I**nstruction **S**et **A**rchitecture.
It is the layer between software and hardware developers.

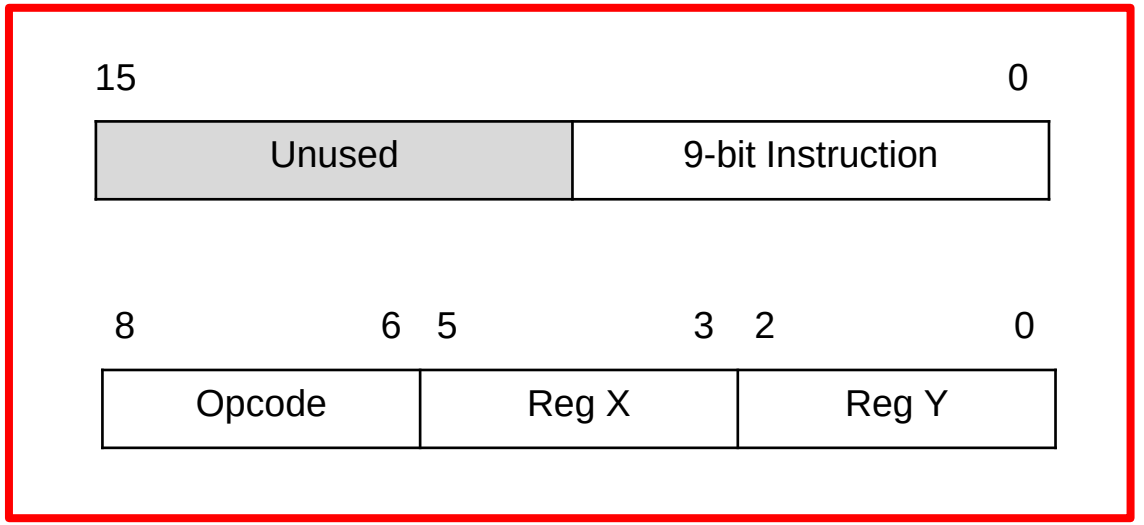
Software tools: assembler, compilers, debugger, linker, etc.

ISA: the interface between software & hardware architects

Processor: ALU, FPU, registers, CSRs, branch predictor, caches, etc.

ISA has to define all these kinds of stuff:

1) How many instructions, and which is which?
2) In an instruction, what field means what?
3) Addressing & data-path *(8/16/32/64/128-bit)*?
4) What is supported and what is not?
5) *etc.*



3. What is RISC-V (4/10) RISC-V ISA

What makes **RISC-V** different: its modular mindset

(modular architecture helps fine-tune the performance based on the developer's needs [1])

Base instruction set: **I**nteger

Extended instruction set: *the rest*

Extension	Description
I	Integer
M	Integer Multiplication and Division
A	Atomics
F	Single-Precision Floating Point
D	Double-Precision Floating Point
G	General Purpose = IMAFD
C	16-bit Compressed Instructions
Non-Standard User-Level Extensions	
Xext	Non-standard extension "ext"

The most common extensions: **IMAFDC** *(also known as **GC**)*

There are also a lot more than just **IMAFDC :**

Base	Version	Status
RVWMO	2.0	**Ratified**
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*
Extension	Version	Status
M	**2.0**	**Ratified**
A	**2.1**	**Ratified**
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
Zicsr	**2.0**	**Ratified**
Zifencei	**2.0**	**Ratified**
Zam	*0.1*	*Draft*
Ztso	*0.1*	*Frozen*

3. What is RISC-V (5/10) RISC-V OS stack

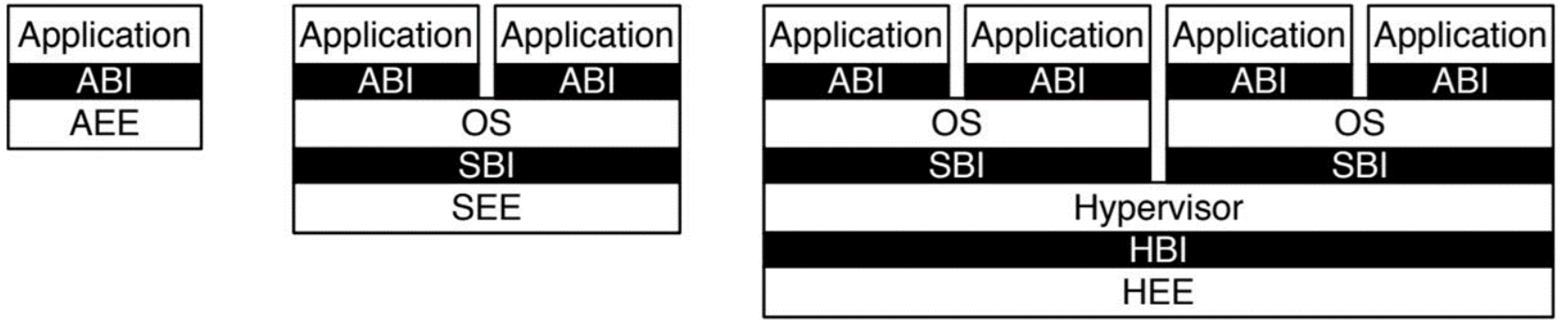
To support an Operating System (OS), the ISA has to support the OS stack *or the* ***M-/S-/U-mode***.

RISC-V privileged architecture:

RISC-V Modes		
Level	Name	Abbr.
0	User/Application	U
1	Supervisor	S
	Reserved	
3	Machine	M

Supported Combinations of Modes	
Supported Levels	Modes
1	M
2	M, U
3	M, S, U

Different scenarios of utilizing the OS stack:

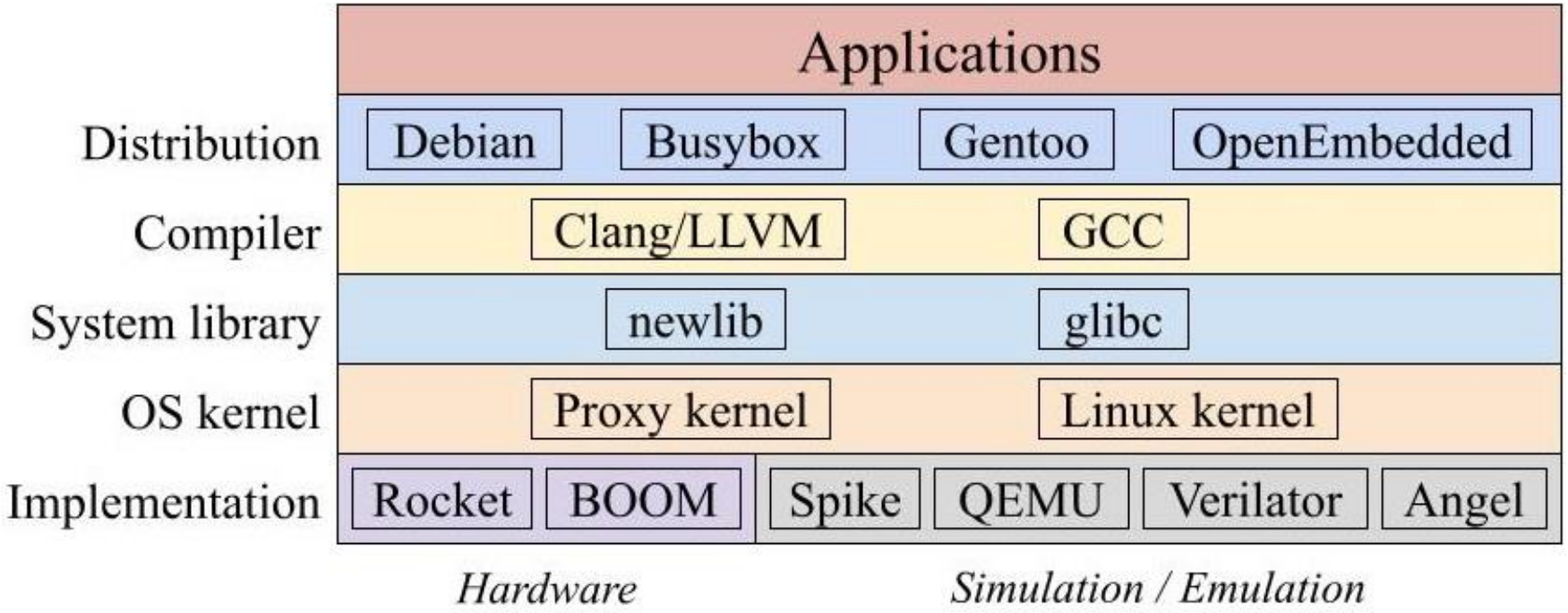


RISC-V ISA not only supports the OS stack,
but also provides a **privileged architecture** [2].

→ Better security scheme by having the hardware recognize different codes executed at different modes.

3. What is RISC-V (6/10) RISC-V ecosystem

RISC-V toolchain and its ecosystem



(User's) software in the top

OS file system

Compiler

Standard libraries

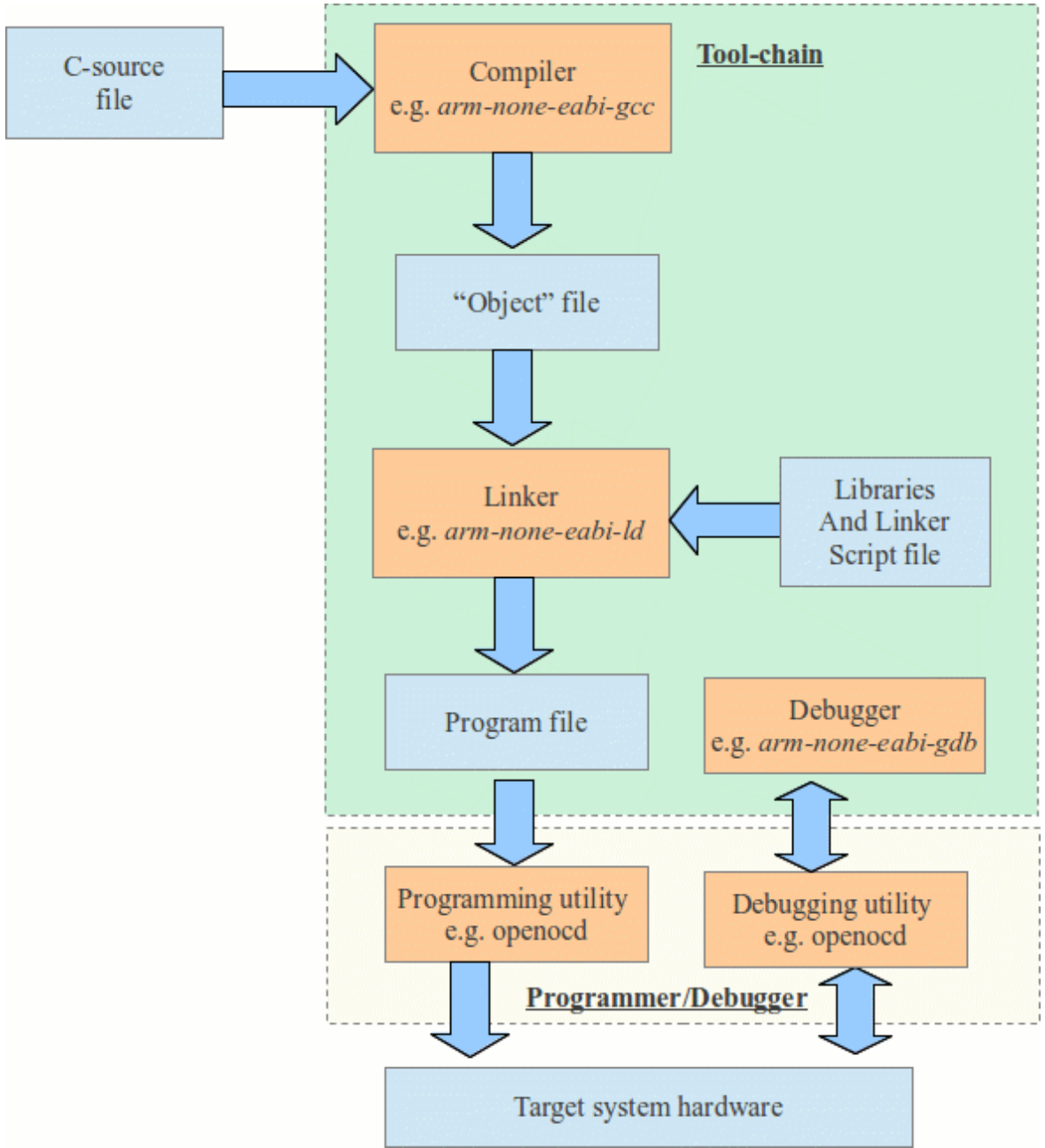
OS kernel

Hardware in the bottom

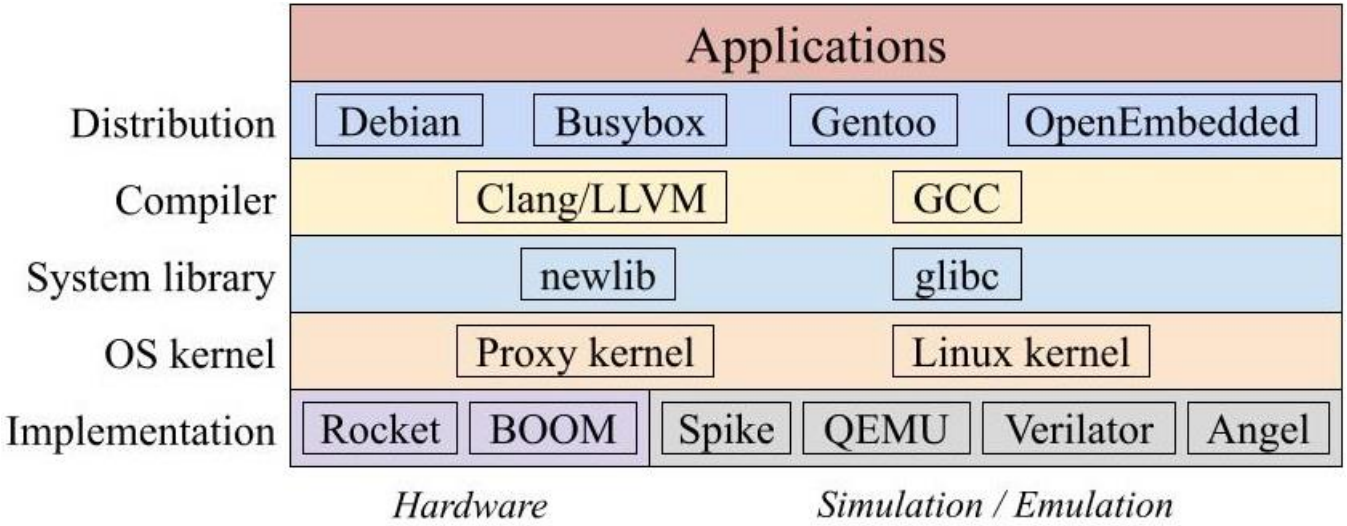
Top-down explanation:

User's applications on the top are operated in an OS file system, which then compiled by a compiler based on multiple standard libraries. After compiled, the execution file is run on the OS kernel that manages the hardware in the bottom.

3. What is RISC-V (7/10) RISC-V toolchain



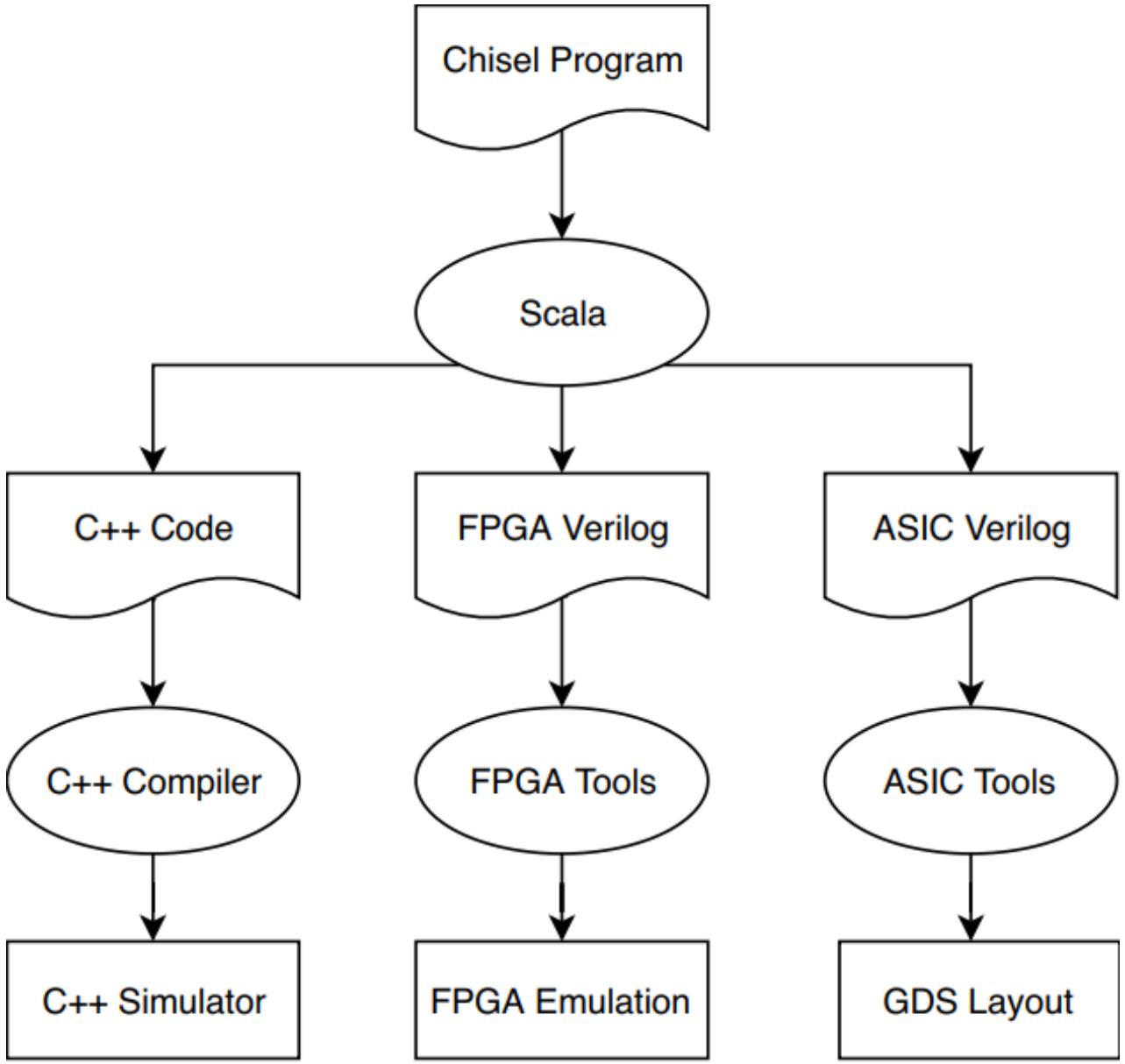
RISC-V toolchain and its ecosystem



Three most important tools

- **GCC:** *(cross C compiler)* makes a C code into assembly code
- **LD:** *(linker)* links standard libraries into the build; also links between multiple C files
- **GDB:** *(debugger)* debug the hardware/simulator/emulator

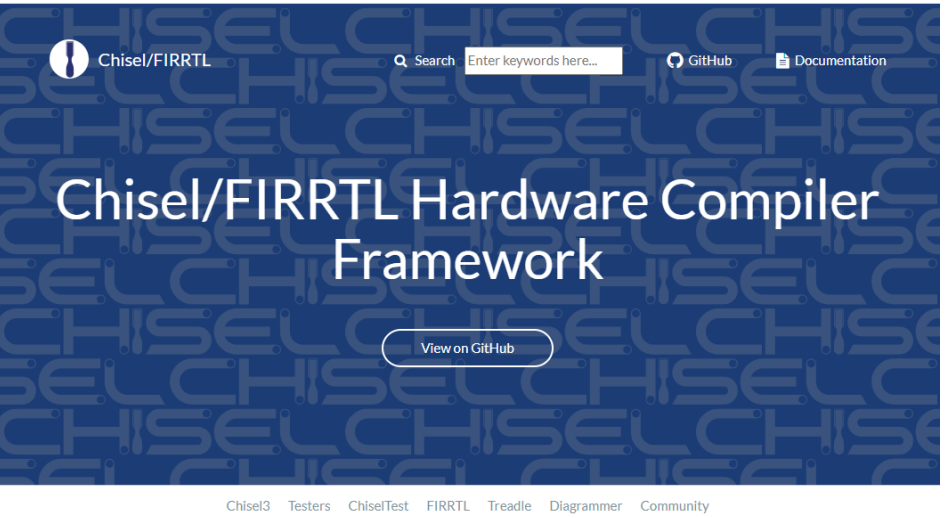
3. What is RISC-V (8/10) CHISEL: a new way to code



Chisel is a library.
Scala is a language.

- **Scala** itself is a *high-level object-oriented programming language*
  → It is not designed for "hardware coding."
- **Chisel** is a library attached to Scala to define a set of coding rules.
  → It is designed for "hardware coding."
- From **Scala** to **Verilog**:
  Scala → Java → FIRRTL → Verilog
  1st arrow: Scala compiler named SBT
  2nd arrow: executing Java
  3rd arrow: FIRRTL compiler

3. What is RISC-V (9/10) CHISEL: a new way to code

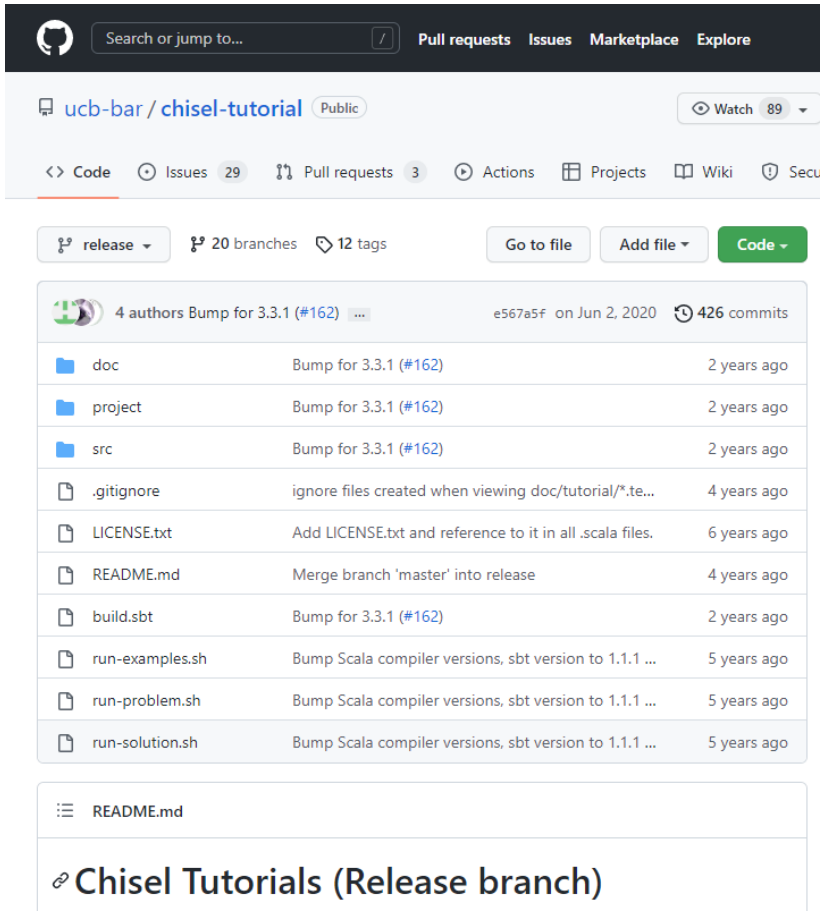


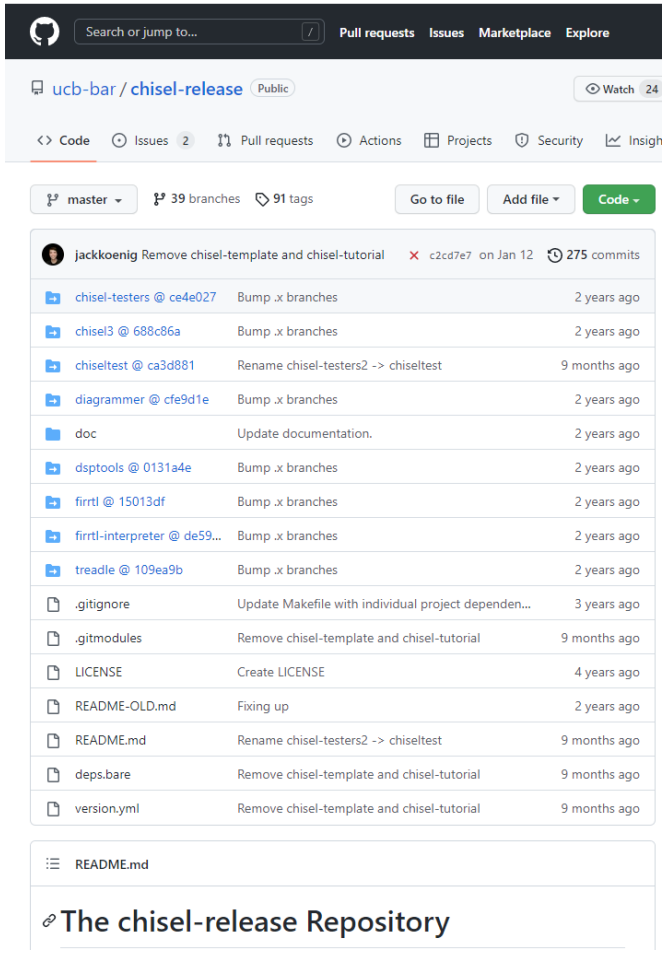
CHISEL

CCC 2022

The official CHISEL website:
https://www.chisel-lang.org/

CHISEL tutorials can be found at:
https://github.com/ucb-bar/chisel-tutorial





Released CHISEL sources:
https://github.com/ucb-bar/chisel-release

3. What is RISC-V (10/10) RISC-V key factors

RISC-V revolutionizes Computer System Design

1. **Modular at heart:**
   customizable ISA and customizable hardware
   → fine-tune the system to your specific needs.

2. **Open-source community:**
   license-free ISA, open cores and SoCs, open-source libraries, open-source software, etc.
   → reuse other developers' designs → save time and effort for R&D

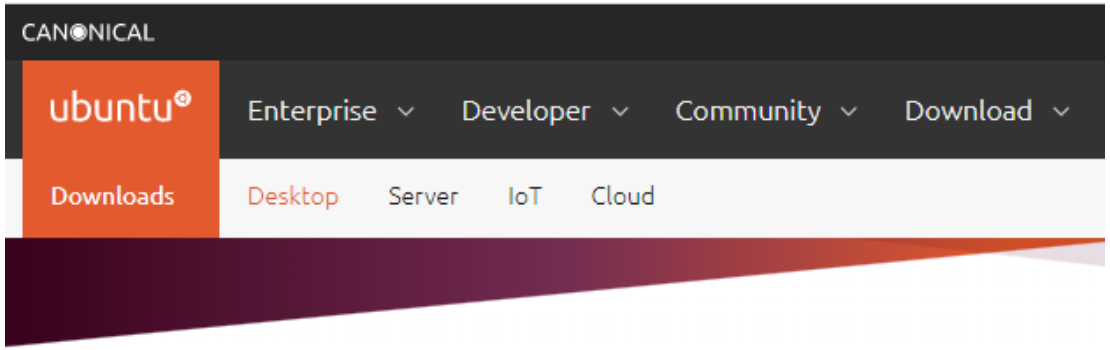
3. **CHISEL** ***(Constructing Hardware In Scala Embedded Language)*****:**
   a new way to "coding" hardware circuits. When compiled, it will generate a true RTL Verilog code.
   → a "meta-programming" language for hardware developers with parameters and sub-designs that can be overridden and extended.
   → easy to develop an "object-oriented" hardware library for reuse purposes.

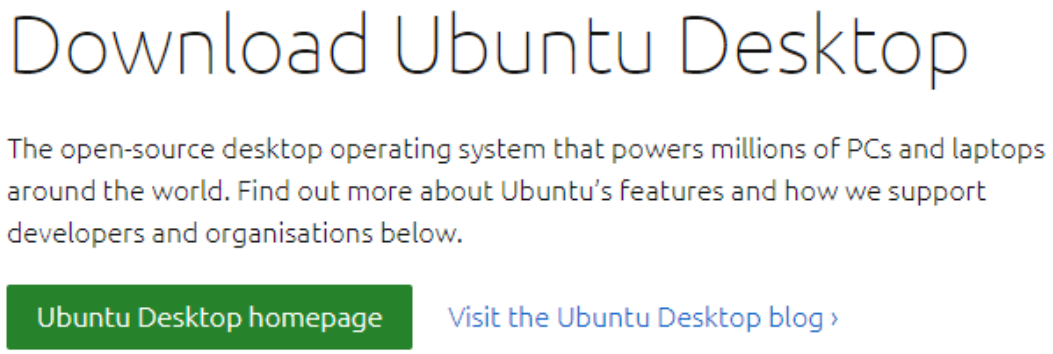
OUTLINE

1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

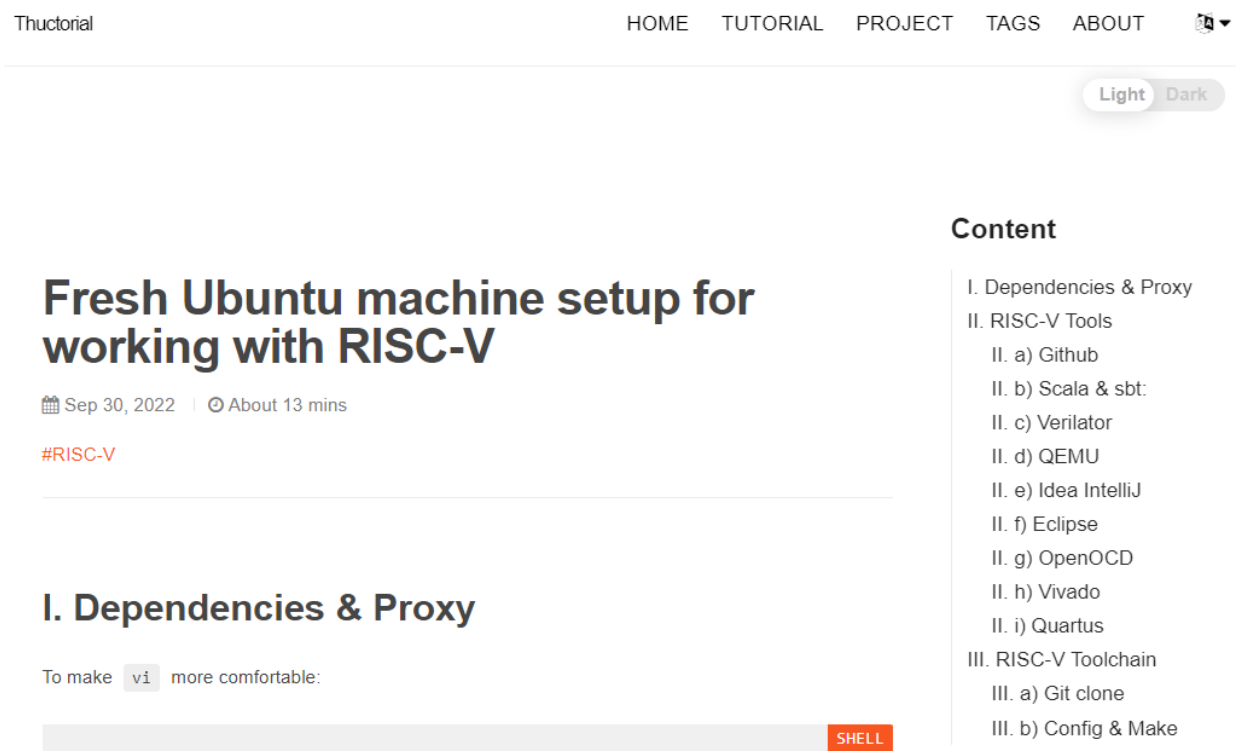
4. Working with RISC-V (1/21) Unix machine

First of all, you'll need an Ubuntu machine (recommend version 20.04-LTS)





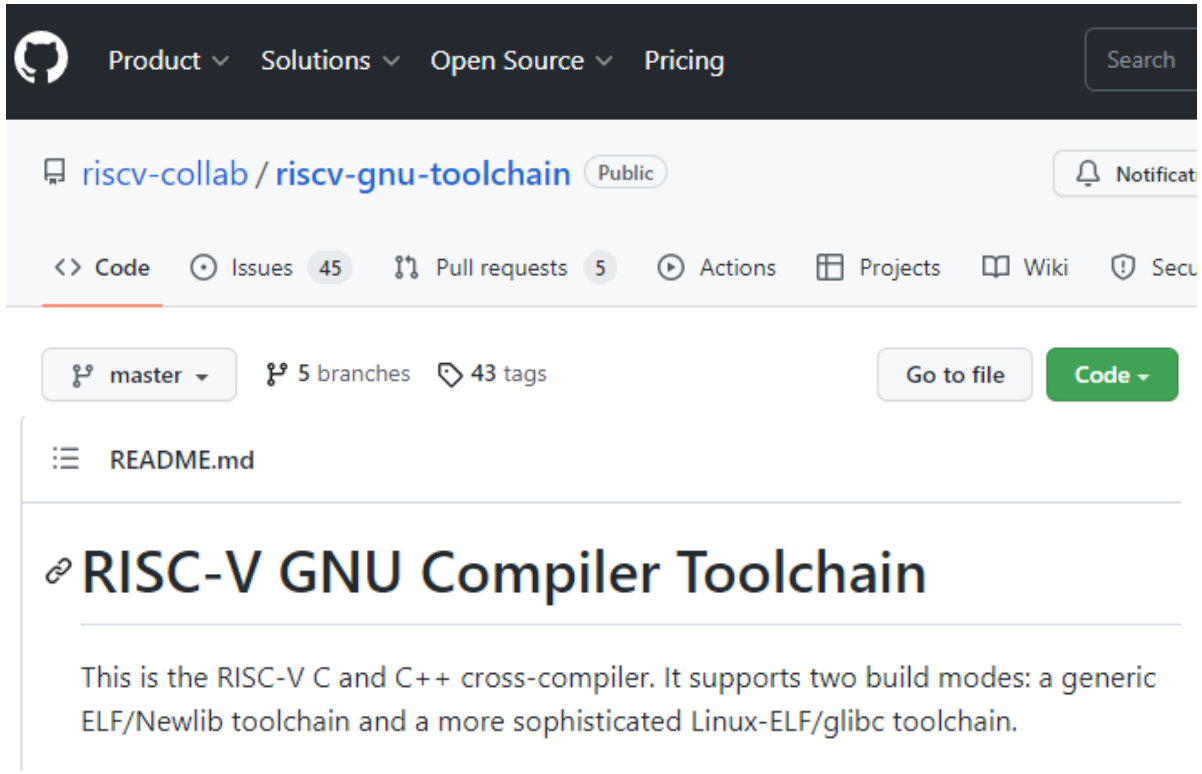
You can follow my guide to set up a new Ubuntu machine for working with RISC-V: https://thuchoang90.github.io/tutorial/2022/09/30/Fresh-Ubuntu-setup



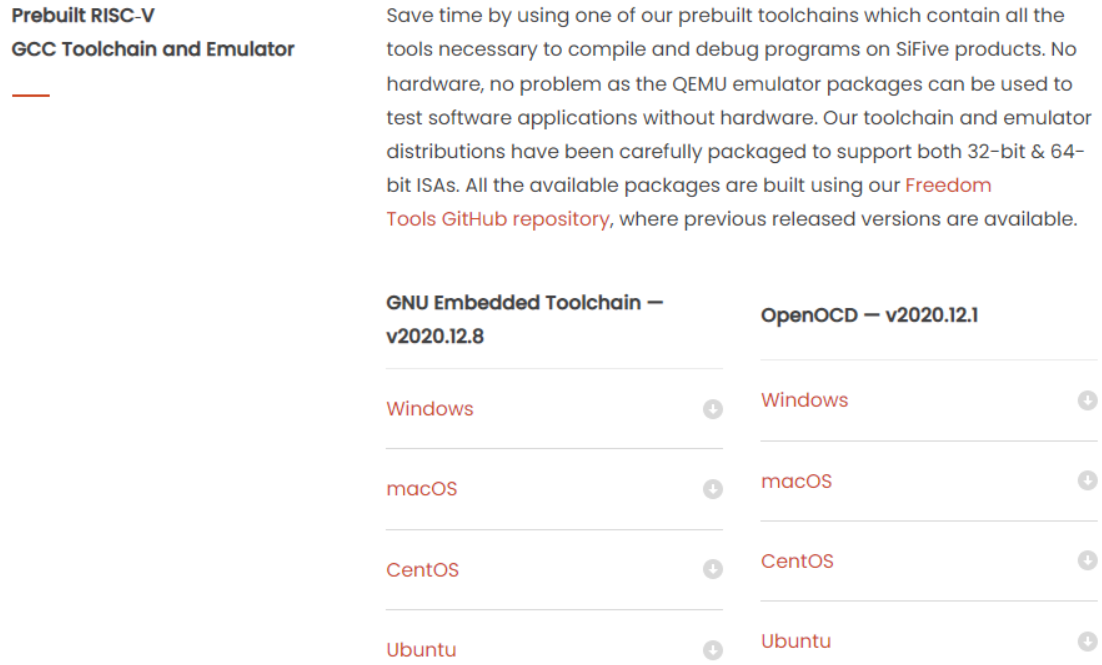
4. Working with RISC-V (2/21) RISC-V toolchain

The next step, preparing the RISC-V GNU toolchain, is the essential part.

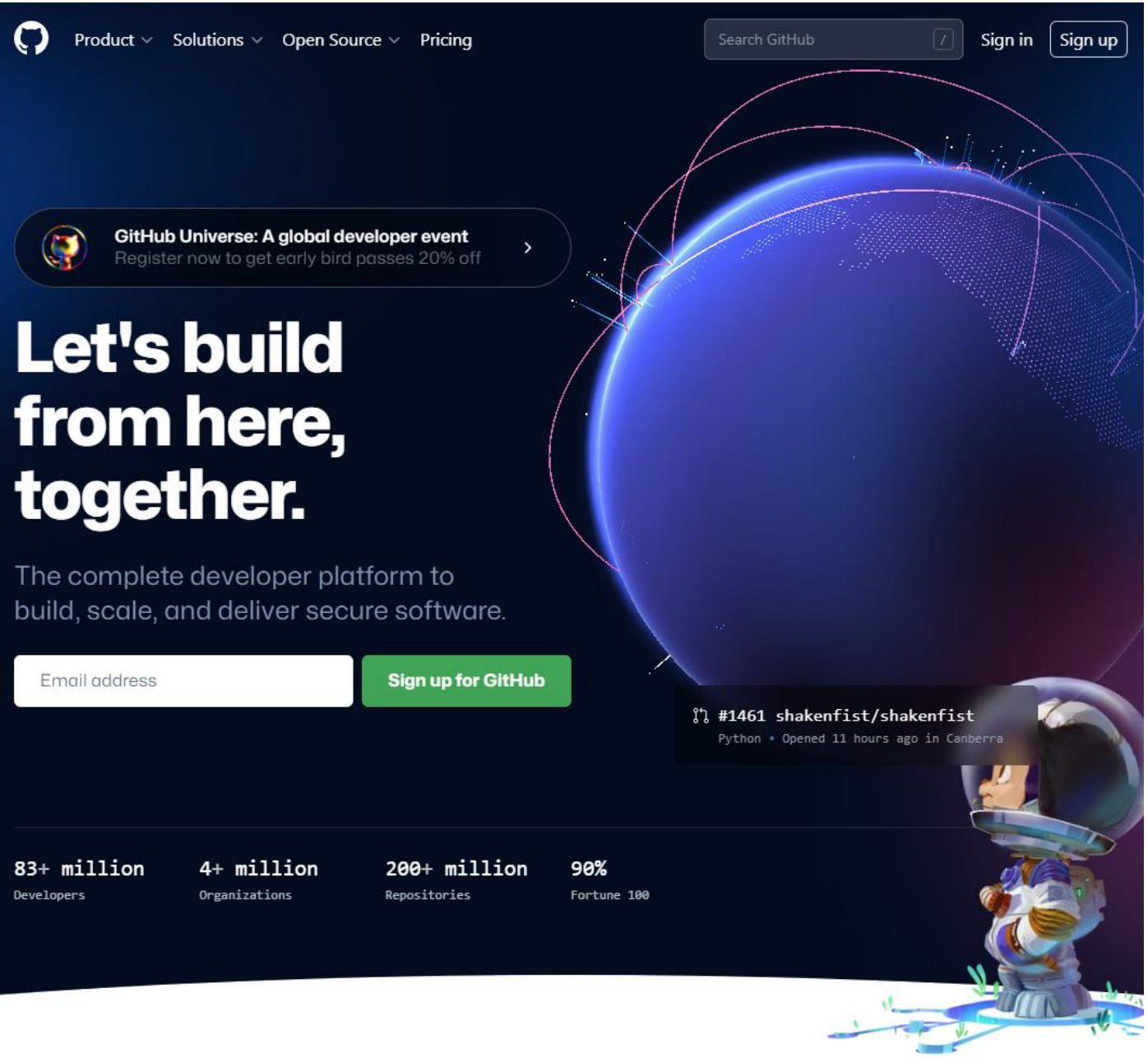
Official GitHub link for compiling from source: https://github.com/riscv-collab/riscv-gnu-toolchain



You can also download a pre-built toolchain on the SiFive webpage: https://www.sifive.com/software *(however, I recommend compiling from source)*



4. Working with RISC-V (3/21) Github



When working with the RISC-V open-source community, you'll be using GitHub a lot.

So it's better to set up a GitHub account for personal use.

https://github.com/

4. Working with RISC-V (4/21) Hardware simulate/emulate

Verilator

- A simulator
- Simulate a Verilog code at each clock cycle

- A tool for hardware developers
- Web: https://www.veripool.org/verilator/

QEMU

- An emulator
- Emulate an ideal RISC-V processor for software to run *(exactly like running a virtual machine; in this case, RISC-V on Linux)*
- A tool for software developers
- Web: https://www.qemu.org/

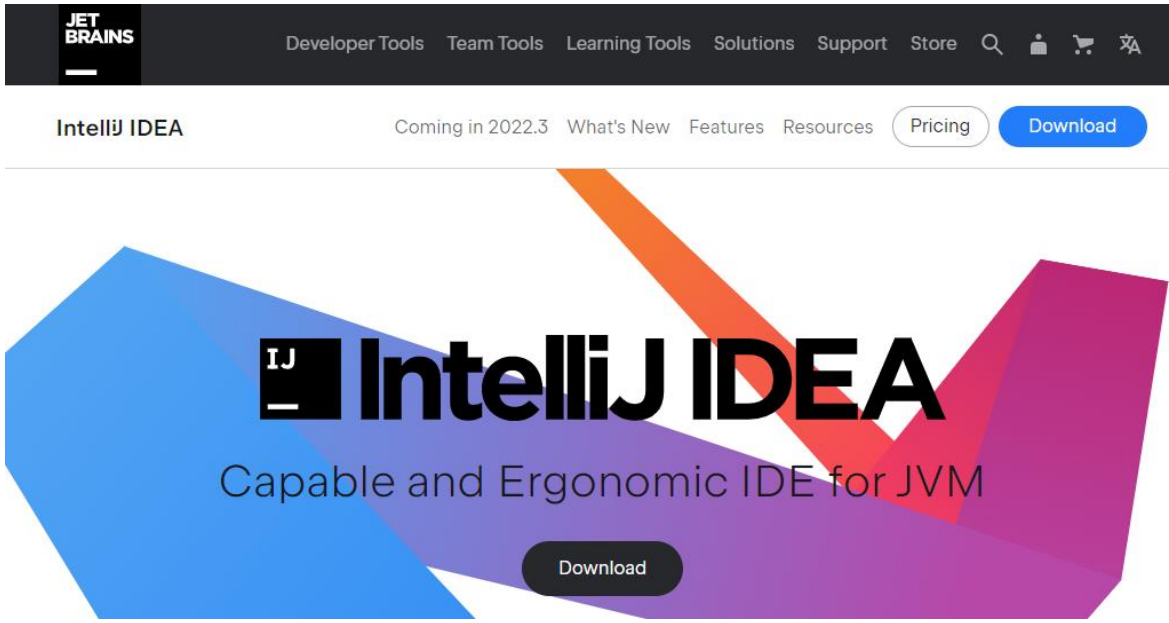
4. Working with RISC-V (5/21) Working with CHISEL

Remind the Chisel-to-Verilog compile flow:
Scala (+Chisel) → Java → FIRRTL → Verilog

- Scala: a programming language
- Chisel: a library *(attached to Scala)*
- Sbt: a Scala compiler

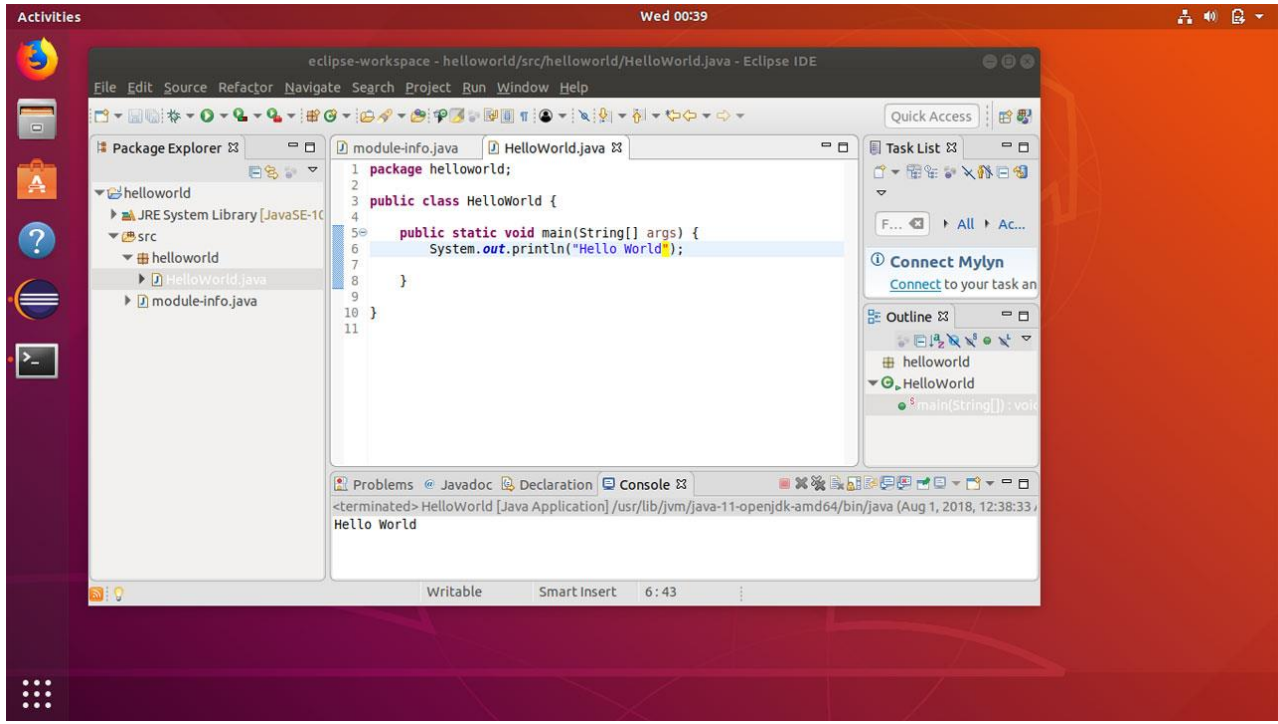
So, we need to install the **Sbt** tool.
Webpage: https://www.scala-sbt.org/

The **Sbt** tool itself is a terminal command line.
Recommend installing **IntelliJ IDEA**, a GUI for Sbt *(like a Visual Studio for C/C++)*.
Webpage: https://www.jetbrains.com/idea/



4. Working with RISC-V (6/21) Software (C/C++) coding

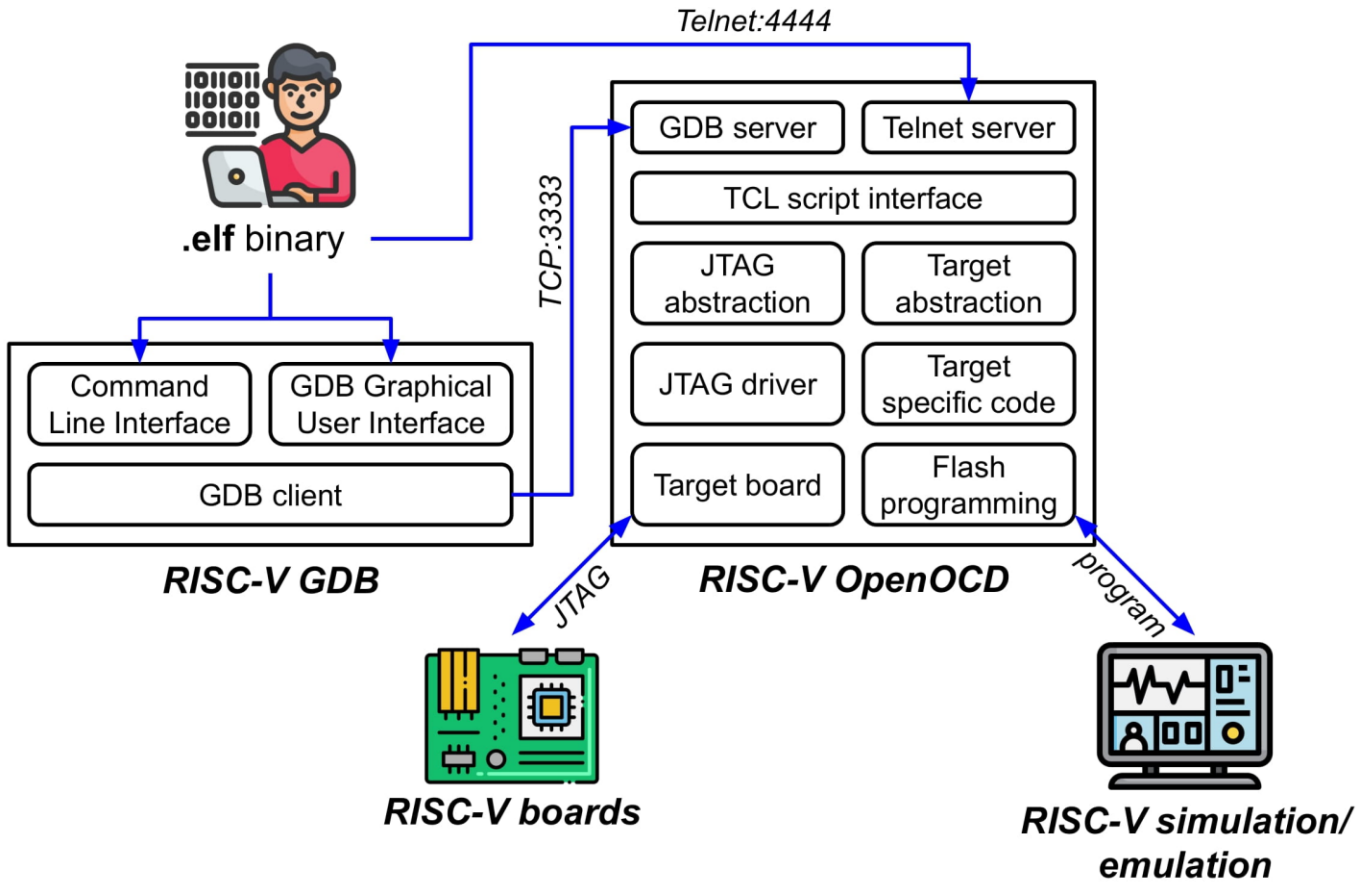
For software coding, especially C/C++, recommend installing **Eclipse**.
Webpage: https://www.eclipse.org/



4. Working with RISC-V (7/21) Debug tools

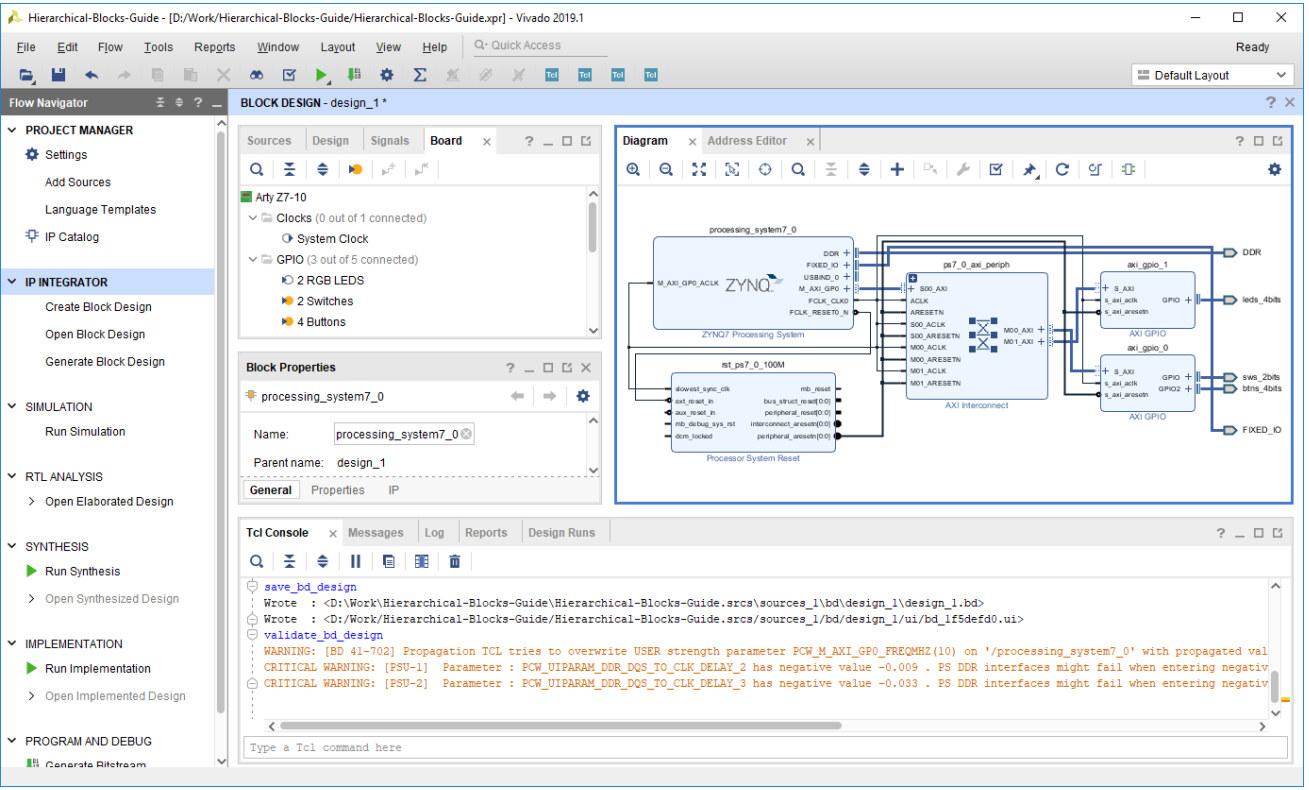
OpenOCD is the tool used for debugging *(together with RISC-V GDB in the toolchain)*:
Webpage: https://openocd.org/
GitHub: https://github.com/riscv/riscv-openocd

A typical debug flow with GDB and OpenOCD:

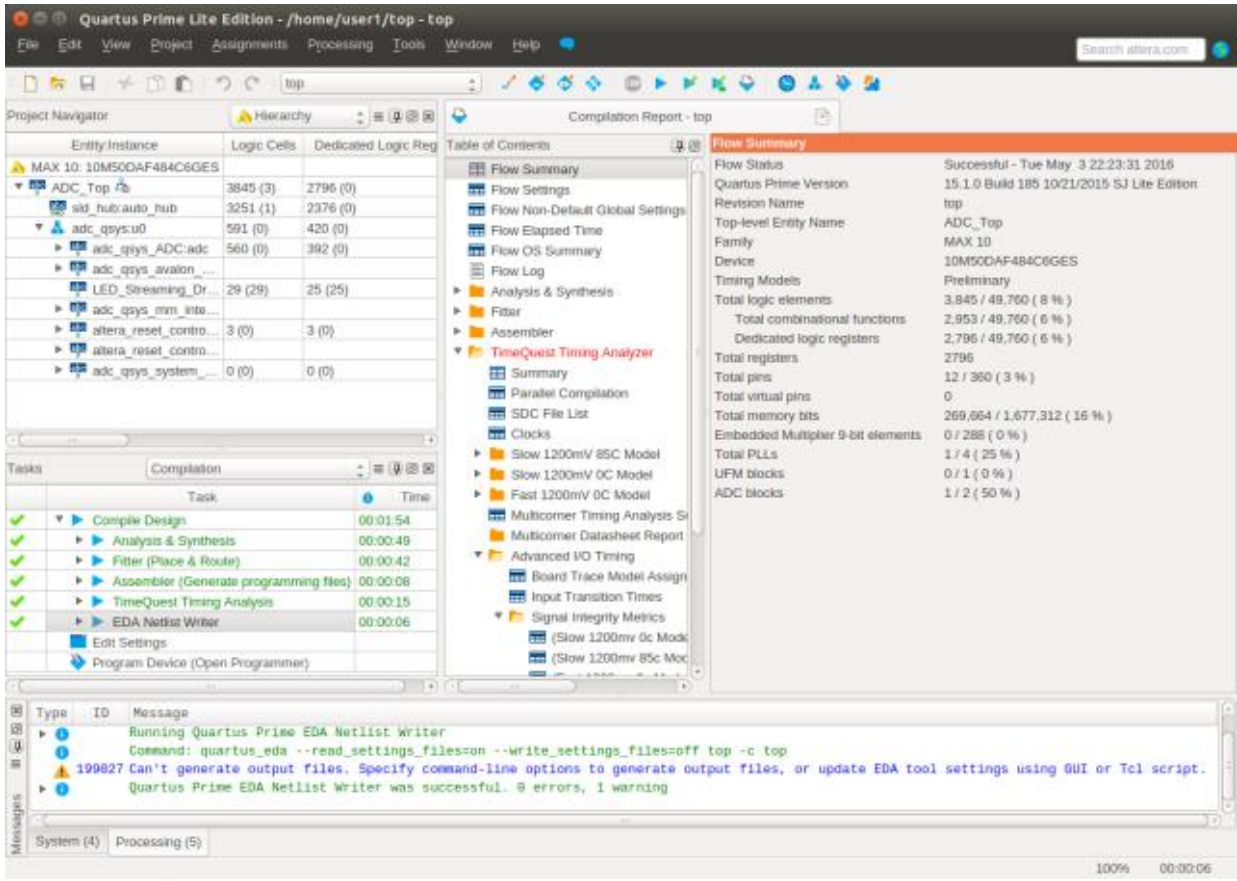


4. Working with RISC-V (8/21) Working with FPGA

Finally, if you are working with FPGAs, you must install Vivado for Xilinx or Quartus for Altera boards.



Vivado ML standard



Quartus Prime Standard

Web:
https://www.xilinx.com/support/download.html

Web:
https://www.intel.com/content/www/us/en/collections/products/fpga/software/downloads.html

4. Open community (9/21) Open libraries

The common open RISC-V libraries that you can use

Chipyard *(contains many common and frequently used open IPs, including RISC-V processors and other peripherals such as uart, spi, sd-card, etc.)*: *https://github.com/ucb-bar/chipyard*

README.md

CHIPYARD

Chipyard Framework

chipyard-ci-process passing

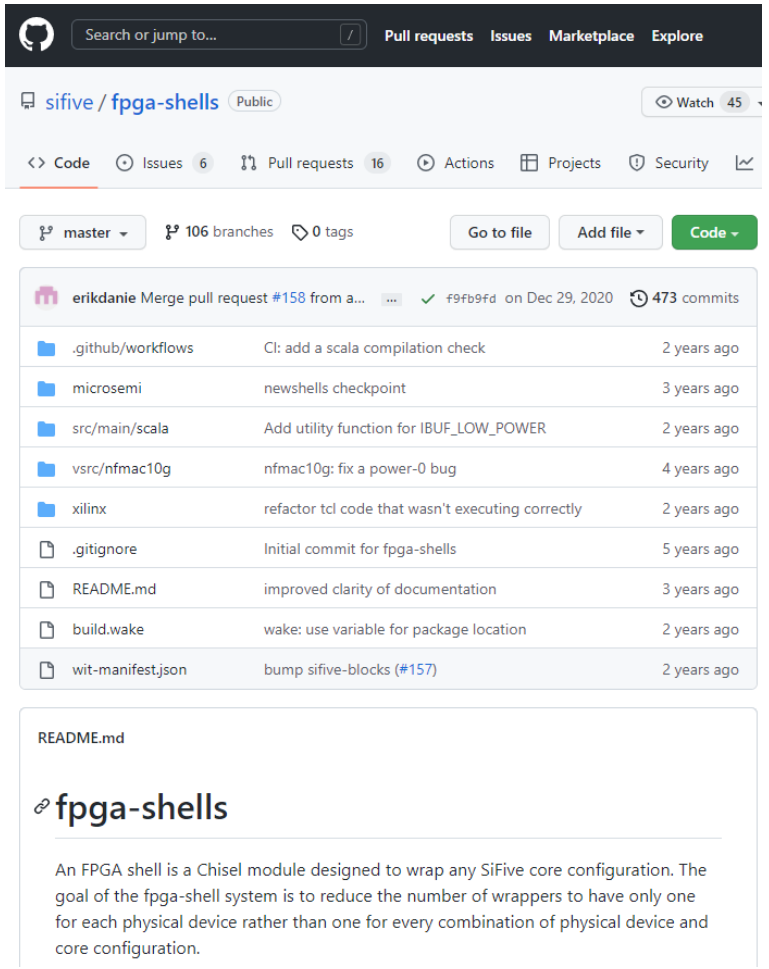
Quick Links

- **Stable Documentation:** https://chipyard.readthedocs.io/
- **User Question Forum:** https://groups.google.com/forum/#!forum/chipyard
- **Bugs and Feature Requests:** https://github.com/ucb-bar/chipyard/issues

Using Chipyard

To get started using Chipyard, see the stable documentation on the Chipyard documentation site: https://chipyard.readthedocs.io/

What is Chipyard



fpga-shells *(contains many common FPGA configurations)*: *https://github.com/sifive/fpga-shells*

4. Open community (10/21) Open processors

Some famous RISC-V processors

Rocket is the most popular among RISC-V processors: https://github.com/chipsalliance/rocket-chip

(it is an in-of-order processor)

README.md

Rocket Chip Generator

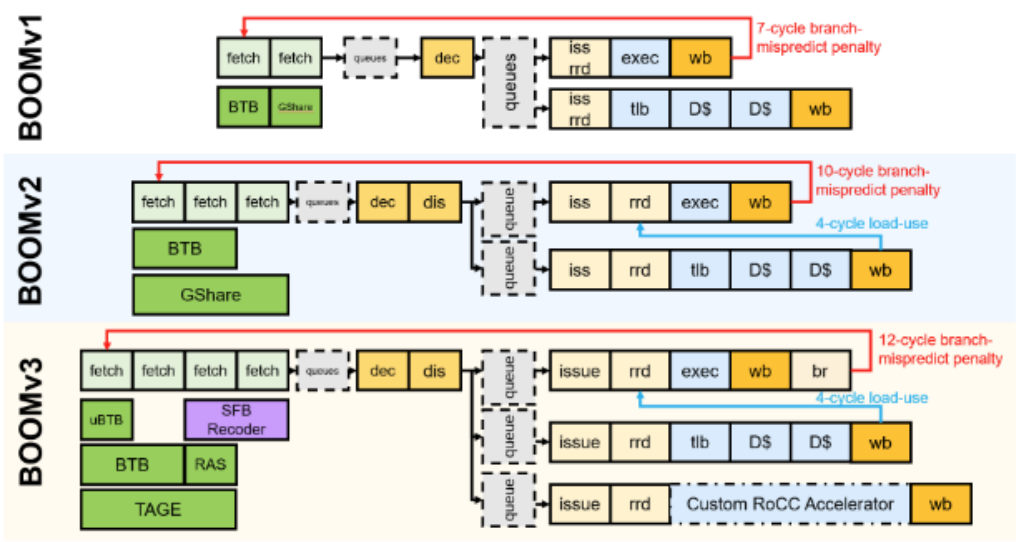
Continuous Integration passing

This repository contains the Rocket chip generator necessary to instantiate the RISC-V Rocket Core. For more information on Rocket Chip, please consult our technical report.

Table of Contents

BOOM is an out-of-order processor that can rival ARM: https://github.com/riscv-boom/riscv-boom

README.md



The Berkeley Out-of-Order RISC-V Processor

FAILED

4. Open community (11/21) Open processors

Some famous RISC-V processors

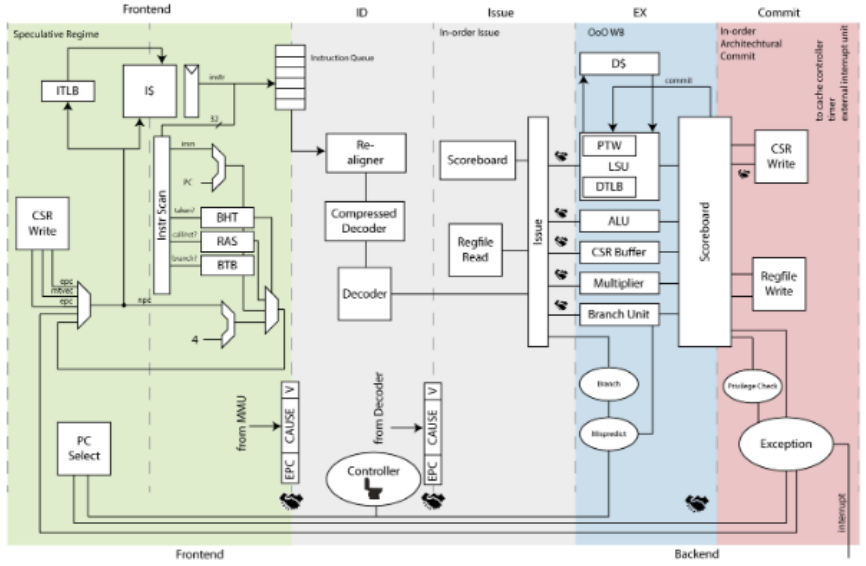
README.md

build unknown

Ariane RISC-V CPU

Ariane is a 6-stage, single issue, in-order CPU which implements the 64-bit RISC-V instruction set. It fully implements I, M, A and C extensions as specified in Volume I: User-Level ISA V 2.3 as well as the draft privilege extension 1.10. It implements three privilege levels M, S, U to fully support a Unix-like operating system. Furthermore it is compliant to the draft external debug spec 0.13.

It has configurable size, separate TLBs, a hardware PTW and branch-prediction (branch target buffer and branch history table). The primary design goal was on reducing critical path length.

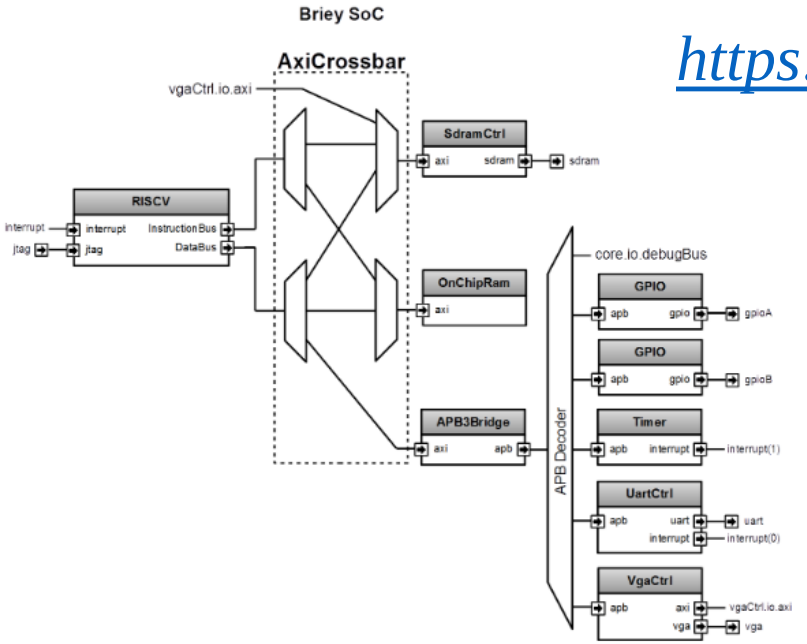


Ariane is a high-performance 64-bit in-of-order processor:
https://github.com/lowRISC/ariane

VexRiscv is a simple in-of-order 32-bit processor that is suited for an MCU:

Briey SoC

As a demonstration, a SoC named Briey is implemented in `src/main/scala/vexriscv/demo/Briey.scala`. This SoC is very similar to the Pinsec SoC:



https://github.com/SpinalHDL/VexRiscv

4. Open community (12/21) Open processors

Some famous RISC-V processors

Ibex is an in-of-order 32-bit processor that focuses on security applications: https://github.com/lowRISC/ibex

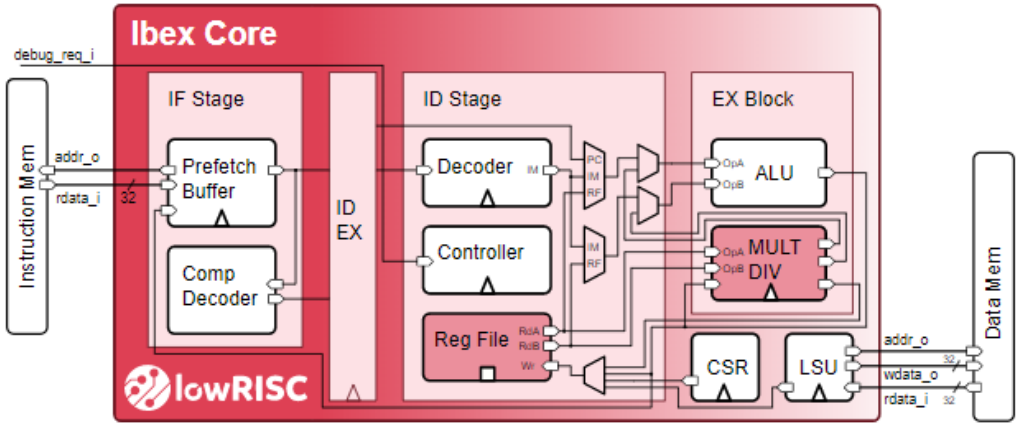
README.md

Azure Pipelines succeeded

Ibex RISC-V Core

Ibex is a production-quality open source 32-bit RISC-V CPU core written in SystemVerilog. The CPU core is heavily parametrizable and well suited for embedded control applications. Ibex is being extensively verified and has seen multiple tape-outs. Ibex supports the Integer (I) or Embedded (E), Integer Multiplication and Division (M), Compressed (C), and B (Bit Manipulation) extensions.

The block diagram below shows the *small* parametrization with a 2-stage pipeline.



Ibex was initially developed as part of the PULP platform under the name "Zero-riscy", and has been contributed to lowRISC who maintains it and develops it further. It is under active development.

4. Open community (13/21) Groups & companies

Some famous RISC-V groups & companies

SiFive is the first and the most famous company that is doing RISC-V-related products:

- Their website: https://www.sifive.com/
- Their github: https://github.com/sifive

HexFive is a company that focuses on RISC-V-based security applications:

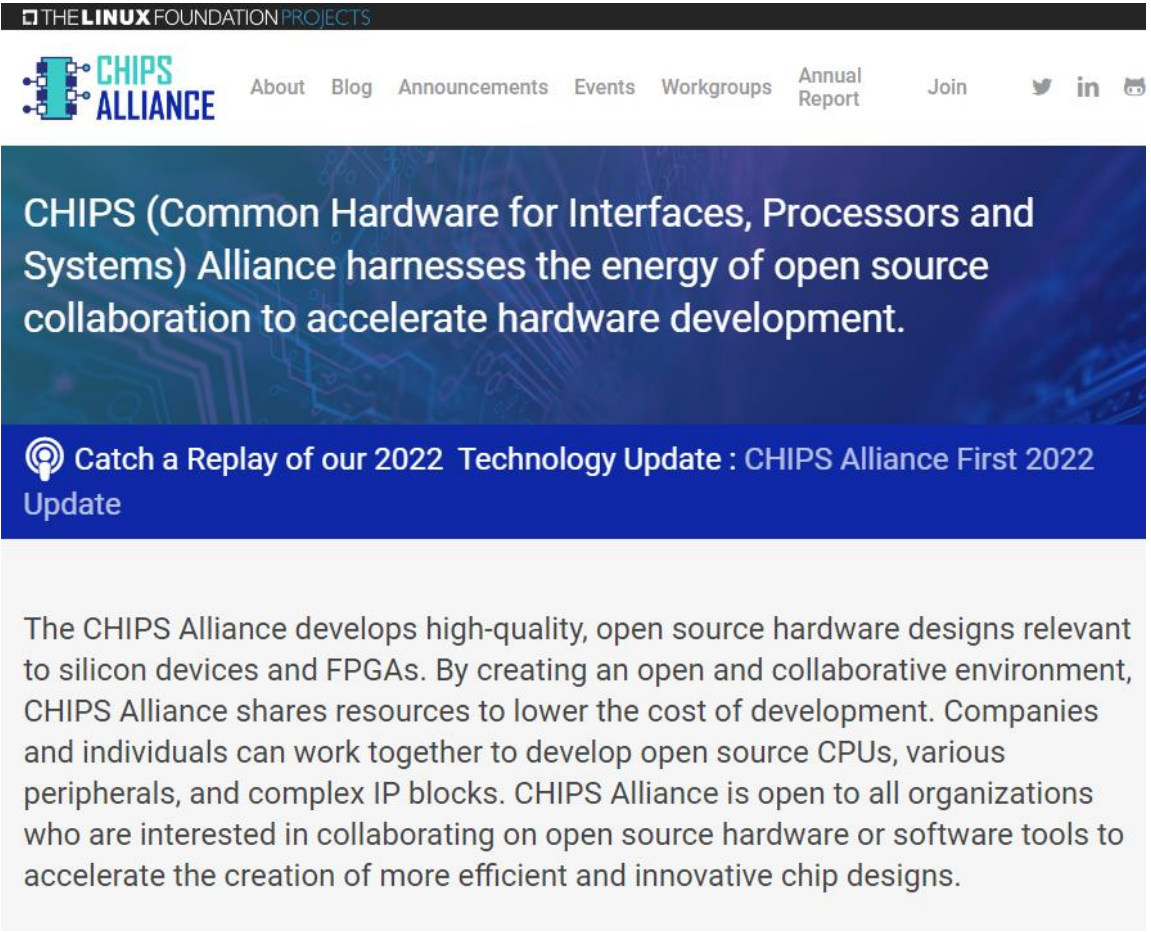
- Their website: https://hex-five.com/
- Their github: https://github.com/hex-five

4. Open community (14/21) Groups & companies

Some famous RISC-V groups & companies

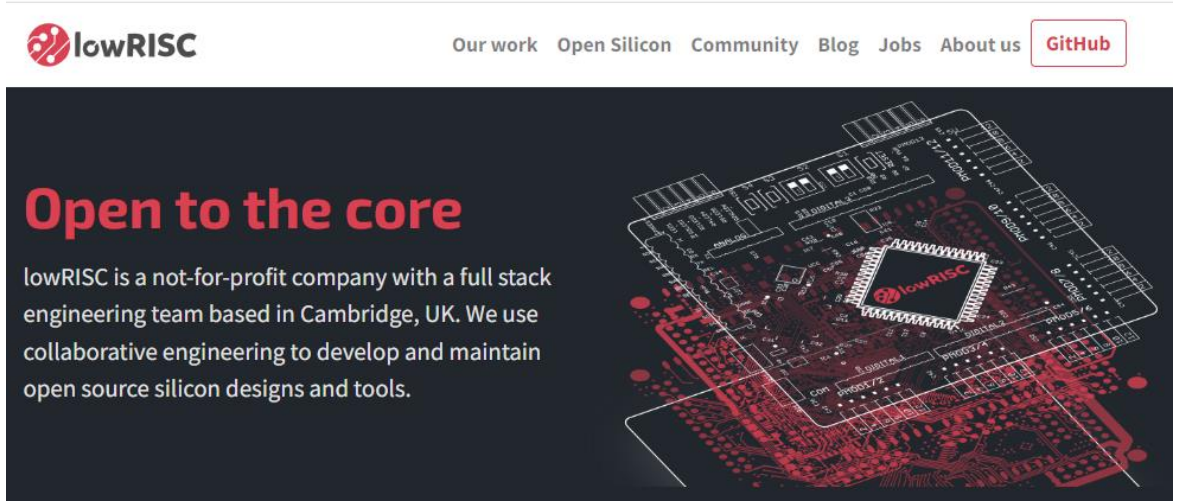
Chips Alliance is an organization that maintains many high-quality open-source IPs used in RISC-V systems:

- Their website: https://chipsalliance.org/
- Their github: https://github.com/chipsalliance



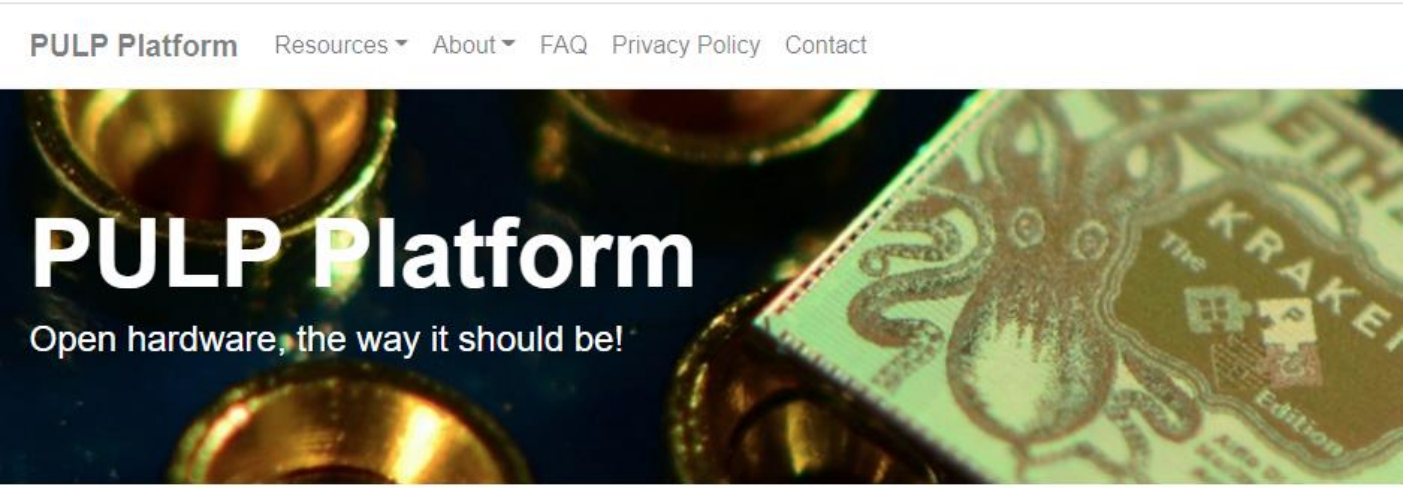
lowRISC is a non-profit company that has published many low-power and high-security IPs for RISC-V systems:

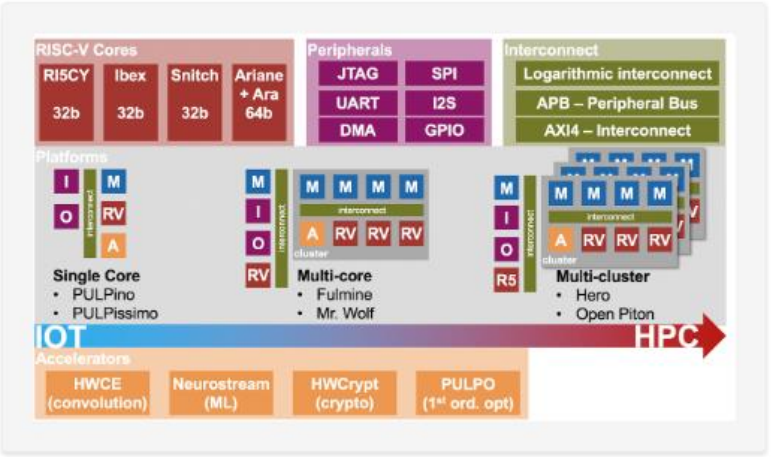
- Their website: https://lowrisc.org/
- Their github: https://github.com/lowRISC

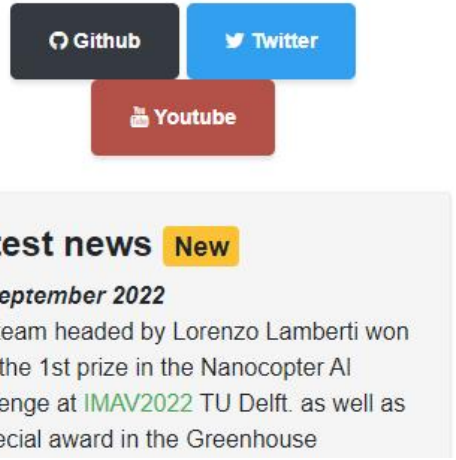


4. Open community (15/21) Groups & companies

Some famous RISC-V groups & companies







PULP is an open group started from a RISC-V project published by two universities.
It has many open-source processors and fabricated many chips:

- Their website: https://pulp-platform.org/
- Their github: https://github.com/pulp-platform

4. Open community (16/21) Products

Some RISC-V-related products

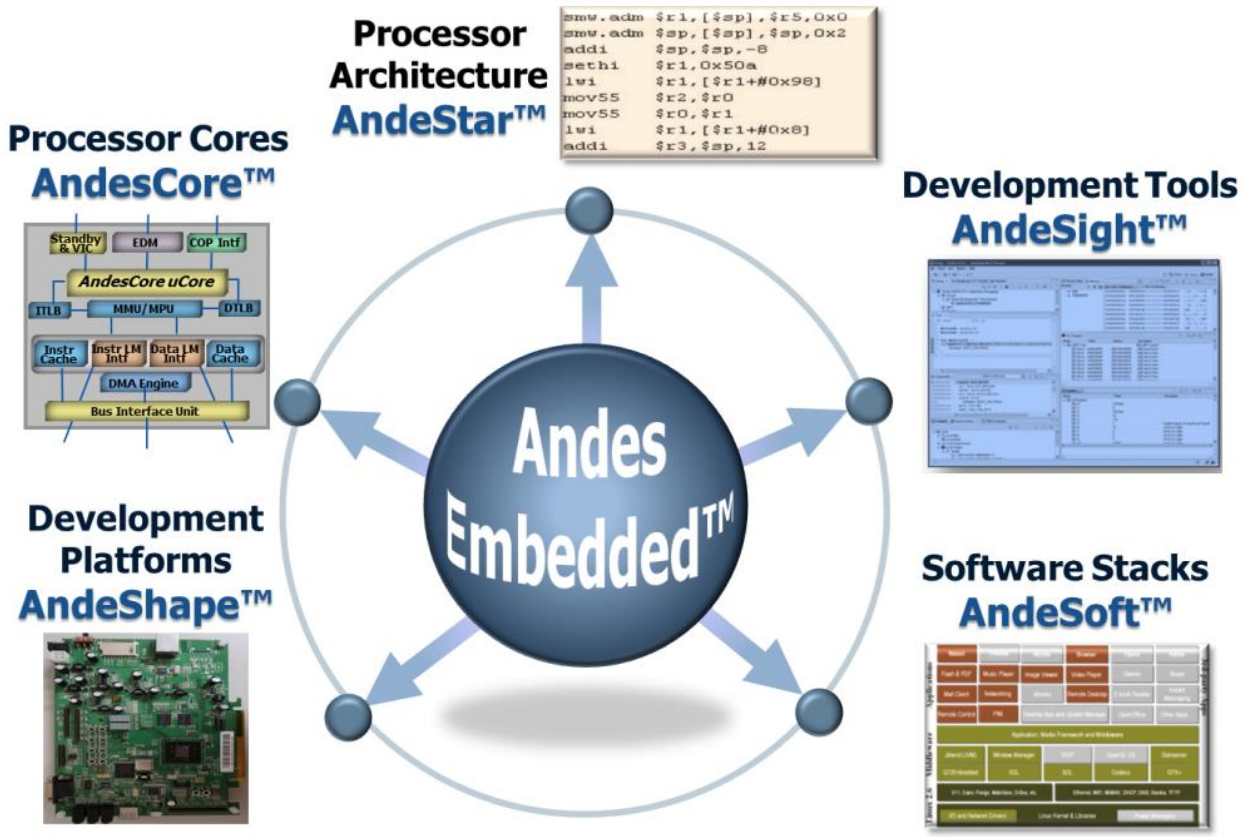
PolarFire SoC by **Microchip** is the first FPGA with a dedicated hard IP for RISC-V processors (Rocket core).

- FPGA: https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas
- Ecosystem (Mi-V): https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/mi-v
- Github page: https://github.com/polarfire-soc

4. Open community (17/21) Products

Some RISC-V-related products

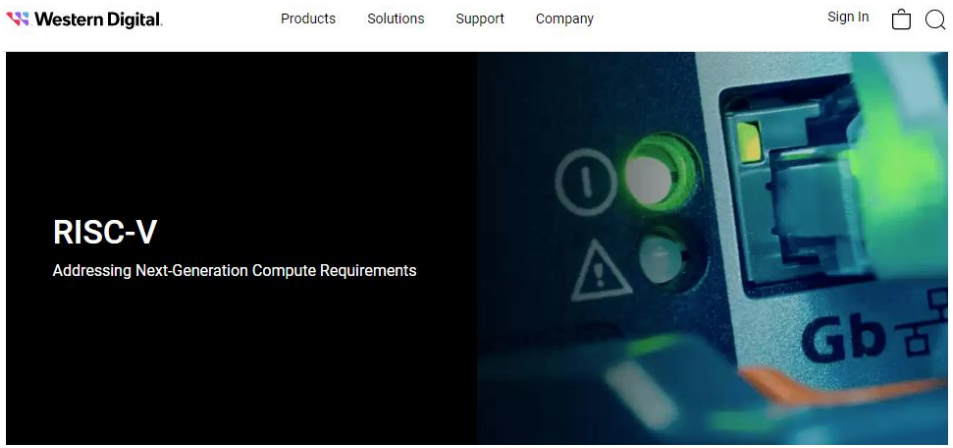
Andes Technology is a company that focuses on delivering commercial RISC-V products:



- Their website: https://www.andestech.com/en/risc-v-andes/
- Their Github page: https://github.com/andestech

4. Open community (18/21) Products

Some RISC-V-related products



Unleashing the power of data through RISC-V initiatives

The open source model, proven by the success of Linux®, now has a hardware platform in RISC-V to enable the next generation of innovation. Another reason is that the configurability RISC-V provides is unparalleled. By leveraging the open collaboration and flexibility of RISC-V, Western Digital can create processors that are purpose-built for data-centric applications.

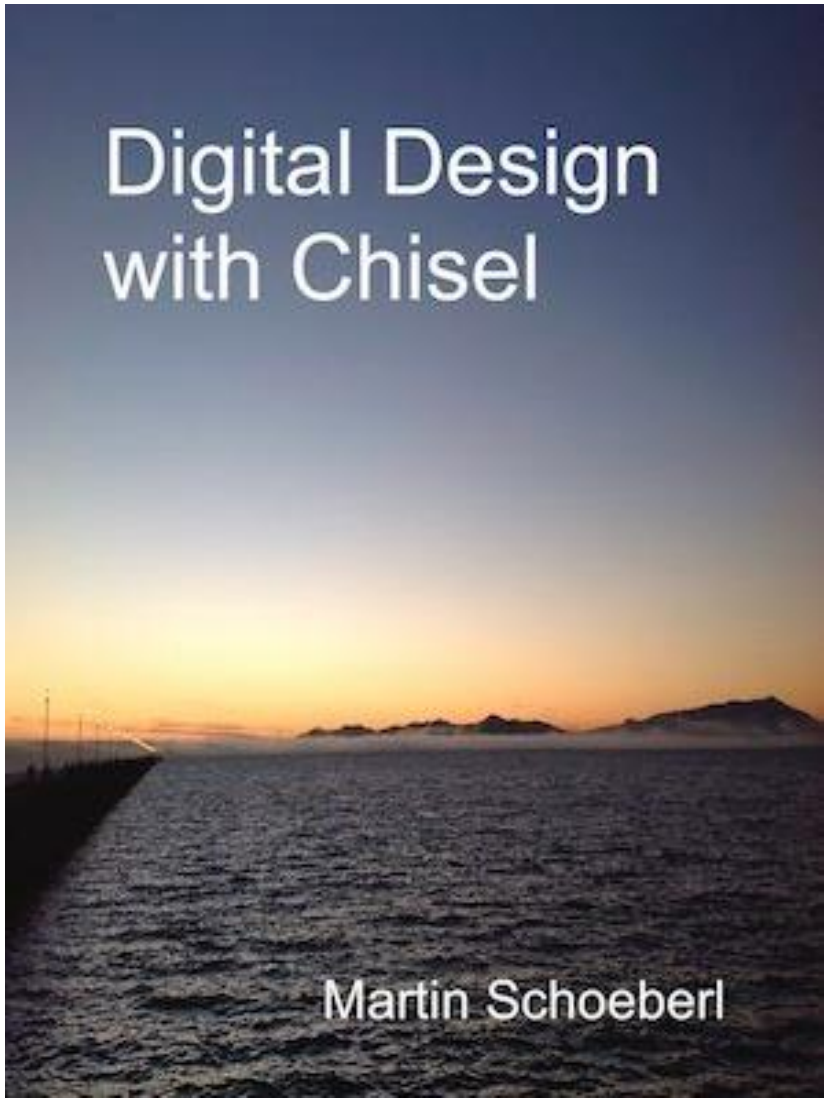
Promising upcoming products in **Western Digital** and **Seagate**

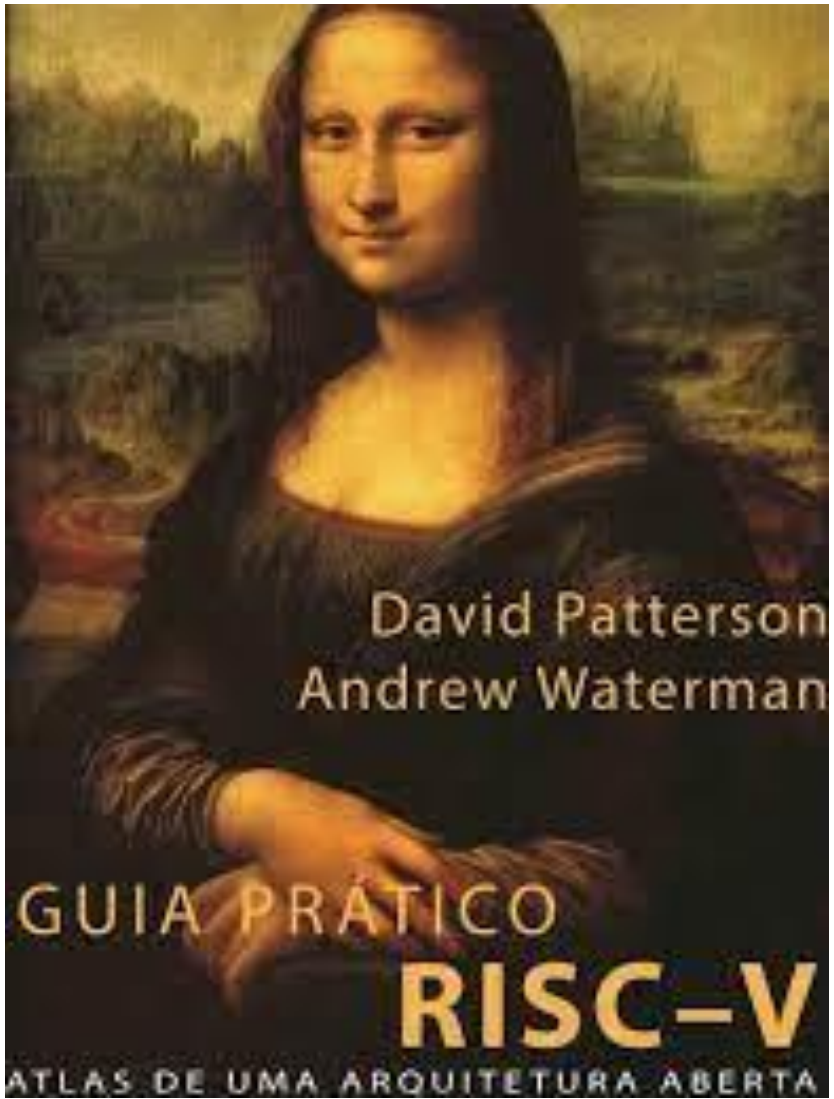
Read more:

- At RISC-V Foundation website: https://riscv.org/news/2020/12/seagate-western-digital-outline-progress-on-risc-v-designs-carol-sliwa-searchstorage-techtarget-com/
- At Seagate website: https://www.seagate.com/jp/ja/innovation/risc-v/
- At WD website: https://www.westerndigital.com/company/innovation/open-source/risc-v

4. Open community (19/21) Books

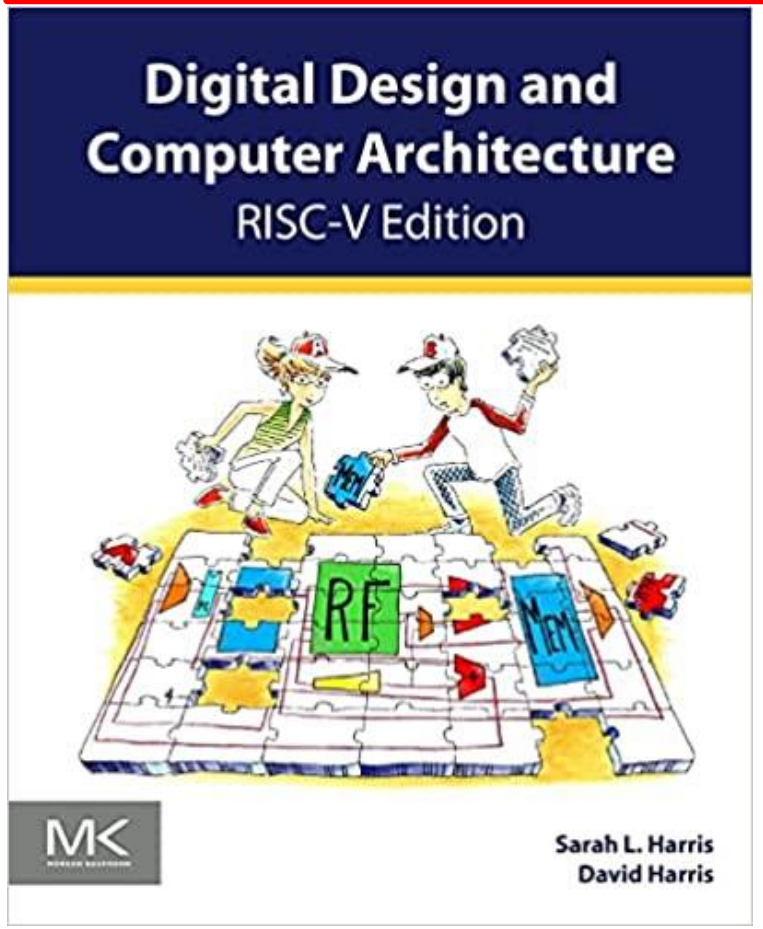
Two “must-have” books for RISC-V developers, from beginners to experts

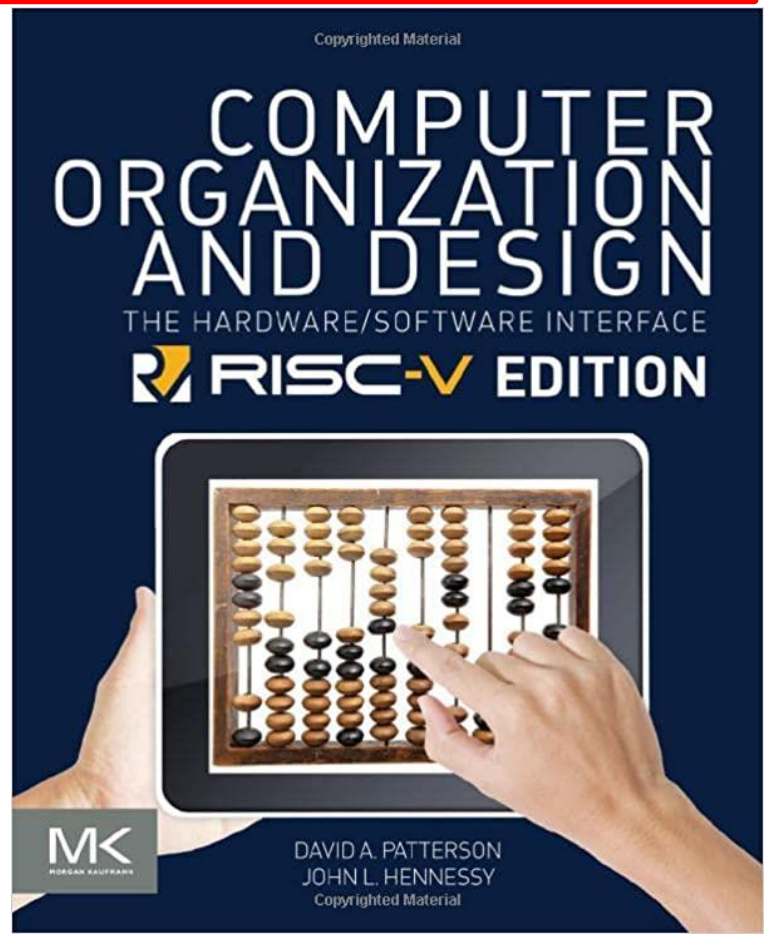




4. Open community (20/21) Books

RISC-V books that often used in universities for teaching



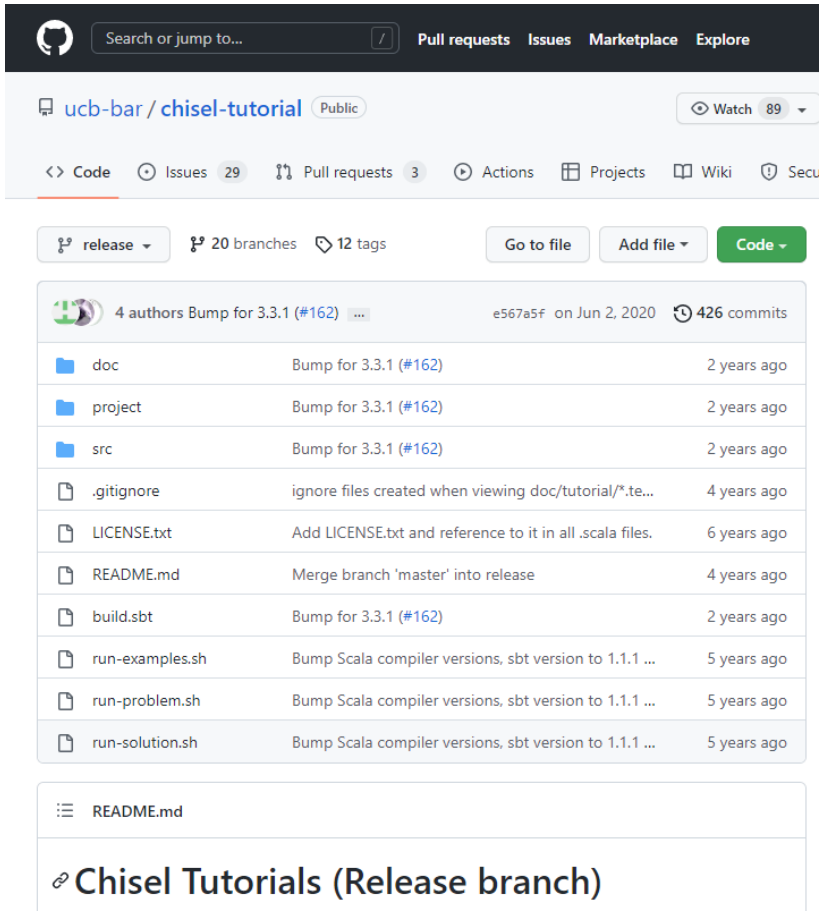


I have a book/material collection on google drive:
https://drive.google.com/drive/folders/1bAfqALmKGJLOcOlqiE51WB-SudU6-EY_?usp=sharing

4. **Open community** (21/21) Recommend for beginners

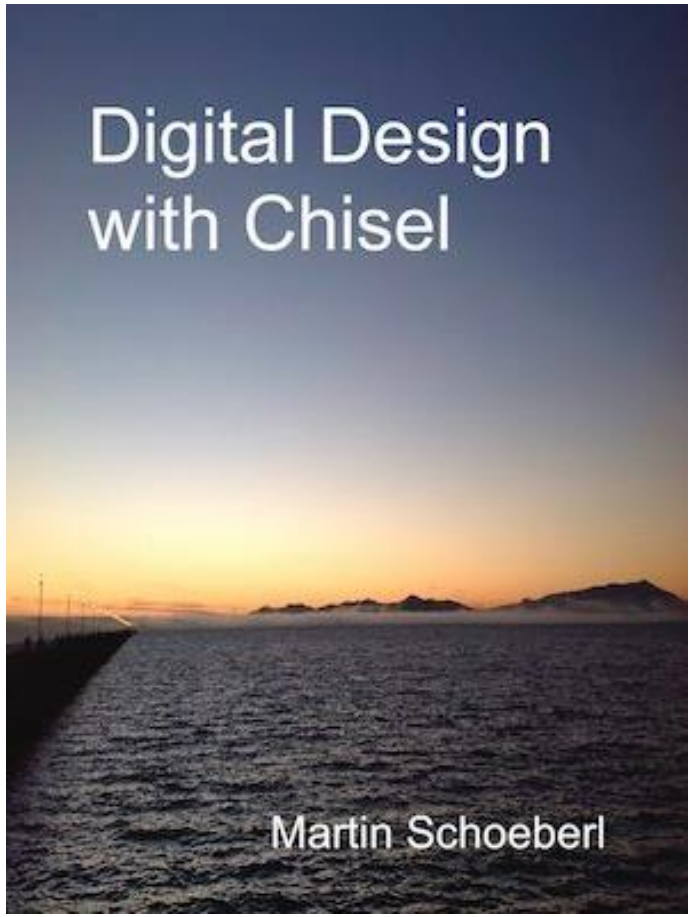
Beginners often started with CHISEL tutorial and specification.

CHISEL tutorials:

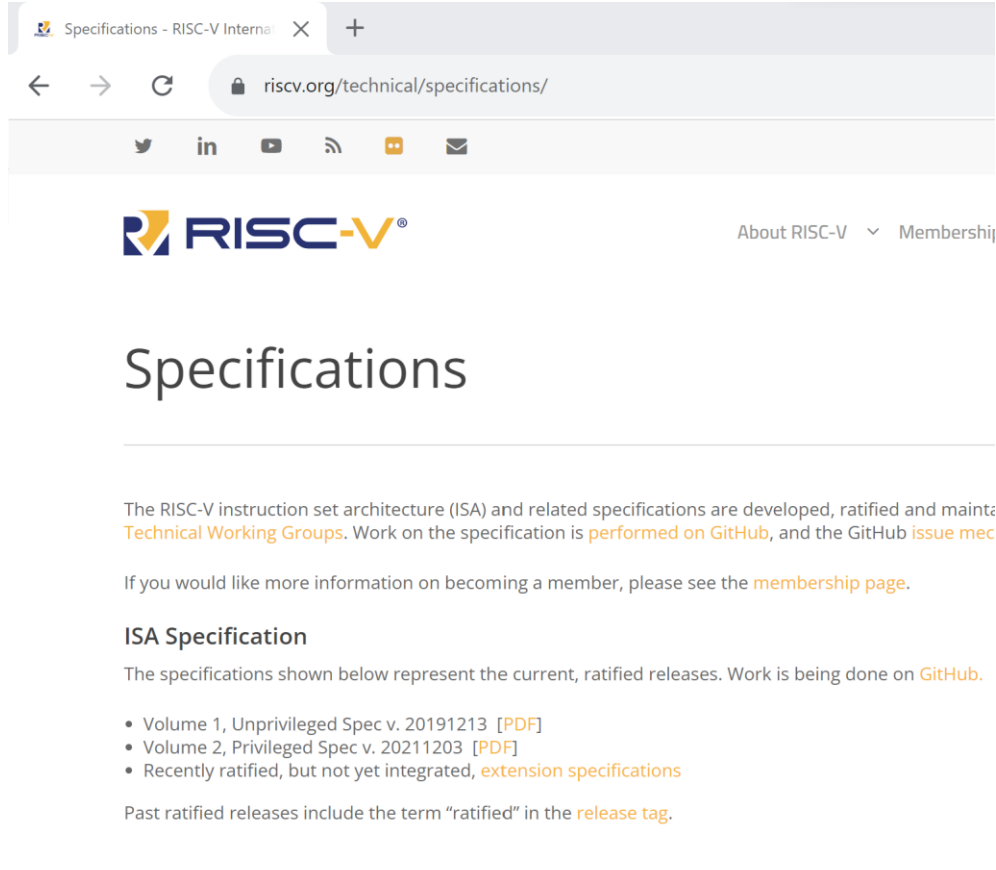


https://github.com/ucb-bar/chisel-tutorial

CHISEL coding:



RISC-V specification:



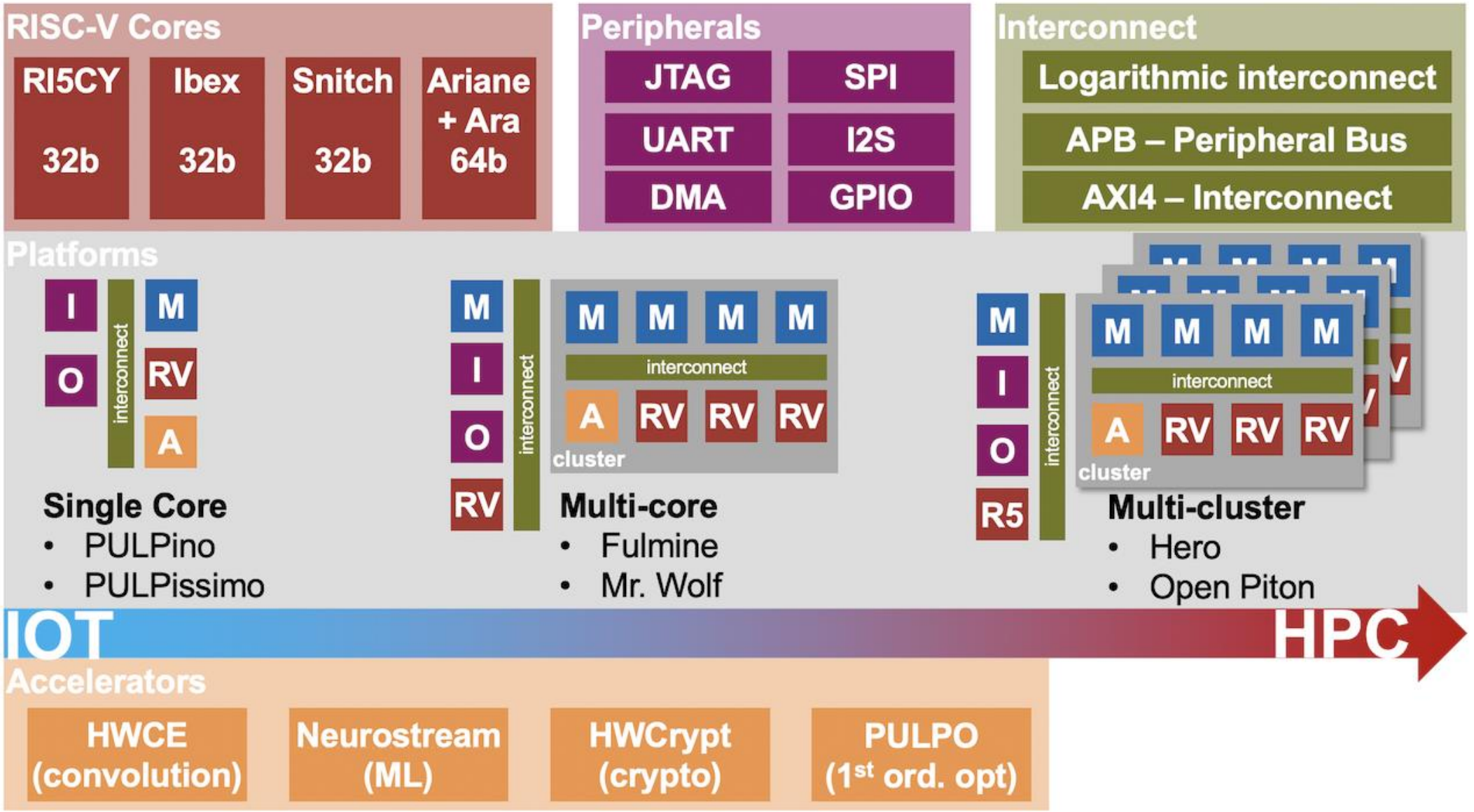
https://riscv.org/technical/specifications/

OUTLINE

1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

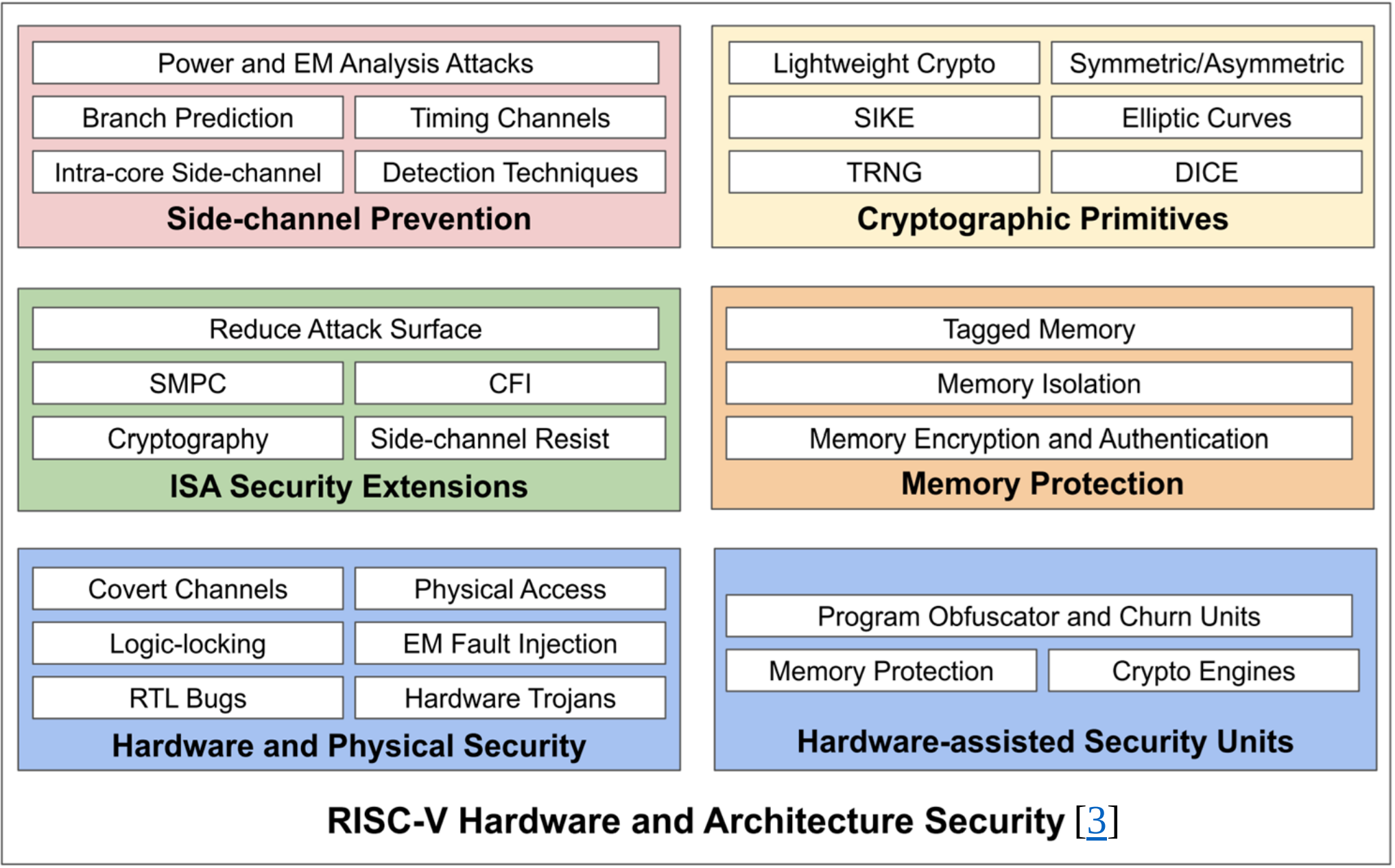
5. Main research @ UEC (1/30) Summary

- **Going low:** aim for low-power IoT
- **Going high:** aim for high-level cyber-security



(https://pulp-platform.org/img/main/pulp_family.png)

5. Main research @ UEC (2/30) Cyber-security overview



Cyber-security topics attract attention in the RISC-V community.

Main reasons:

- Opportunity to re-design from the bottom
- Price-sensitive applications
- Open approach
- Open community

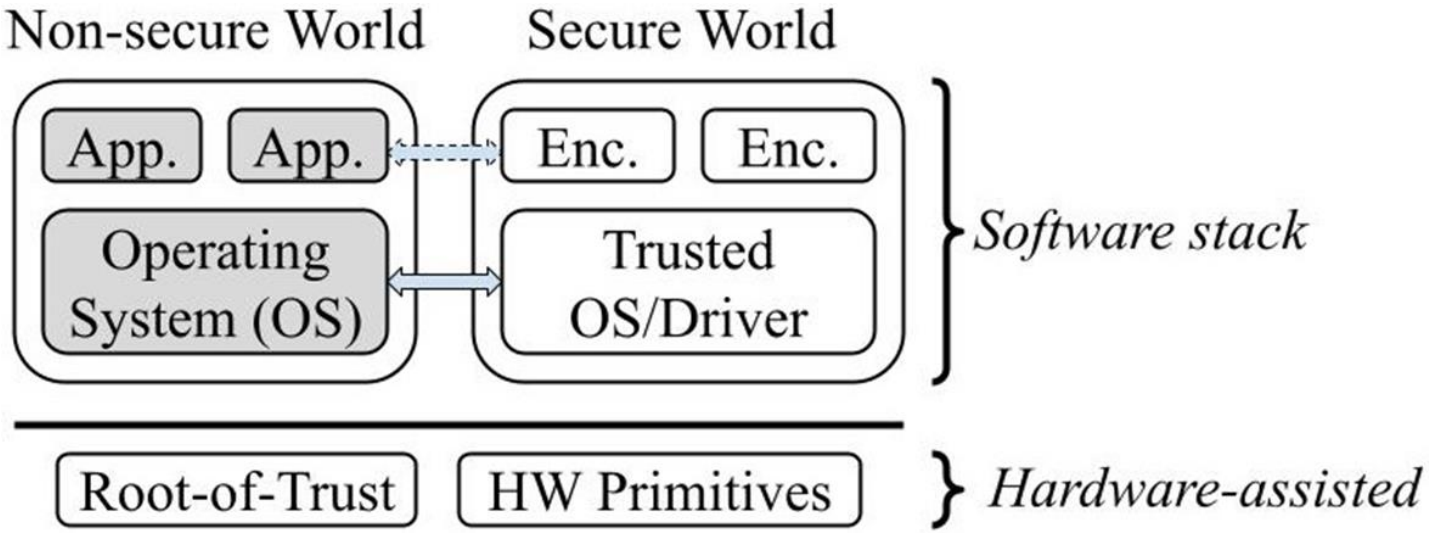
5. Main research @ UEC (3/30) TEE?

Trusted Execution Environment (TEE) provides:

1. *Integrity:* the code and data cannot be tampered.
2. *Confidentiality:* the application's content cannot be read.
3. *Attestation:* proof to a remote party that the system is safe.

A typical TEE setup:

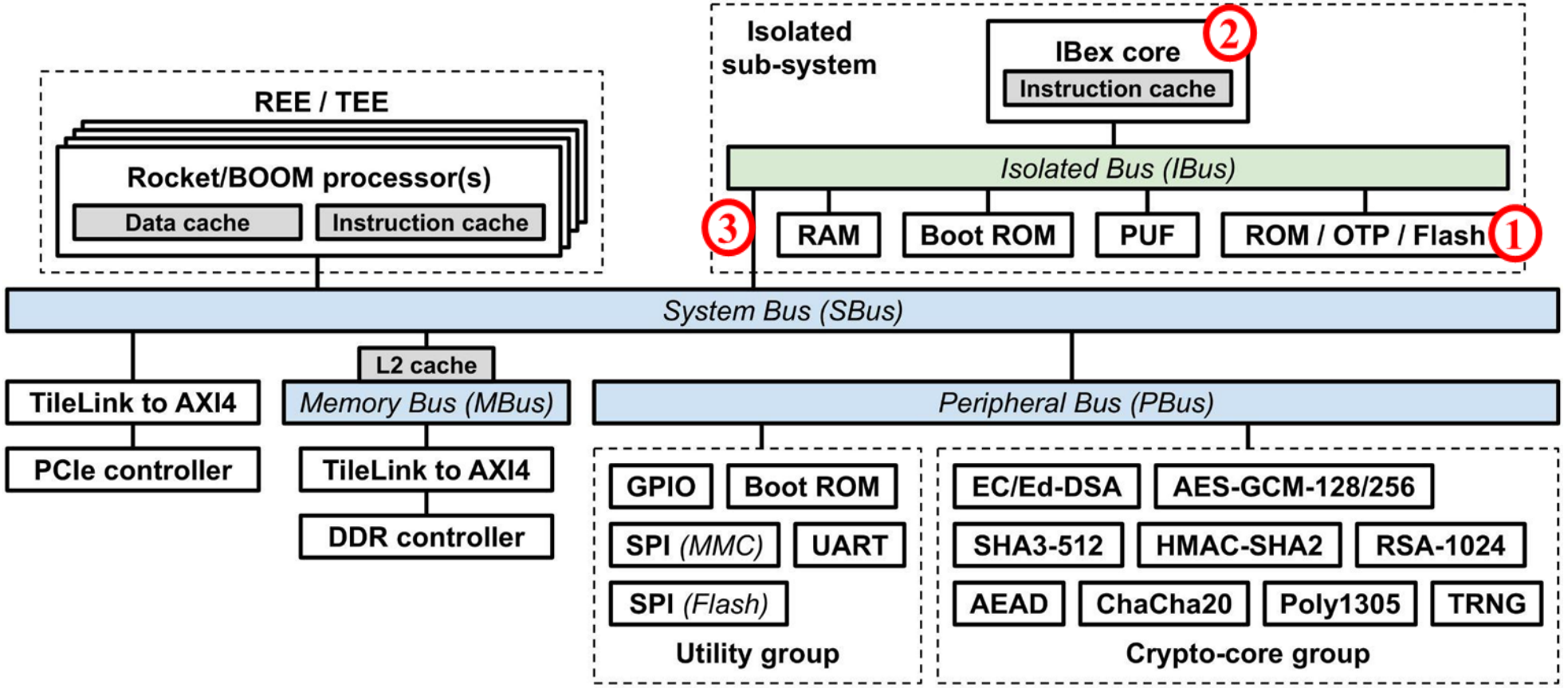
- Secure (trusted) vs. non-secure (untrusted) worlds.
- Barrier enforcer by: software *AND* hardware.
- All TEEs need some sort of hardware-assisted modules: Root-of-Trust (RoT) and primitives.



- HW primitives *(examples)*: cache flushing, cache partitioning, memory isolation, memory encryption, keys management, bus access controller, enclave encryption, and so on.

5. Main research @ UEC (4/30) RoT for TEE

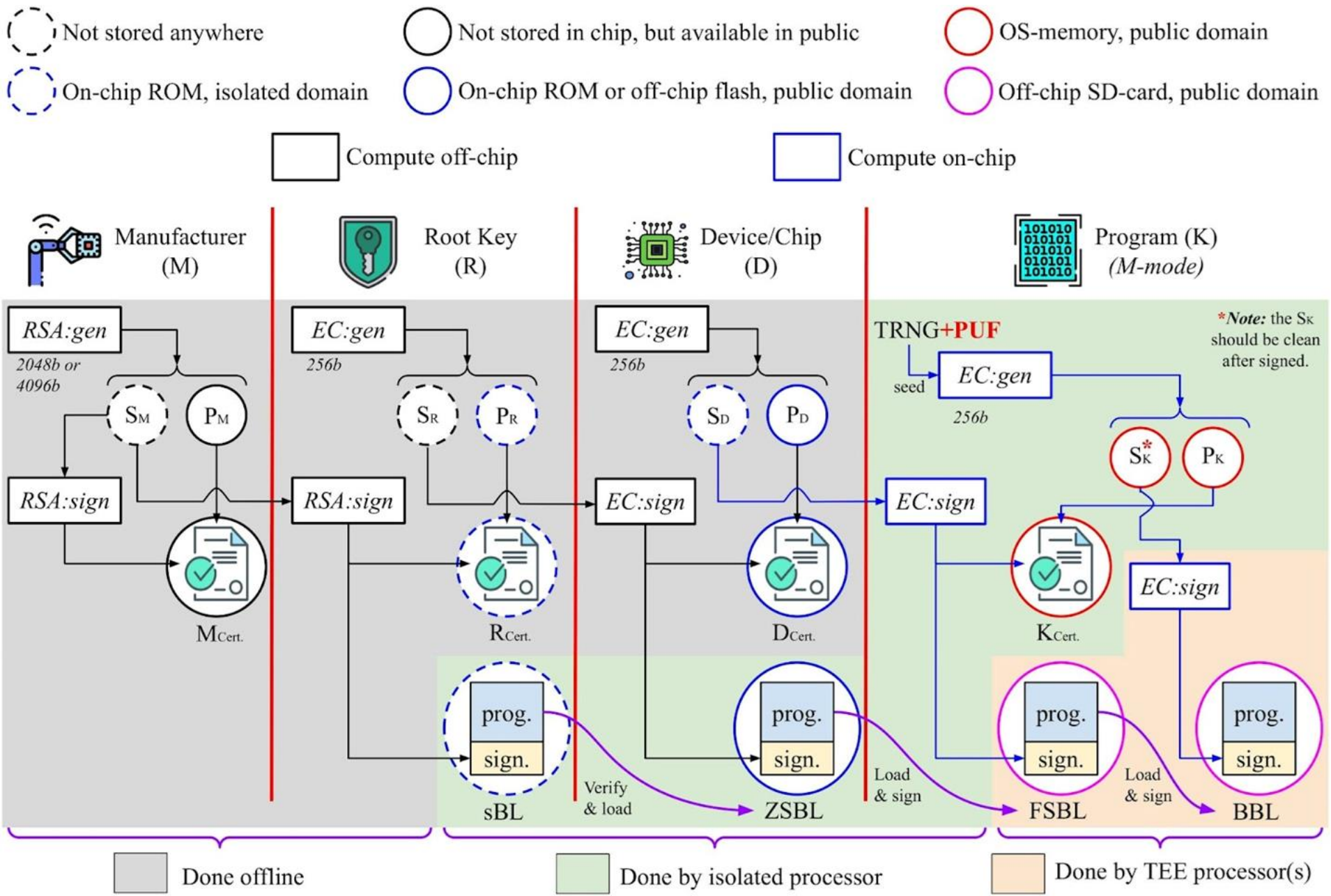
A secure boot process with Root-of-Trust (RoT) for TEE



Main points of the proposed architecture:

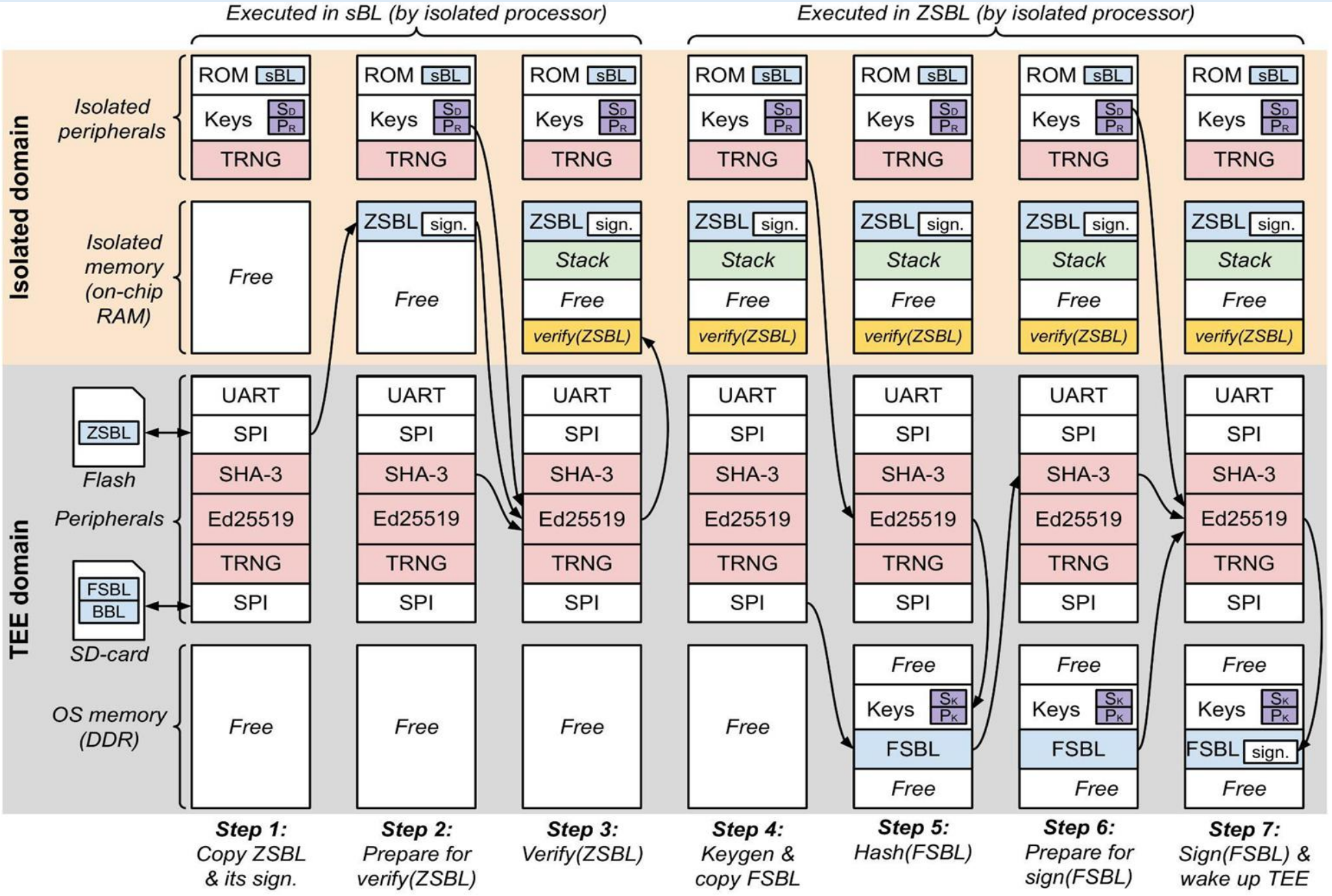
1. Root key installed at the time manufactured.
2. Hidden MCU for the flexible boot program
3. Hierarchy-bus: TEE processors cannot access RAM/ROMs in the isolated domain *(BUT the isolated core can access ALL)*

5. Main research @ UEC (5/30) RoT for TEE



The proposed keys scheduling scheme.

5. Main research @ UEC (6/30) RoT for TEE



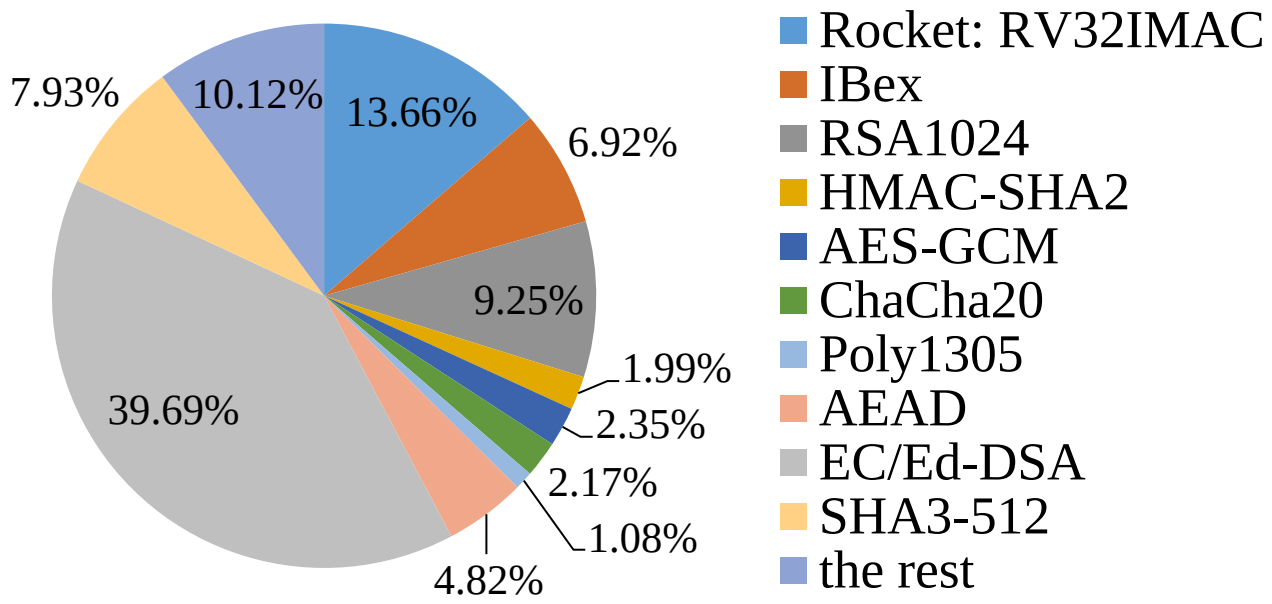
The proposed boot sequence.

5. Main research @ UEC (7/30) RoT for TEE

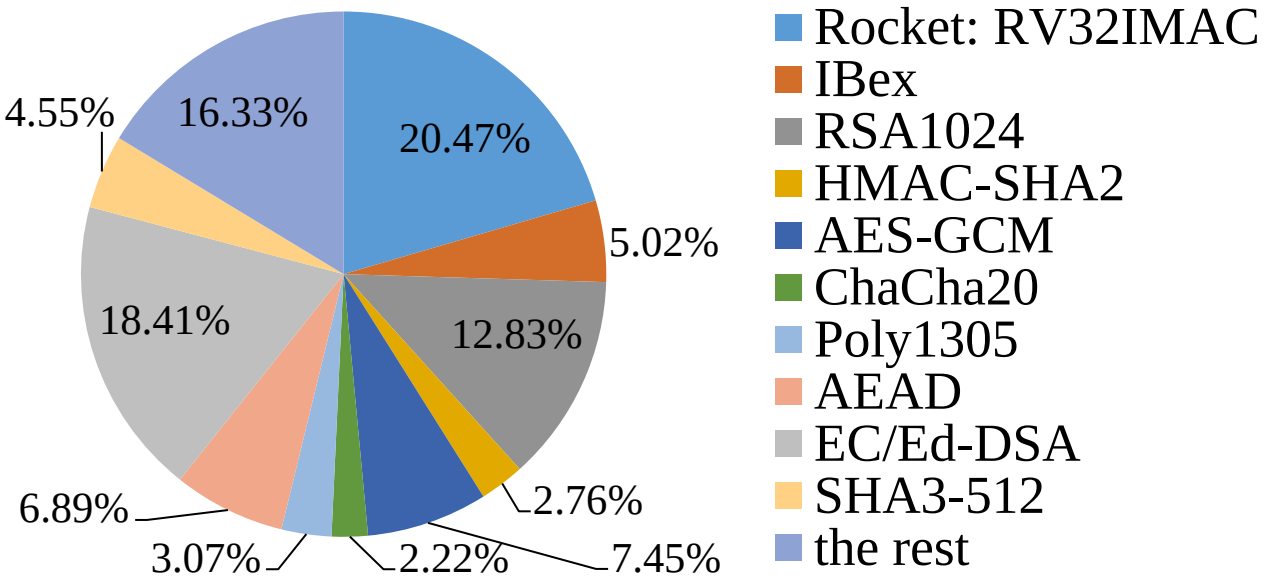
SoC resources utilization pie chart:
All crypto-cores + Single-core RV32IMAC Rocket

VC707 FPGA (Virtex-7) build report

LUT Ultilization



Register Utilization



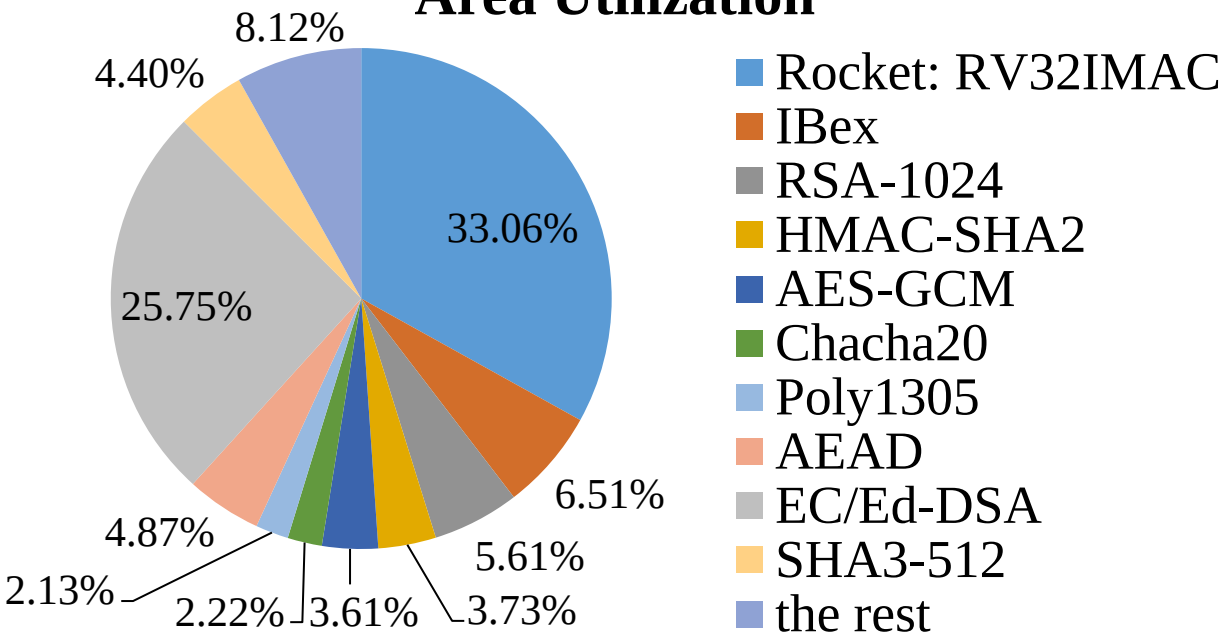
***Note**: "*the rest*" means all the buses, TRNG and utility-group peripherals such as GPIO, SPI, boot ROM, etc.

5. Main research @ UEC (8/30) RoT for TEE

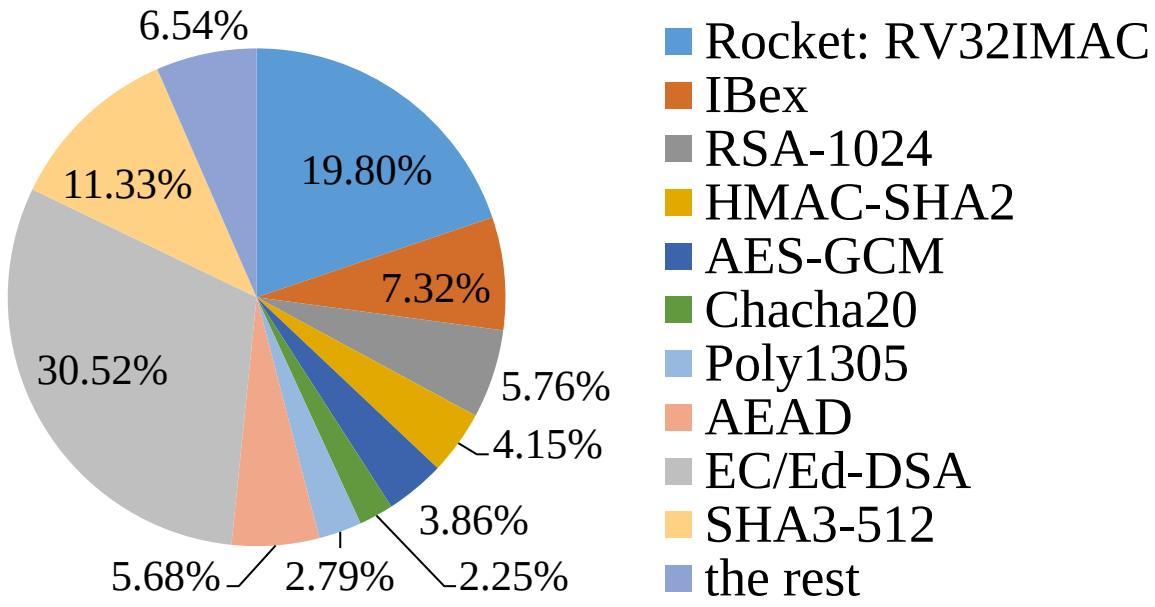
SoC resources utilization pie chart:
All crypto-cores + Single-core RV32IMAC Rocket

ROHM-180nm chip build report

Area Utilization



Power Utilization



***Note**: "*the rest*" means all the buses, TRNG and utility-group peripherals such as GPIO, SPI, boot ROM, etc.

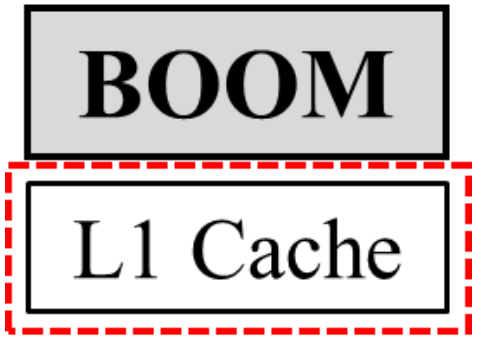
5. Main research @ UEC (9/30) Spectre attack?

Spectre: a Cache side-channel attack

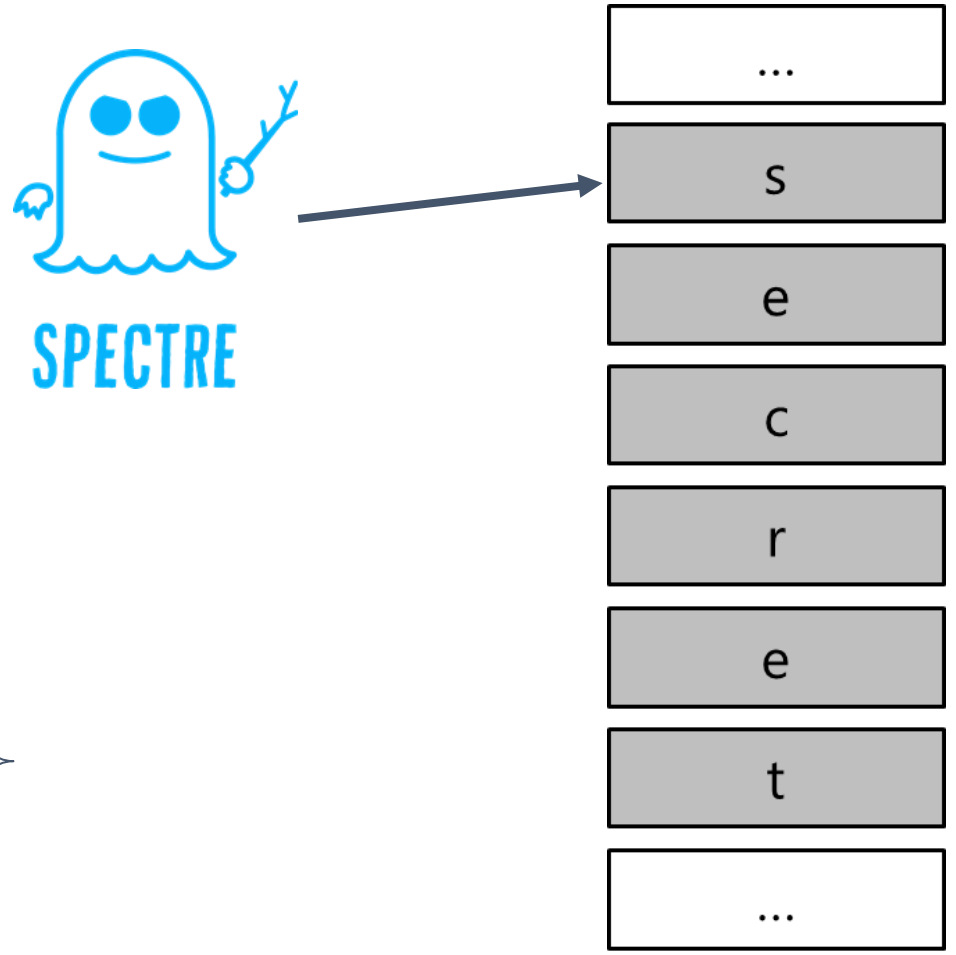
Target: RISC-V Out-of-order (ex: BOOM)

First variants:

- Spectre v1: Bound Check Bypass
- Spectre v2: Branch Target Injection



- Branch Predictor Unit
- Speculative Execution
- Caching
- …



Cache memory

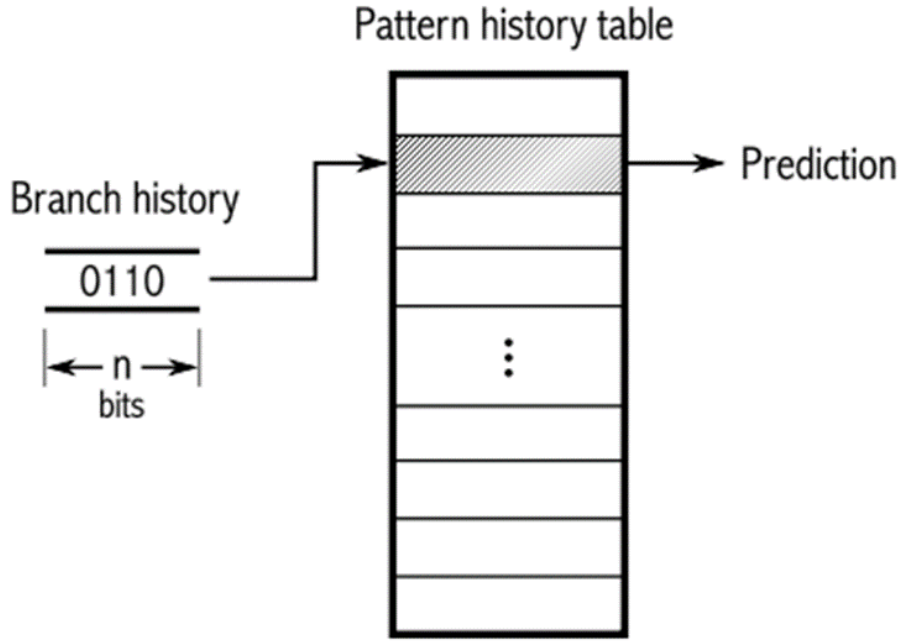
BOOM suitable for Spectre

5. Main research @ UEC (10/30) Spectre attack?

Speculative execution example:

IF (a < 10) Run B

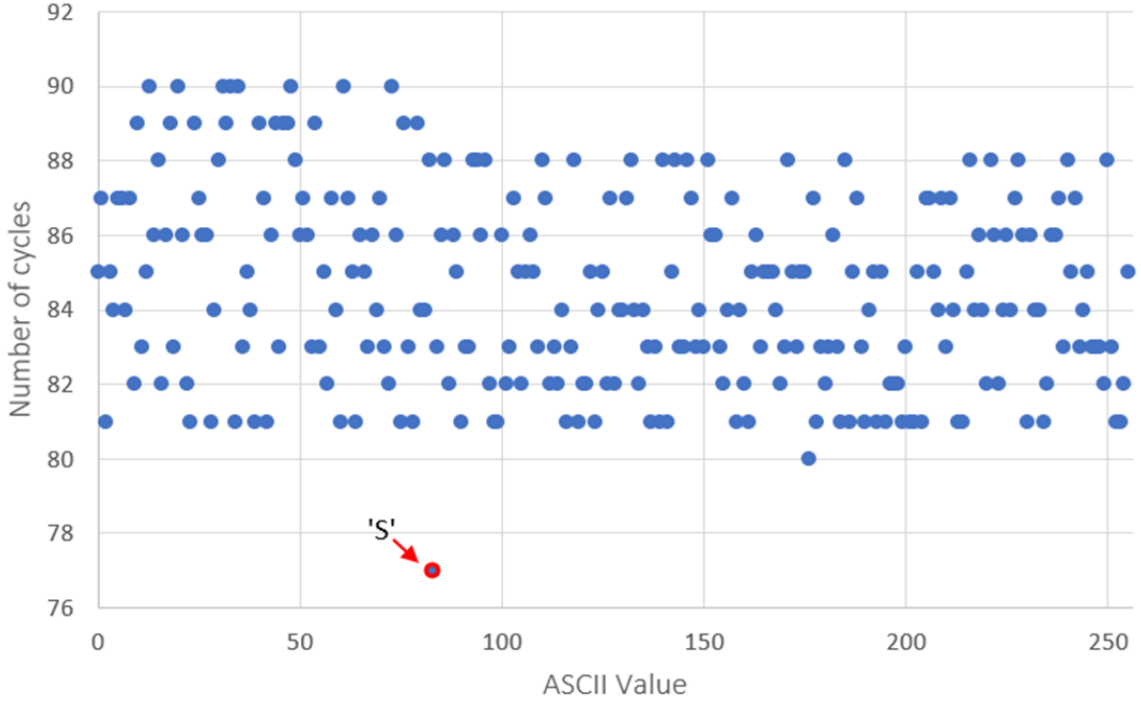
User input	Process
a = 1	**TRUE** => Execute **B**
a = 2	**TRUE** => Execute **B**
a = 3	**TRUE** => Execute **B**
…	**TRUE** => Execute **B**
a = X	*Guess as* **TRUE** => Execute **B**



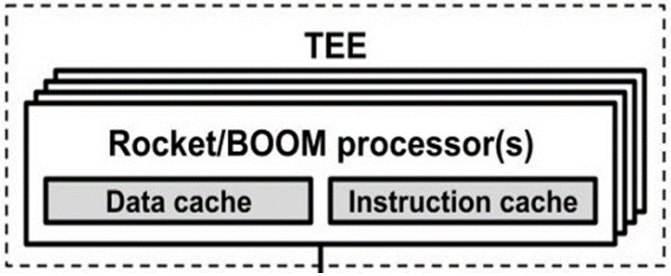
5. Main research @ UEC (11/30) Spectre attack?

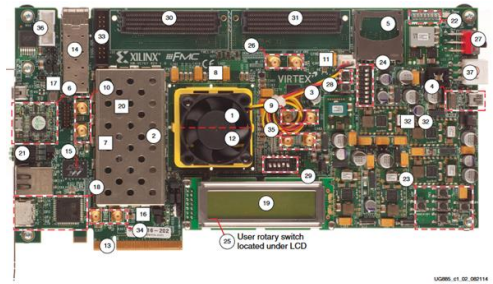
Implement RISC-V processor

- BOOM (out-of-order) core: exploited
- Rocket (in-of-order) core: cannot exploited



Observe cache accessing time after an attack attempt



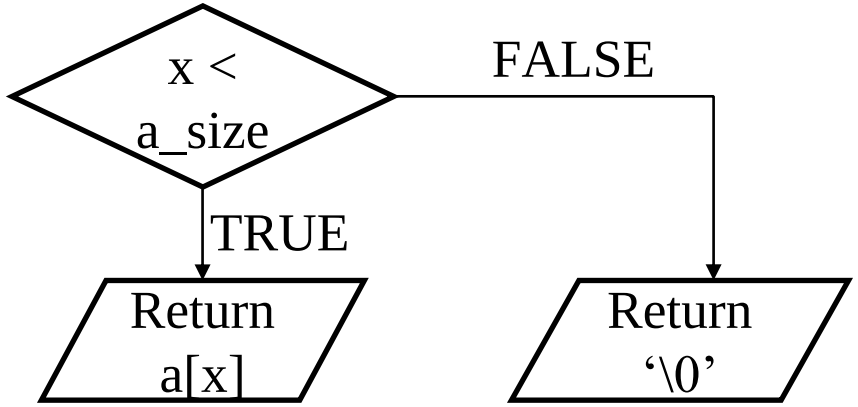


FPGA VC707

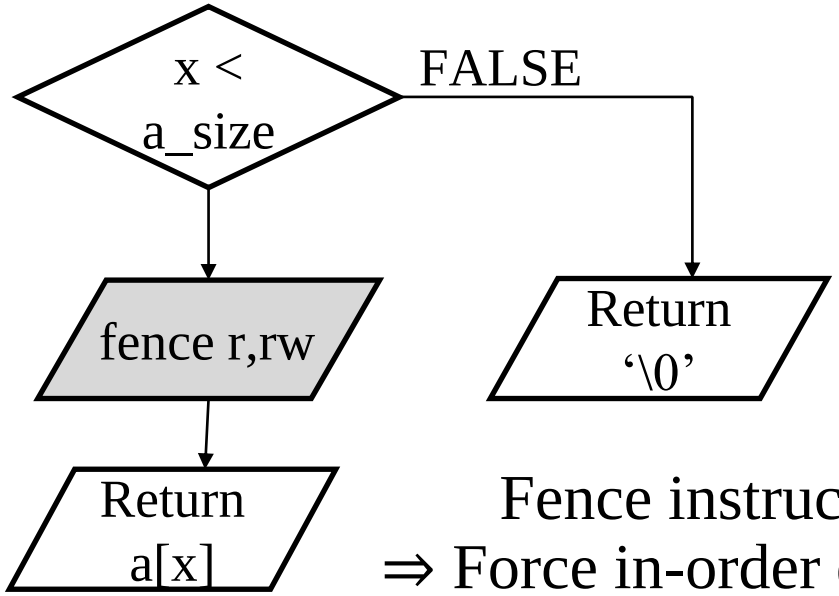
```
char(S) | guess_char(hits,score,value) 1.(3, 83, S)
char(e) | guess_char(hits,score,value) 1.(9, 101, e)
char(c) | guess_char(hits,score,value) 1.(7, 99, c)
char(r) | guess_char(hits,score,value) 1.(8, 114, r)
char(e) | guess_char(hits,score,value) 1.(8, 101, e)
char(t) | guess_char(hits,score,value) 1.(9, 116, t)
char( ) | guess_char(hits,score,value) 1.(10, 32, )
char(K) | guess_char(hits,score,value) 1.(8, 75, K)
char(e) | guess_char(hits,score,value) 1.(8, 101, e)
char(y) | guess_char(hits,score,value) 1.(8, 121, y)
```

Attack log (success case)

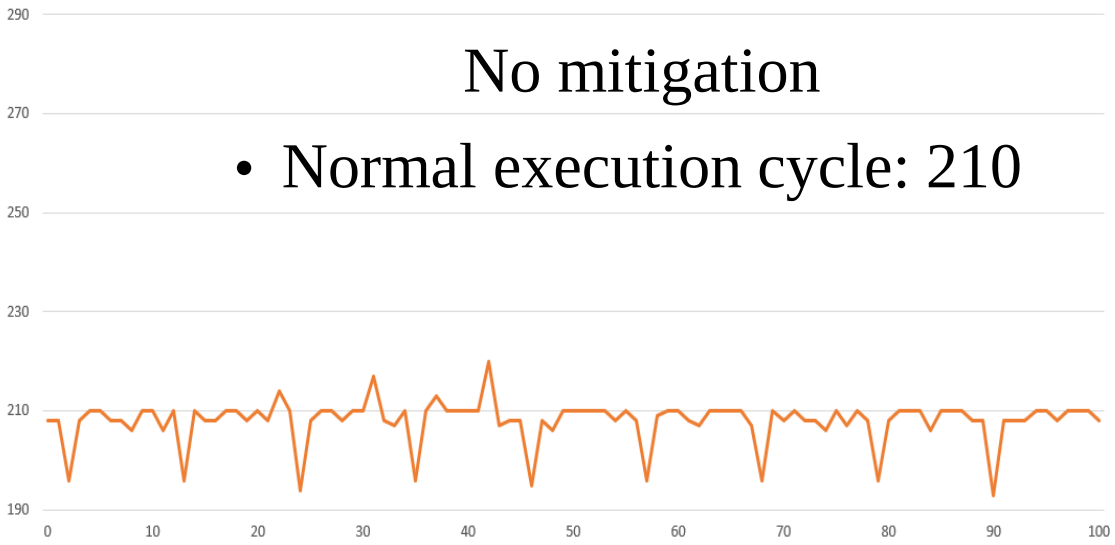
5. Main research @ UEC (12/30) Spectre software mitigate



Original code

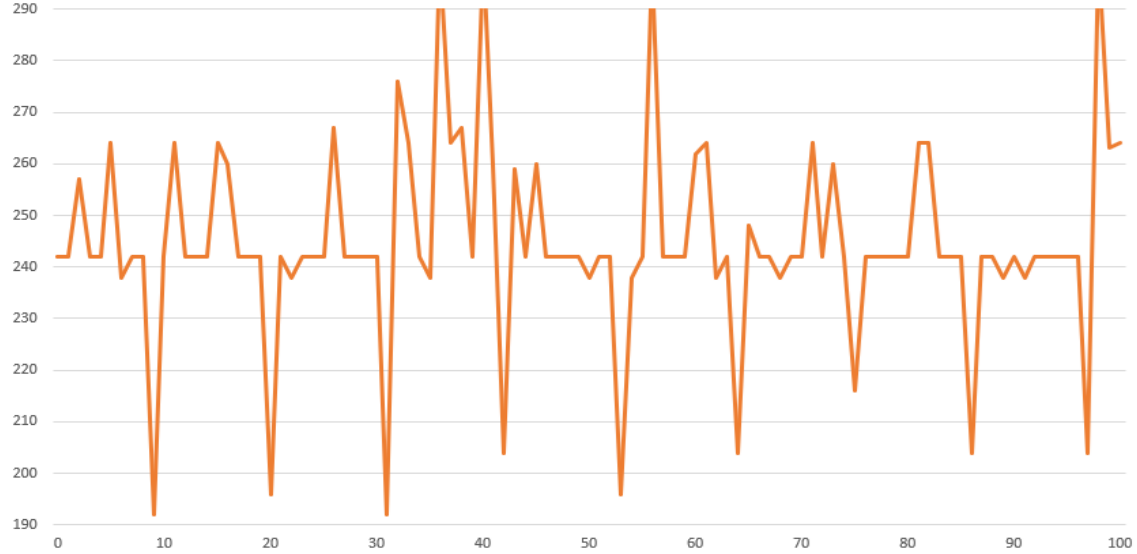


Fence instructions
⇒ Force in-order execution



No mitigation

- Normal execution cycle: 210



- Performance loss: 15 – 43% (242 – 290 cycles)

5. Main research @ UEC (13/30) Spectre hardware mitigate

Hardware mitigation method: modifying MSHRs

- MSHRs: miss status holding registers
- Located in The Load/Store Unit (LSU)
- Handling data forwarding when mis-speculative events

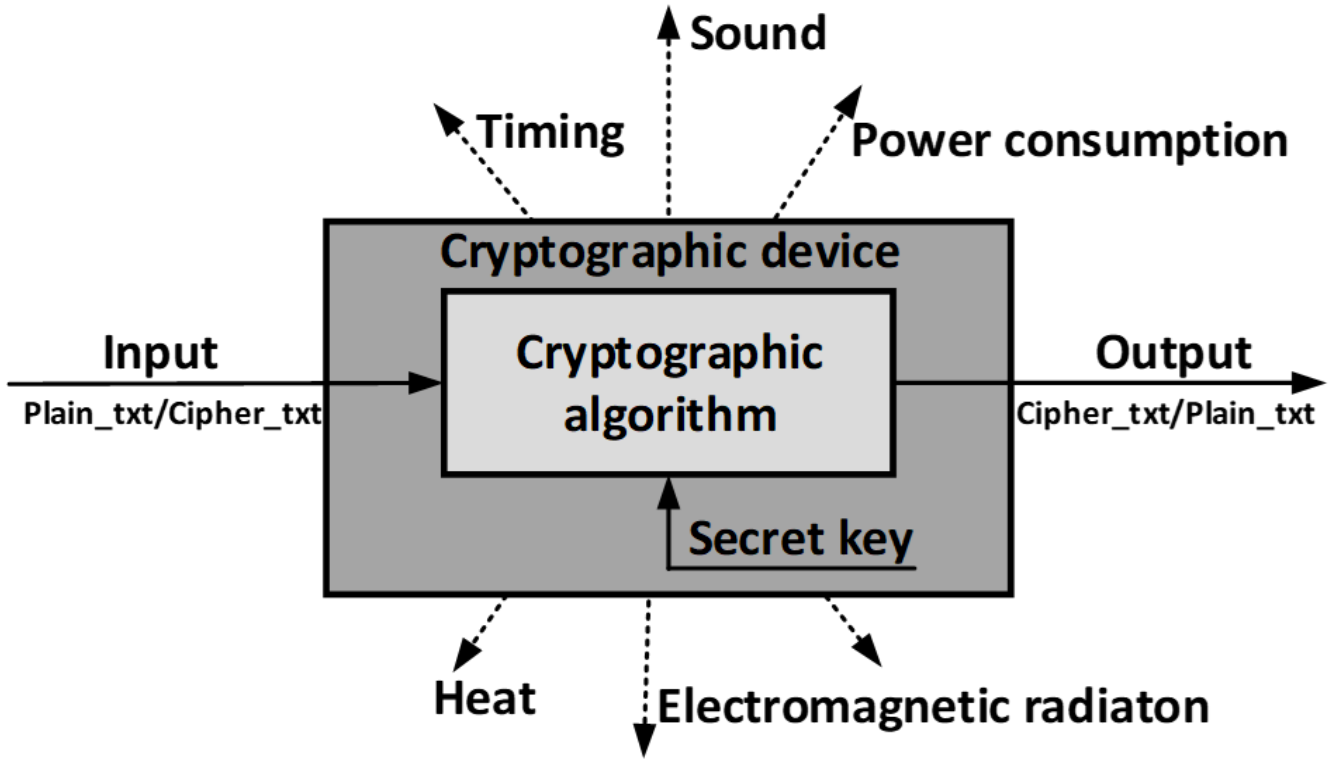
⇒ Delay the data forwarding when mis-speculative

MSHRs resources utilization

Configuration	LUT	FF
Normal MSHR	1926	1120
Secure MSHR	1980 (2.8%)	1124 (0.4%)

5. Main research @ UEC (14/30) CPA?

A cryptographic device leaks side-channel information



Side-channel attacks:

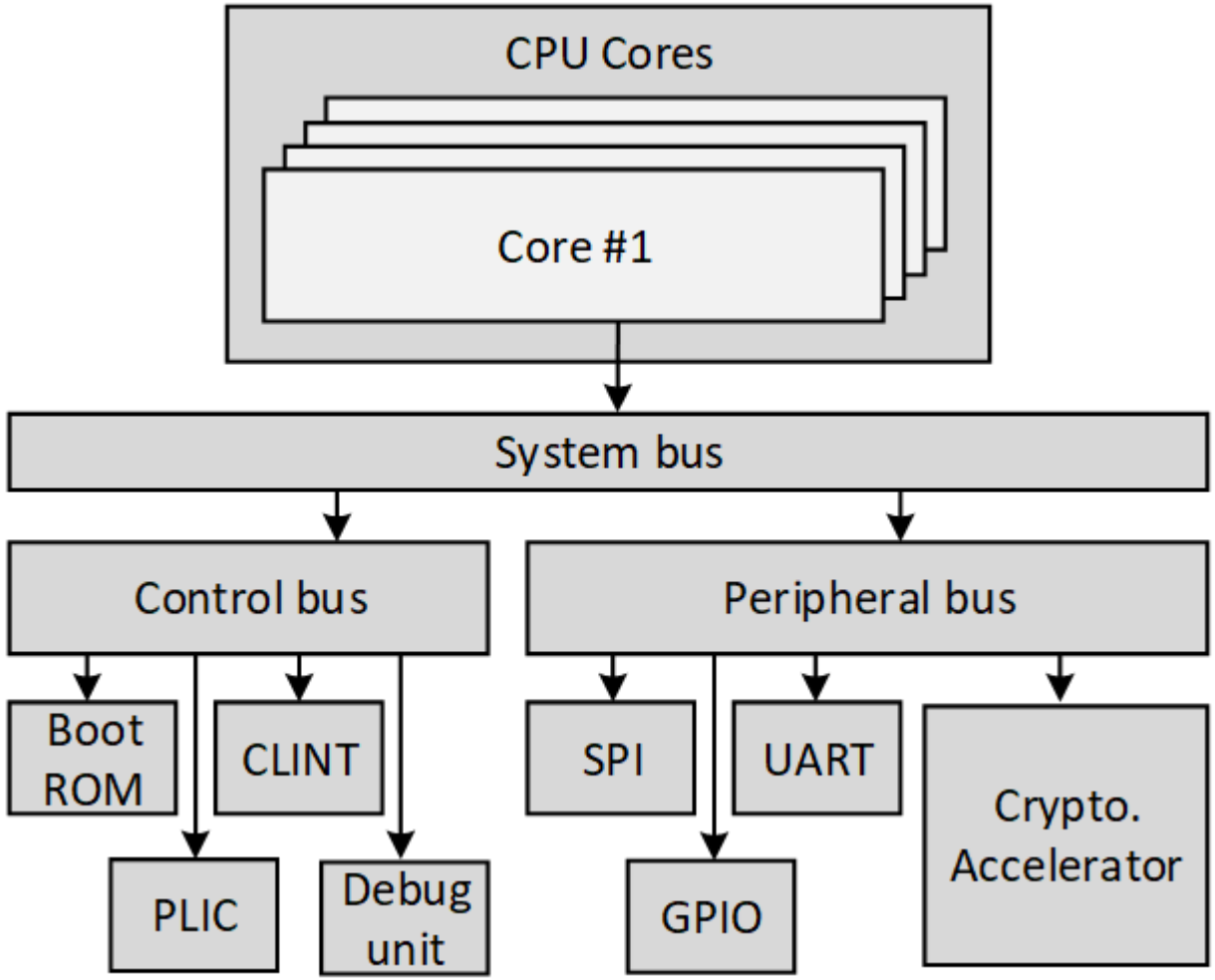
Exploit unavoidable side-channel information in cryptanalysis.

Correlation Power Analysis (CPA) attacks:

Using Power consumption or Electromagnetic radiation.

5. Main research @ UEC (15/30) CPA countermeasure

Example of Cryptographic SoC



Countermeasures for Cryptographic SoC:

Existing techniques are not suitable:

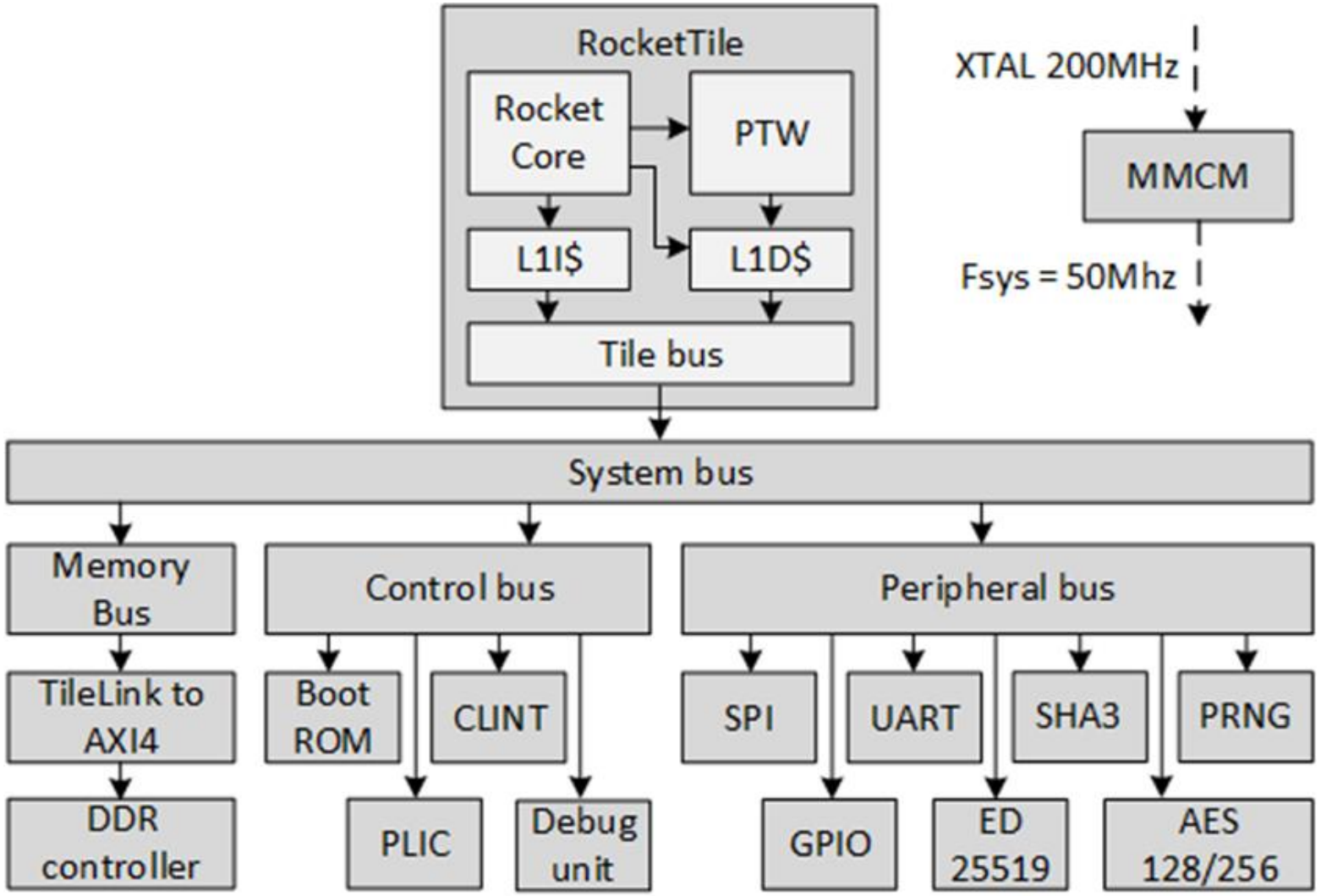
- Masking: Reduce performance, Increase power, area.
- Hiding: Huge hardware overheads.

⇒ **Proposed Ideas:**

- Randomly scale the clock freq. of Crypto.Acc. after each encryption/decryption.
- Only applied to the Crypto.Acc.
- Create as many Clock frequencies as possible.

5. Main research @ UEC (16/30) CPA countermeasure

Unprotected Cryptographic SoC (TEE-Hardware)

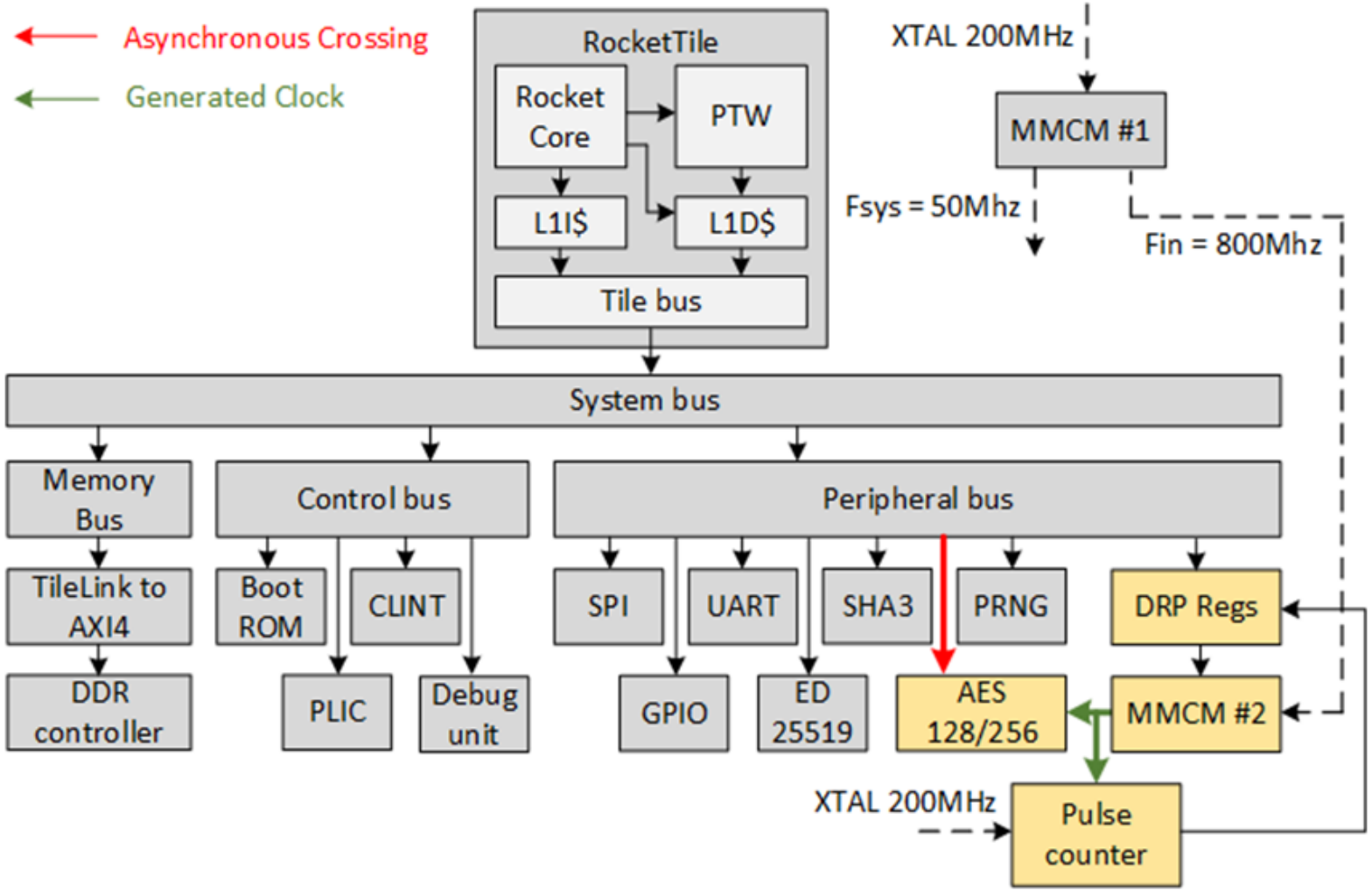


Unprotected SoC (TEE-hardware):

- 32-bit RISC-V SoC
- DDR Controller ⇒ support Linux OS
- Crypto. Accelerator:
  - AES-128/256
  - SHA3
  - ED25519
  - PRNG
- Fixed system Clock: Fsys = 50MHz

5. Main research @ UEC (17/30) CPA countermeasure

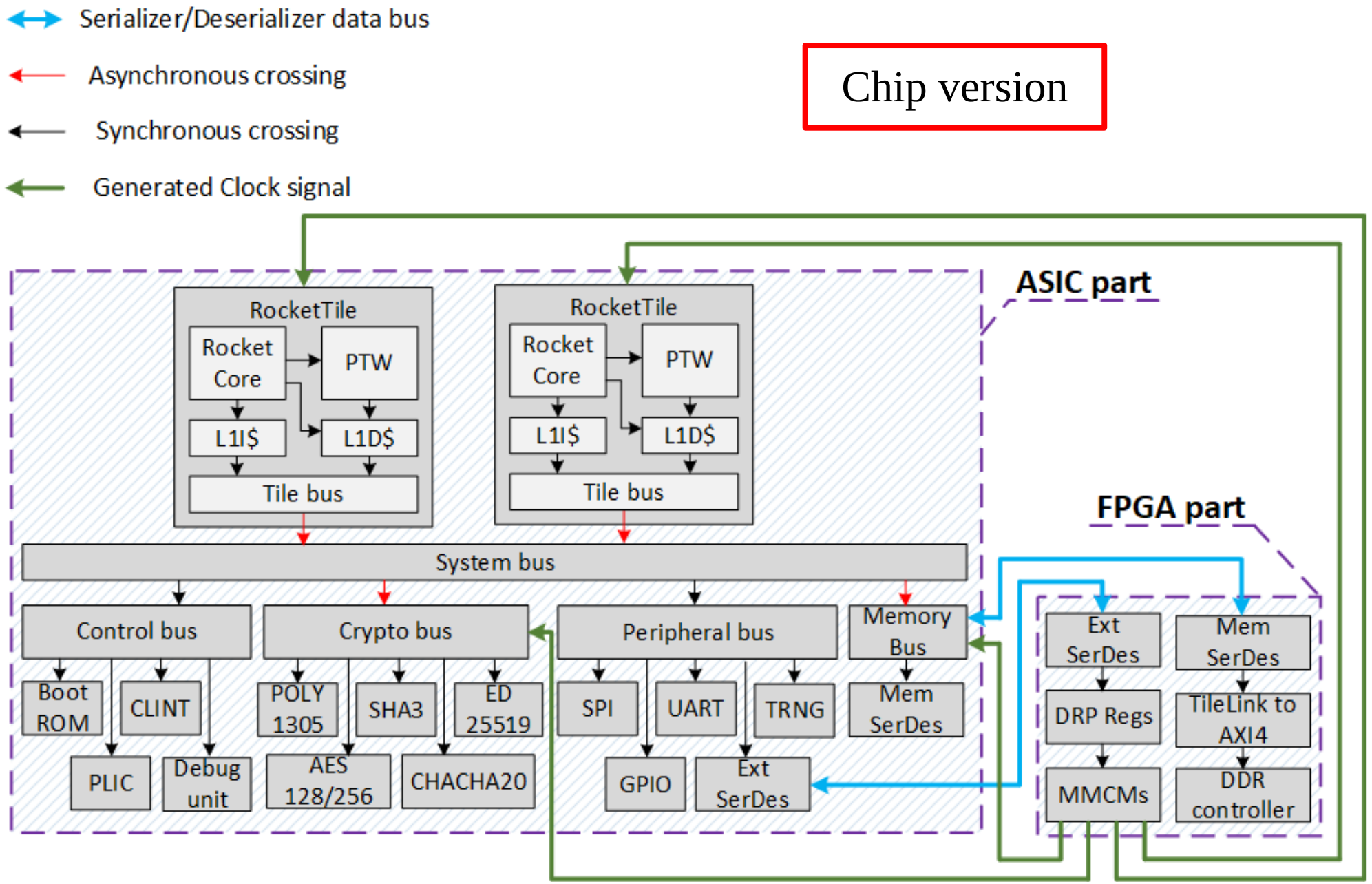
Protected Cryptographic SoC with RDFS



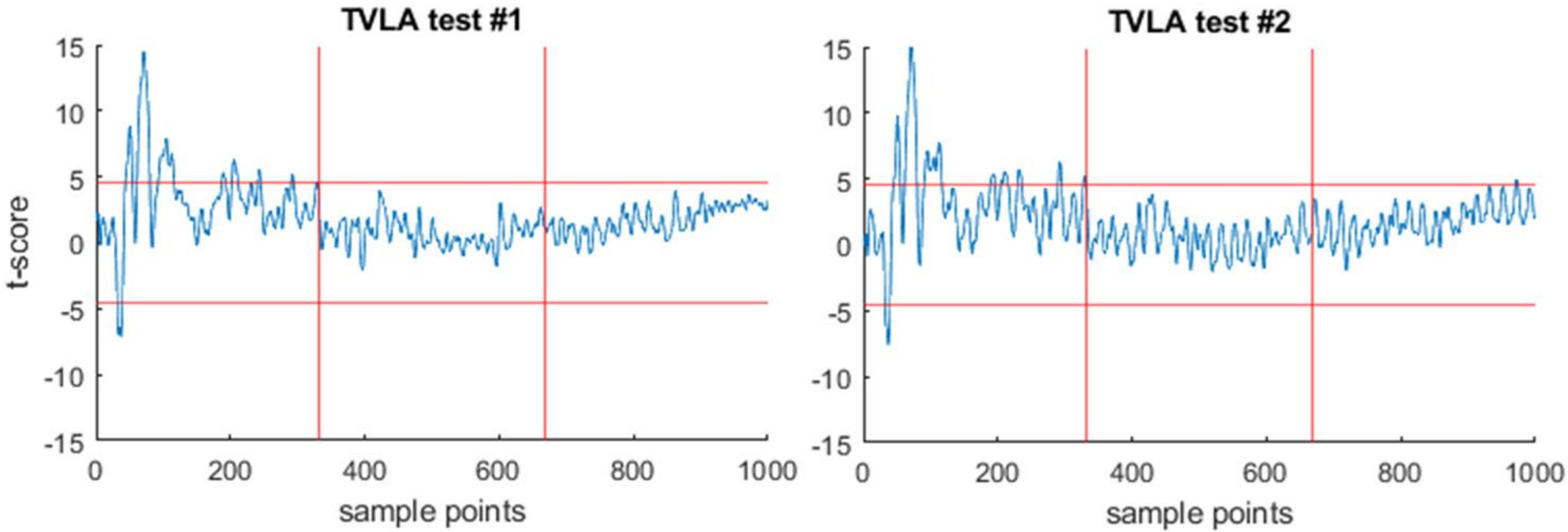
Protected SoC with RDFS:

- Add Clock Generation peripherals (use Xilinx's Clock Manager IP)
- Create > 219.000 frequencies (in range from 50MHz to 100MHz)
- Verify accuracy by Pulse counter
- Only applied to AES-128 module
- Scale AES's CLK after each encryption

5. Main research @ UEC (18/30) CPA countermeasure



5. Main research @ UEC (19/30) CPA countermeasure

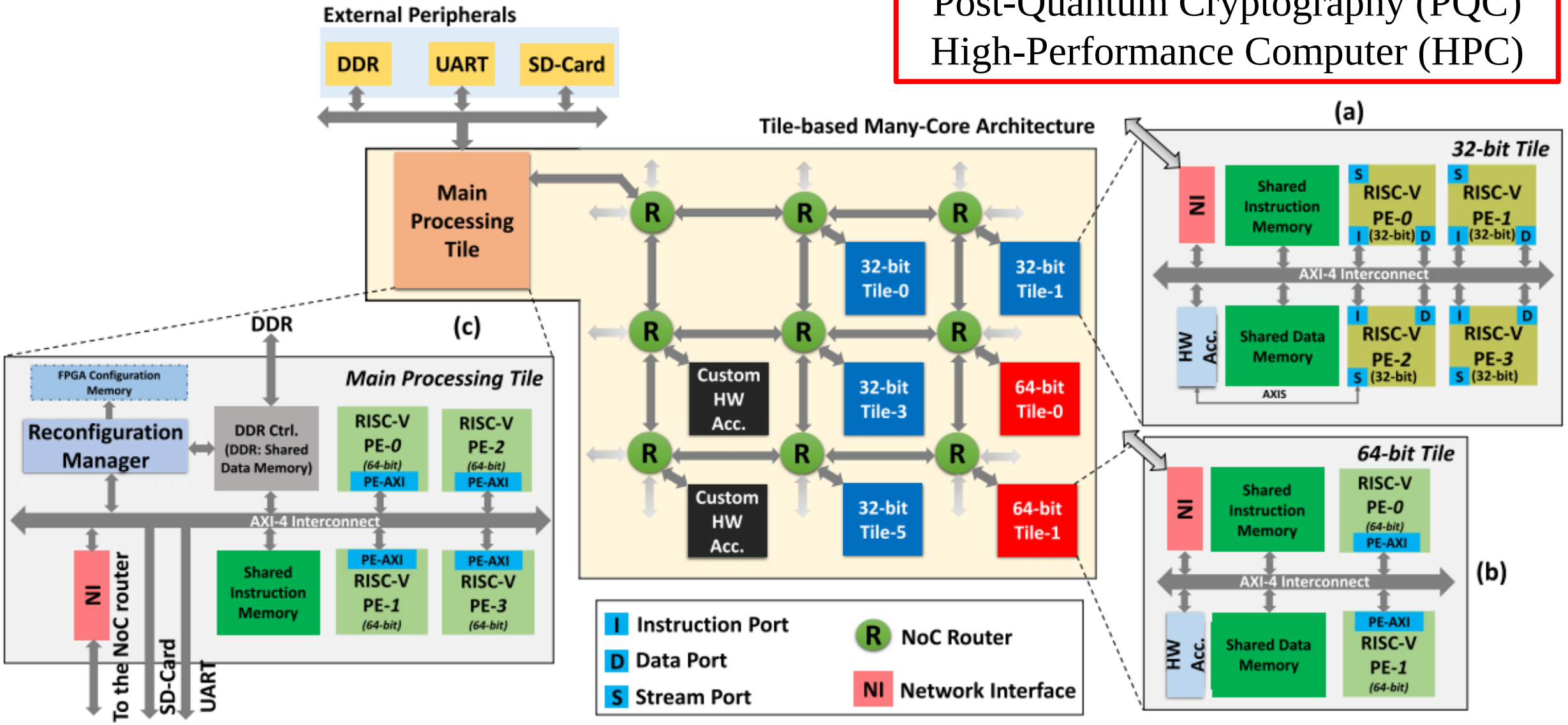


Test Vector Leakage Assessment (TVLA) results:

- RDFS with 219,412 clk freq. (50MHz - 100MHz)
- Does not detect any leakage in 5 million power traces

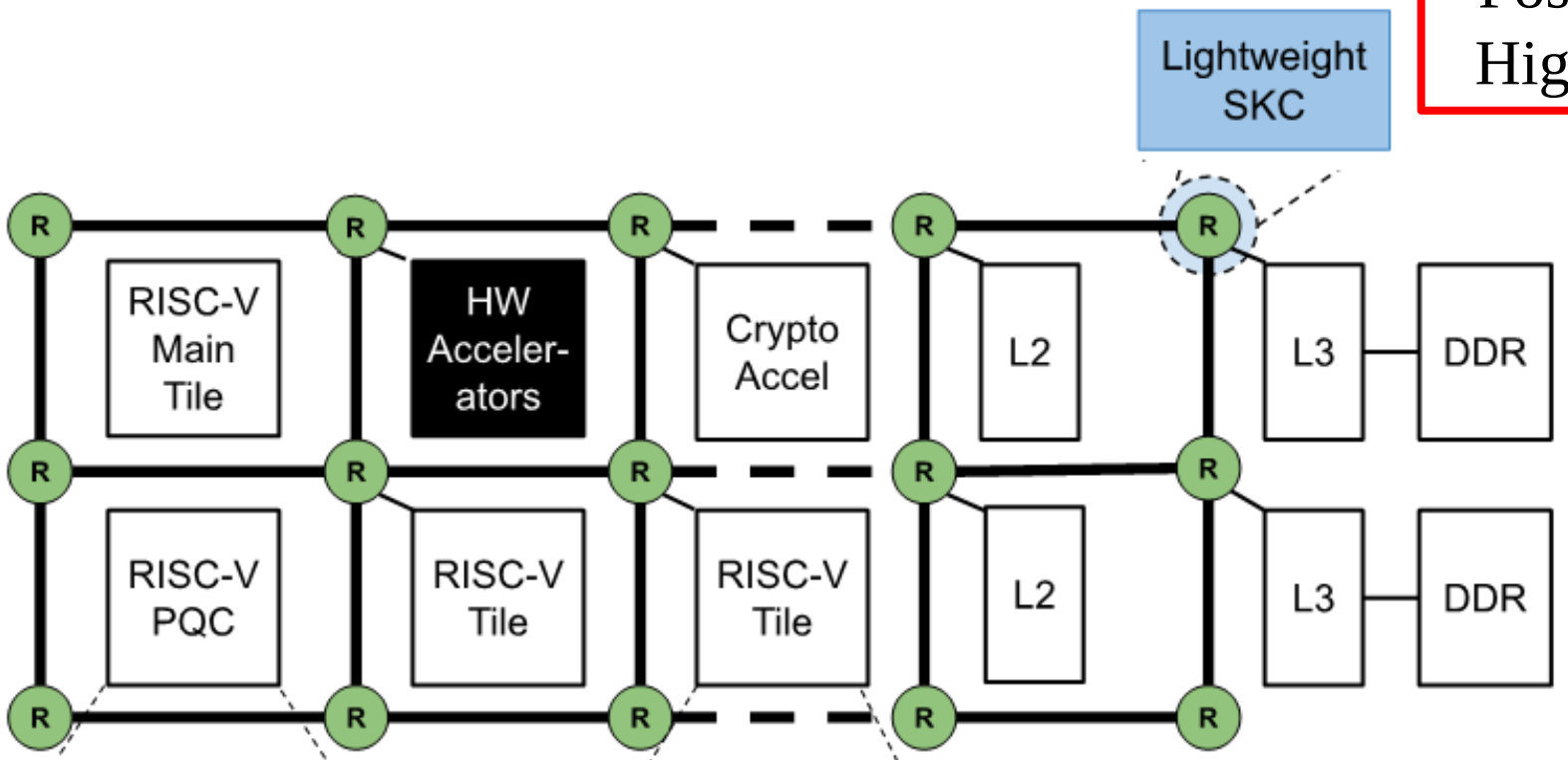
5. Main research @ UEC (20/30) PQC-HPC *(future project)*

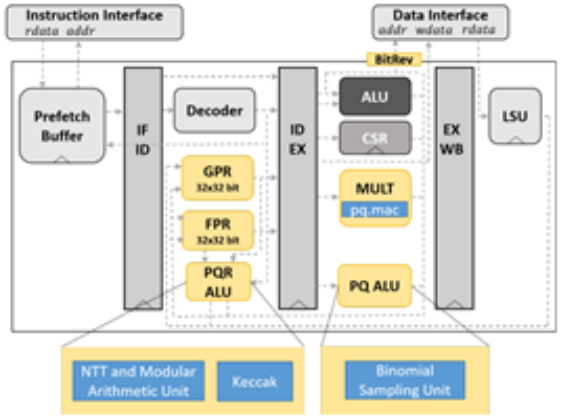
Post-Quantum Cryptography (PQC)
High-Performance Computer (HPC)

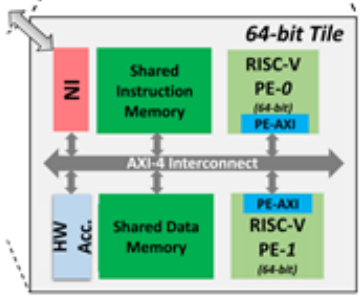


5. Main research @ UEC (21/30) PQC-HPC *(future project)*

Post-Quantum Cryptography (PQC)
High-Performance Computer (HPC)



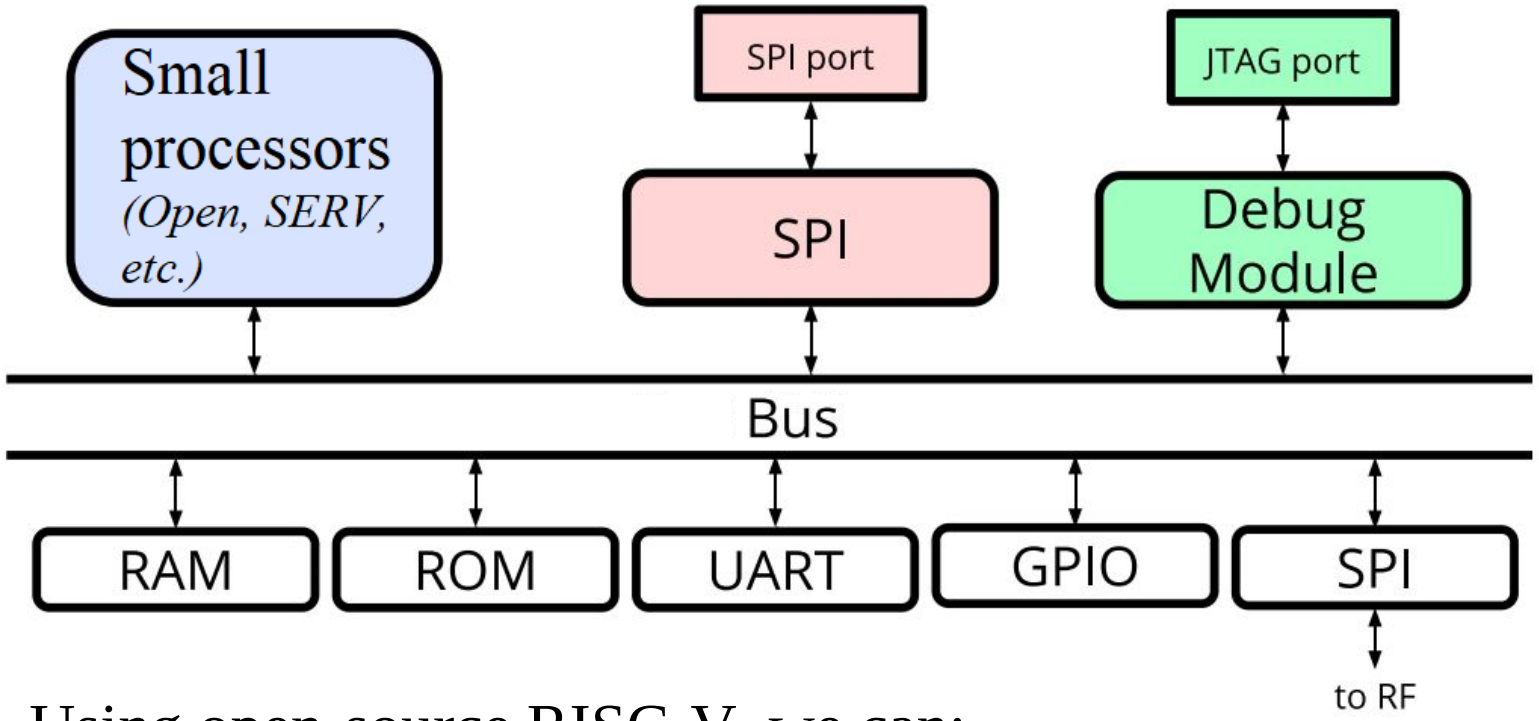




- RISC-V main tile including a Network Interface (NI) to connect with NoC.
- Lightweight Symmetric Key (SKC) is applied to Router/NI.
- PQCs integrated inside RISC-V cores.
- Hardware accelerators
- Exploit special RISC-V ISA extensions for PQC.

5. Main research @ UEC (22/30) ULP-SoC

Modular for Ultra-Low Power (ULP) IoT-SoC architecture based on RISC-V



Open-source modular framework aiming for:

- Highly customizable
- Flexible and portable
- Wide range of applications

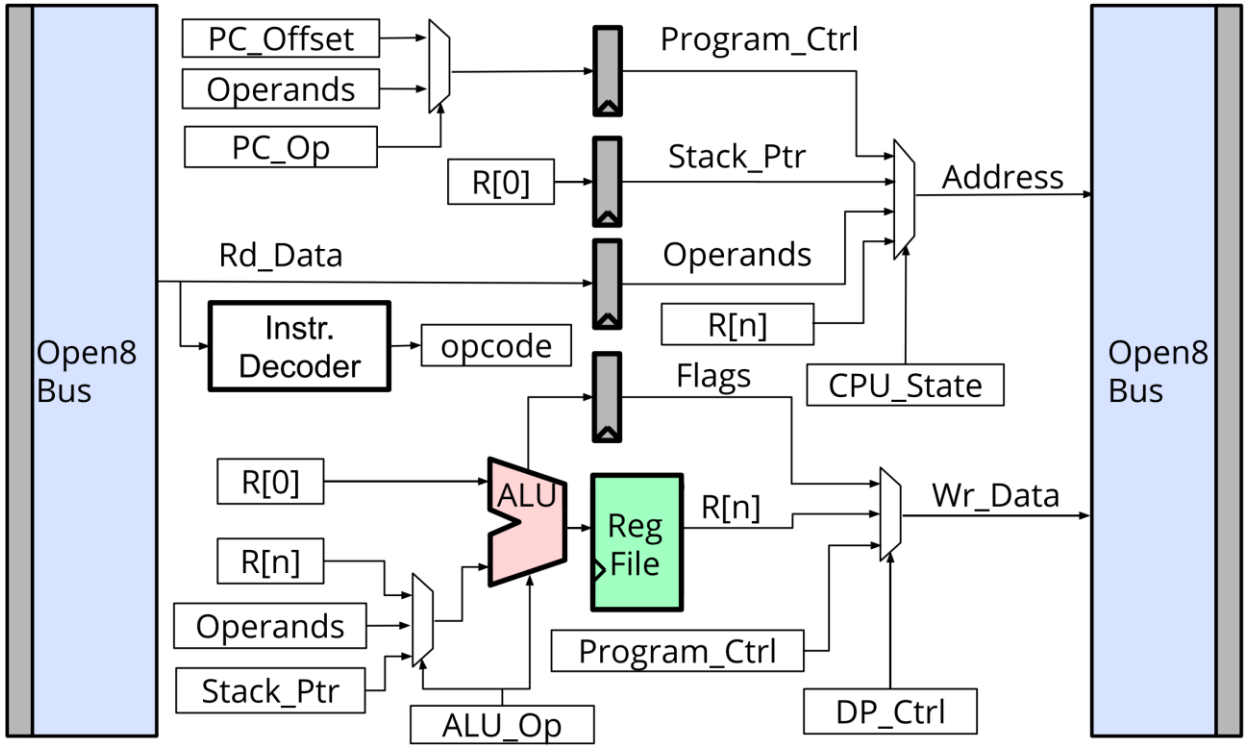
Using open-source RISC-V, we can:

- Cheery-pick the wanted ISA extensions.
- Multiple options for the core processor(s) with wanted features *(i.e., low-power, high-performance, and security)*.
- Easy to develop & debug software.
- Vast options of peripherals: open sources, commercial IPs, in-house developments.

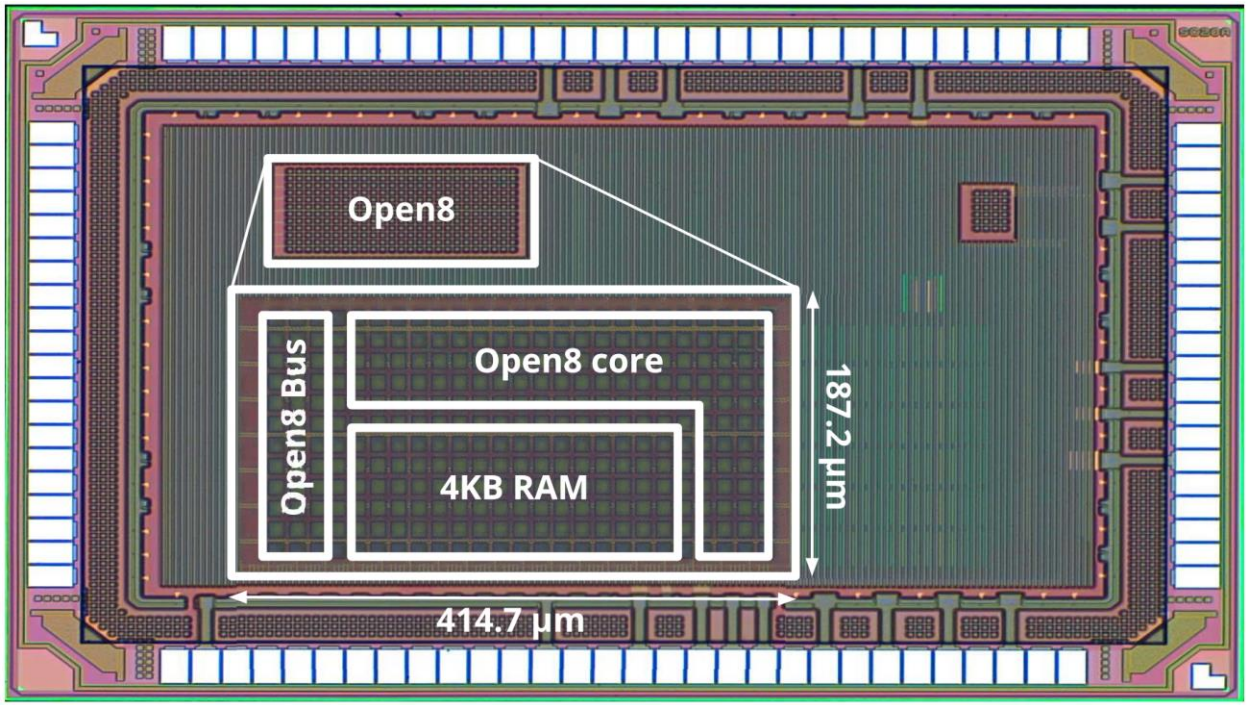
5. Main research @ UEC (23/30) ULP-SoC

Experiment with small MCUs:

try to achieve sub-µW power consumption



Open8 core data-path



Open8 in SOTB-65nm

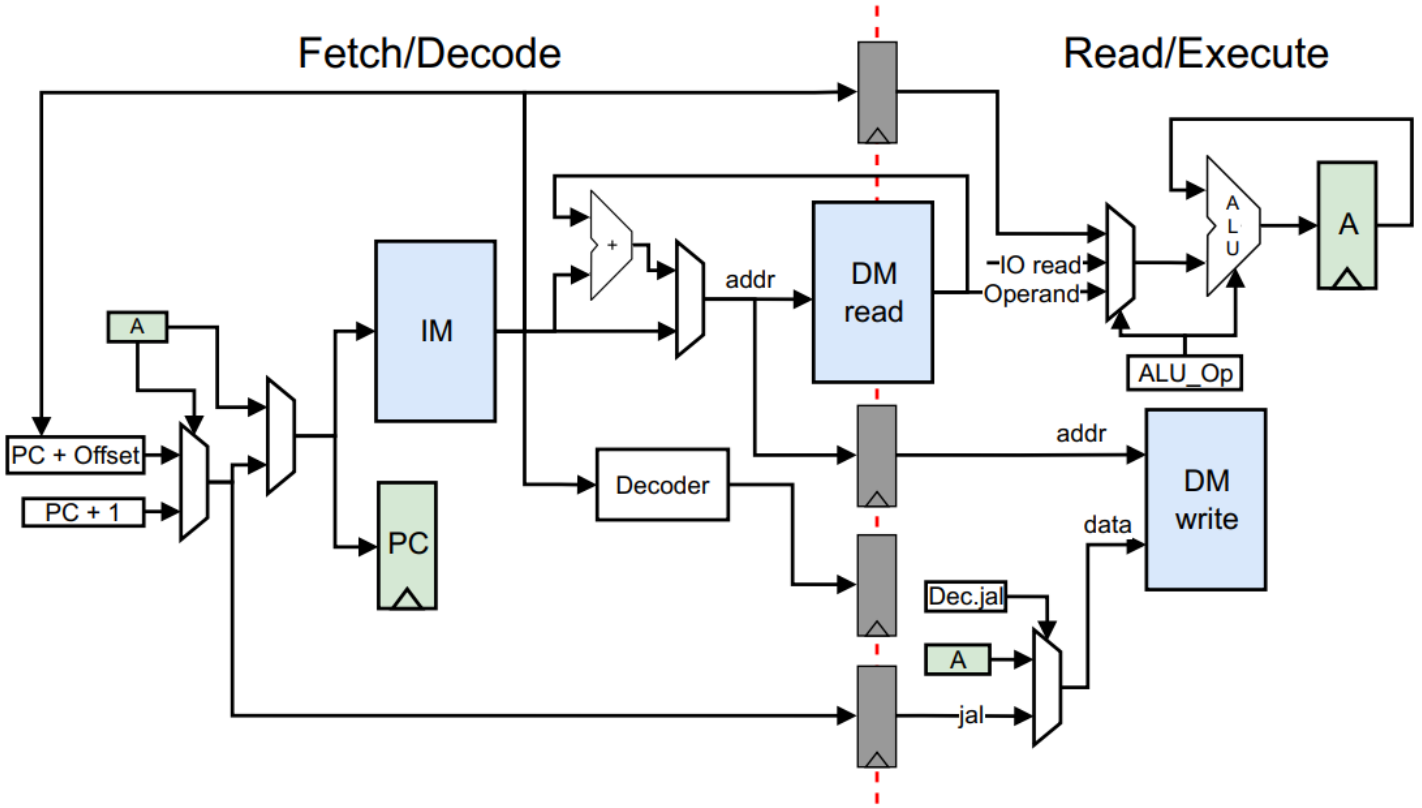
8-bit RISC processor

Open8 in *TCAS-II* (2021)

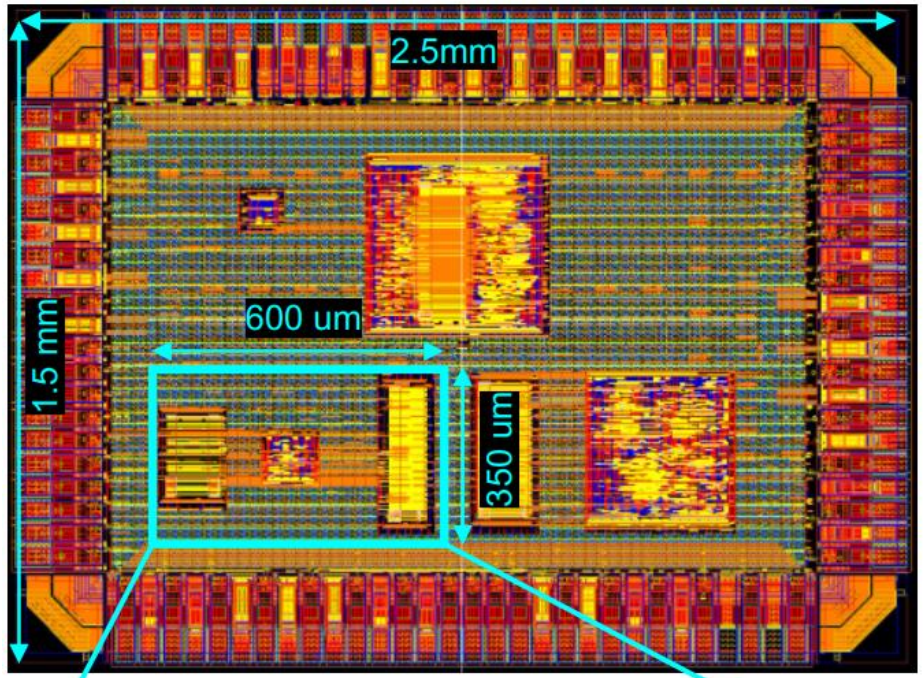
5. Main research @ UEC (24/30) ULP-SoC

Experiment with small MCUs:

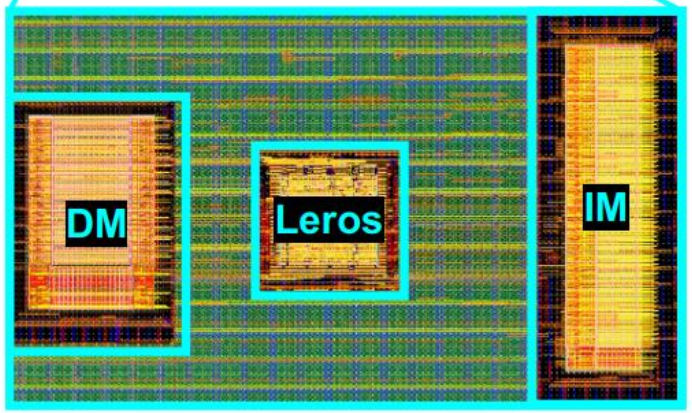
try to achieve sub-μW power consumption



Leros core data-path



Leros in SOTB-65nm



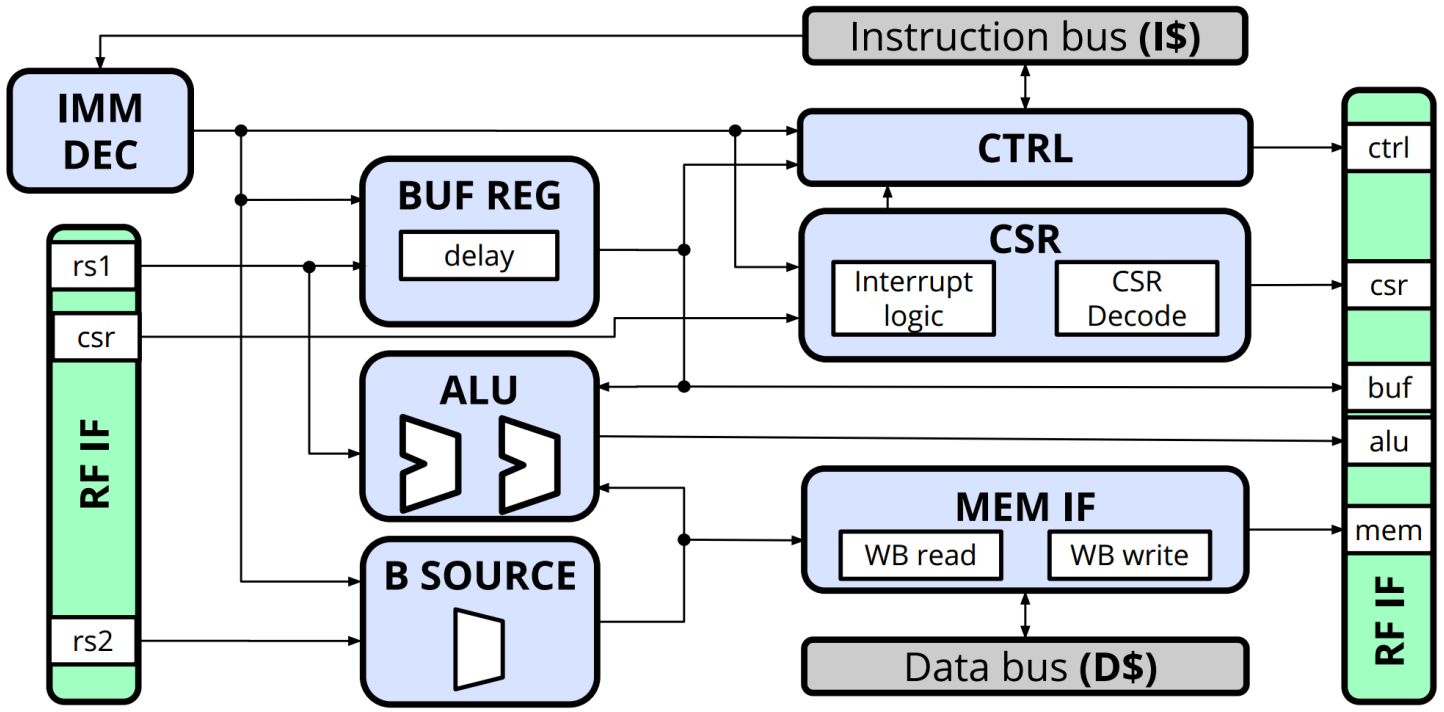
16-bit RISC processor

Leros in *ICICDT* (2022)

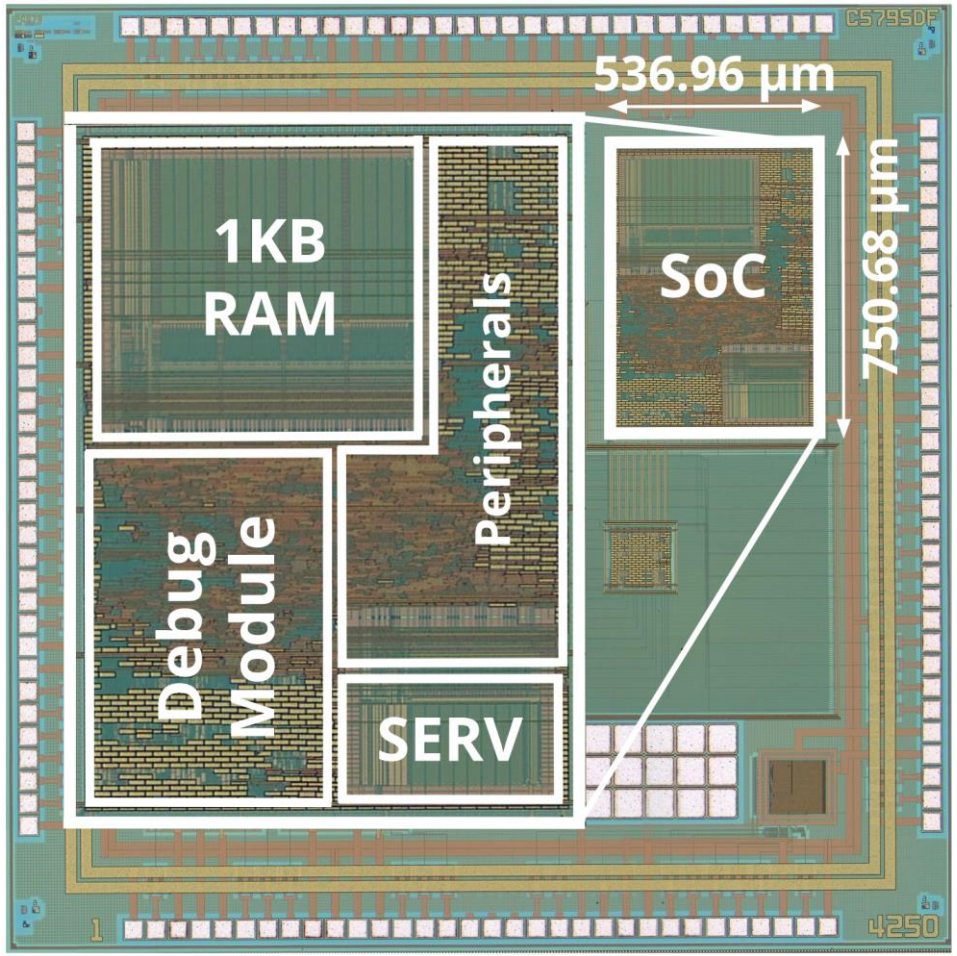
5. Main research @ UEC (25/30) ULP-SoC

Experiment with small MCUs:

try to achieve sub-µW power consumption



SERV core data-path



SERV in ROHM-180nm

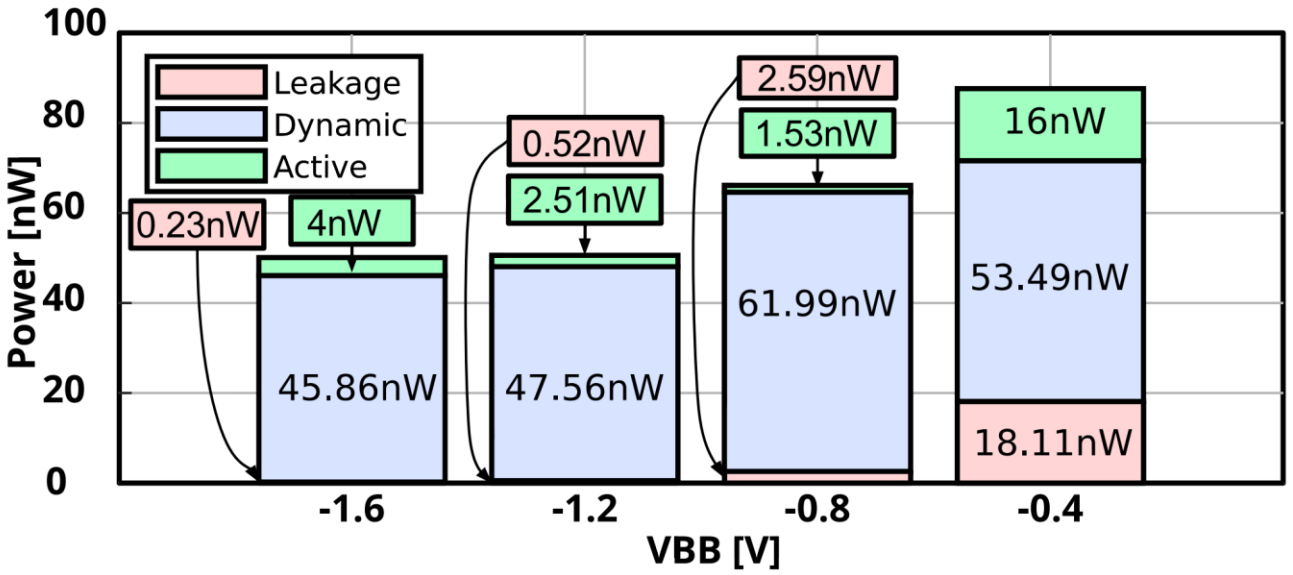
32-bit RISC-V serial processor

SERV in *ISOCC* (2021)

5. Main research @ UEC (26/30) ULP-SoC

Experiment with small MCUs:

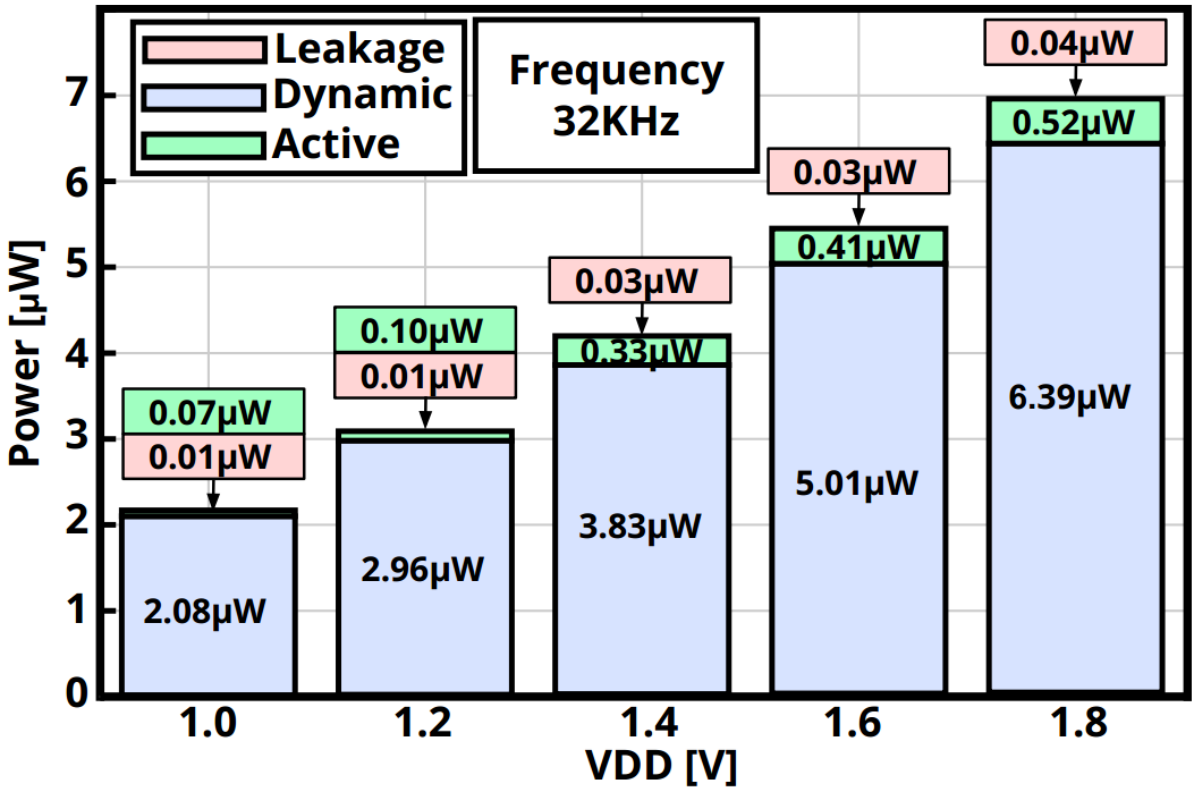
try to achieve sub-μW power consumption



Open8 in SOTB65

(fixed V_{DD}=0.5-V & F=32-KHz)

Best P = 46.13nW *@V_{DD}=0.5-V, V_{BB}=−1.6-V & F=32-KHz*



SERV in ROHM180nm

Best P = 2.16μW

@V_{DD}=1.0-V & F=32-KHz

Open8 in *TCAS-II* (2021)

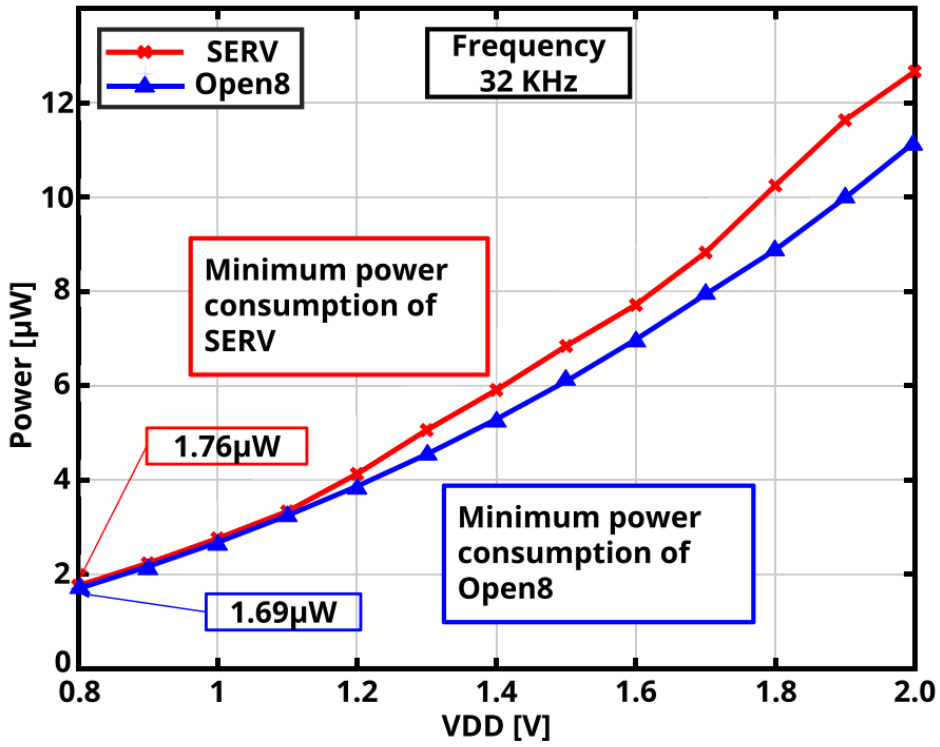
SERV in *ISOCC* (2021)

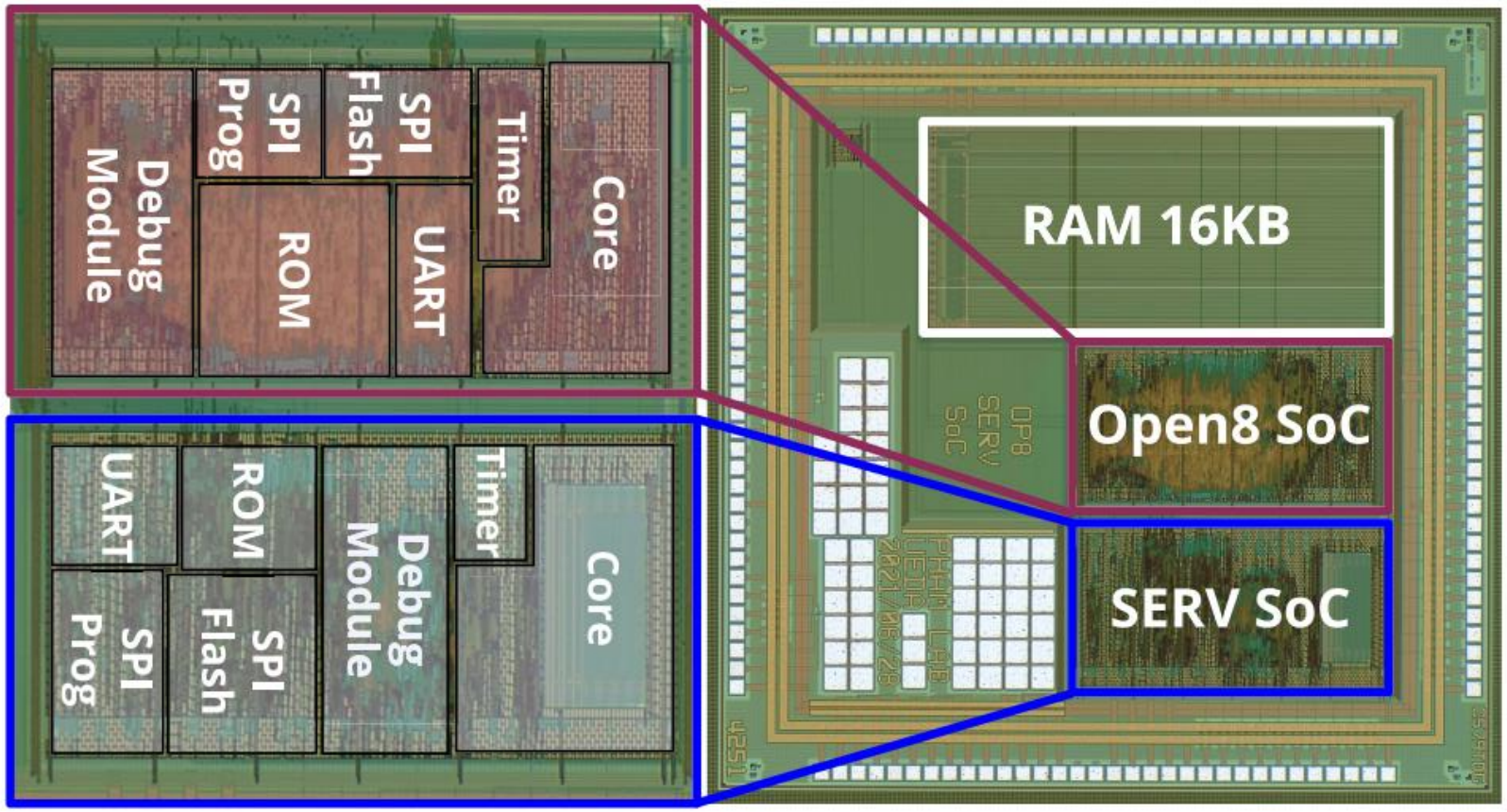
5. Main research @ UEC (27/30) ULP-SoC

Experiment with small MCUs:

try to achieve sub-μW power consumption

The dual-core of Open8 & SERV in this chip is just for comparison fairness.





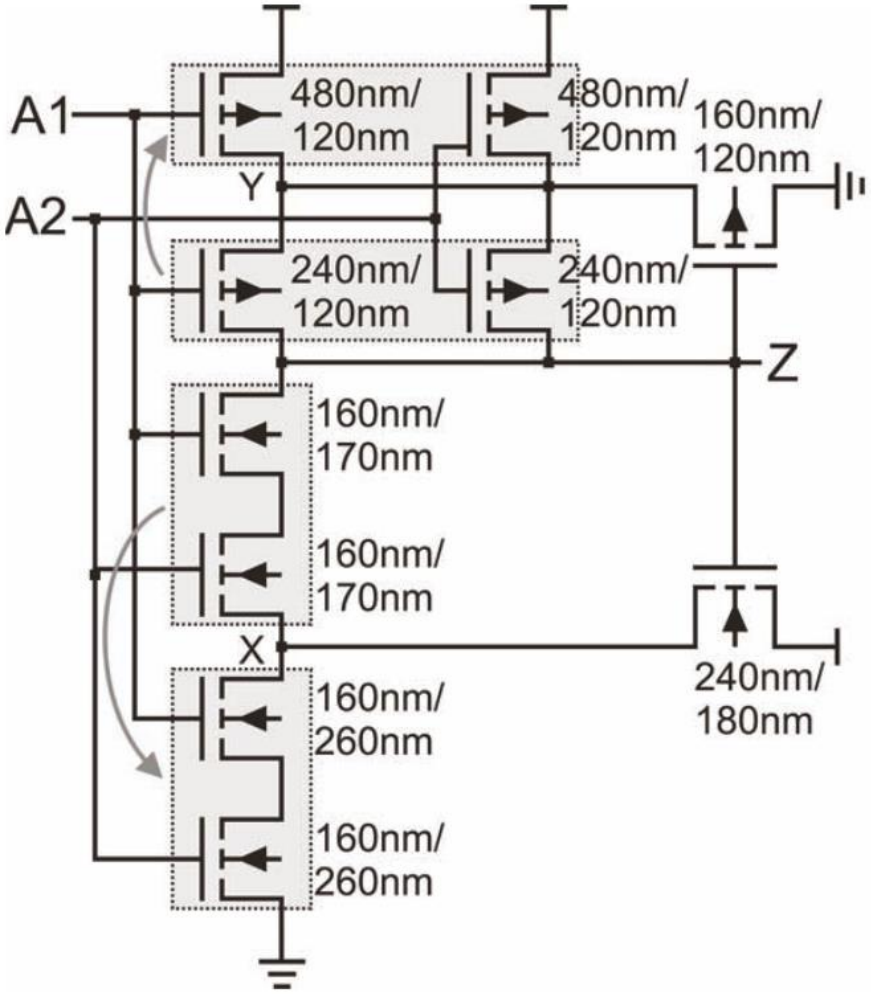
Open8+SERV in [TCAS-II](#) (2022)

Open8+SERV in ROHM-180nm

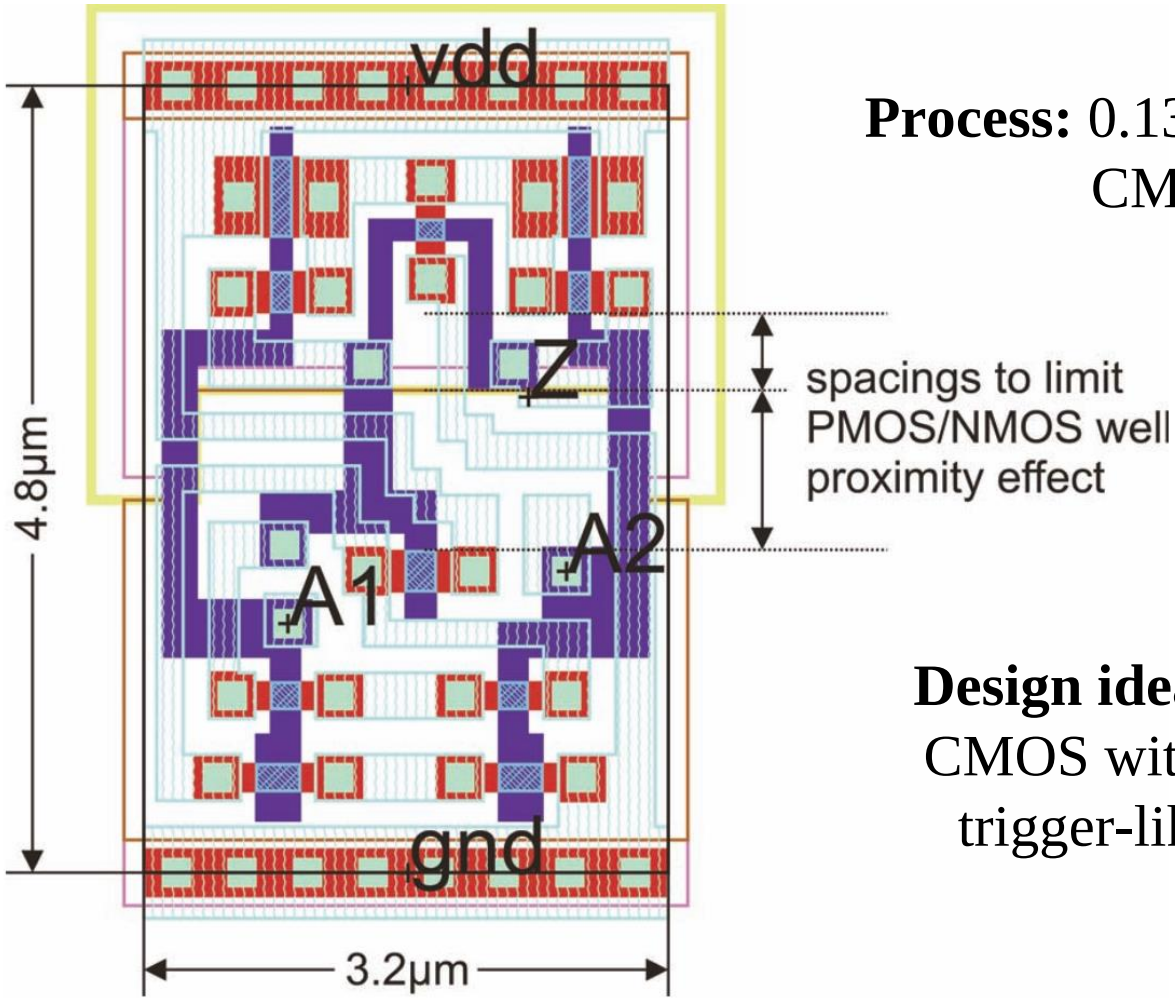
5. Main research @ UEC (28/30) ULP-SoC *(future project)*

Develop Ultra-Low-Power (ULP) standard cell

Design a ULP standard cells library: intentionally trade-off F_{Max} for low power



Schematic of the NAND2 gate



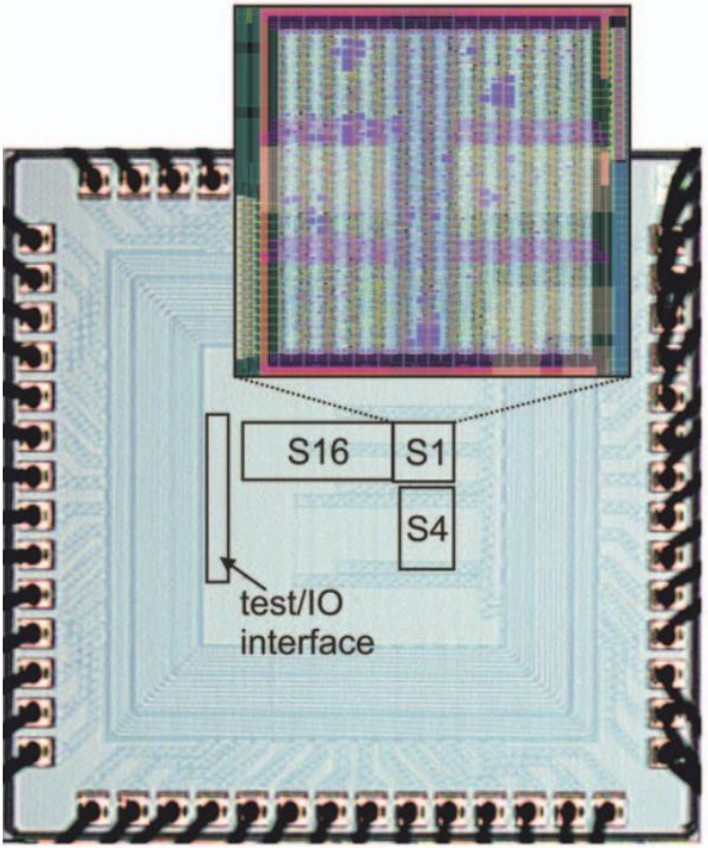
Layout of the NAND2 gate

Process: 0.13μm standard CMOS

Design idea: standard CMOS with Schmitt-trigger-like design

Source: *ISSCC* (2011)

5. Main research @ UEC (29/30) ULP-SoC *(future project)*



Source: *ISSCC* (2011)

Process		0.13μm 8-metal CMOS
Design		8x8 Multiplier
Area (S1/S4/S16)		19176/23978/45002μm²
Supply Voltage (S1/S4/S16)		min. 84/68/62mV max. 1.2V
Minimum Energy/Operation Point		
Frequency	S1	544.8kHz@260mV
	S4	328.3kHz@233mV
	S16	210.1kHz@226mV
Power	S1	346.6nW@260mV
	S4	243.8nW@233mV
	S16	275.6nW@226mV
Energy/ Operation	S1	0.63pJ@260mV
	S4	0.74pJ@233mV
	S16	1.31pJ@226mV
Minimum Supply Voltage		
Frequency	S1	15.20kHz@84mV
	S4	10.35kHz@68mV
	S16	5.15kHz@62mV
Power	S1	29.9nW@84mV
	S4	21.1nW@68mV
	S16	17.9nW@62mV
Energy/ Operation	S1	1.97pJ@84mV
	S4	2.04pJ@68mV
	S16	3.48pJ@62mV

509.6pW/MHz-GE@0.26-V

⇩

Scaling equations in *Integration* (2017)

⇩

345.14pW/MHz-GE @0.5-V *(SOTB65 no bias)*

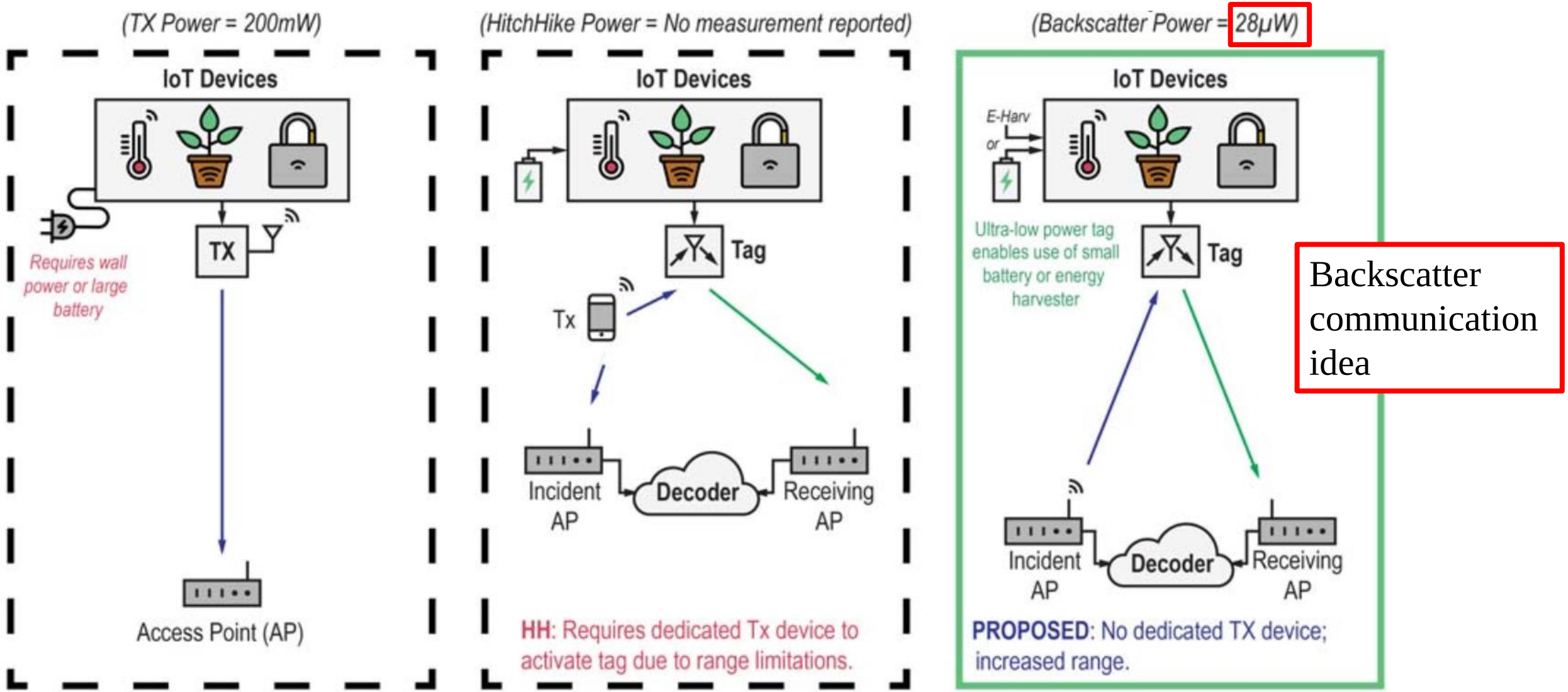
⇩

Open8 [26] *(9kGE)*	SOTB-65nm: 99.4nW @0.5-V no bias & 32-KHz
SERV [27] *(2kGE)*	SOTB-65nm: 22.09nW @0.5-V no bias & 32-KHz

**Note:* ~10× more power reduction when a reverse bias is applied.

5. Main research @ UEC (30/30) ULP-SoC *(future project)*

Develop ULP RF transceiver & communication protocol



Backscatter communication idea

ISSCC (2020)

OUTLINE

1. Introduction
2. Achievement highlight
3. What is RISC-V?
4. Working with RISC-V & open community
5. Main research @ UEC
6. RISC-V courses @ UEC

6. RISC-V courses @ UEC (1/6) Five courses

Five main courses: from *basic* to *chip* and beyond

No.	Name	Core knowledge
1	RISC-V Computer System Integration	• How to add your custom hardware *(Verilog or VHDL)* to an existing RISC-V computer system → generate a new → control/debug it in software
2	Hardware Design with Chisel/Scala	• Learn how to design a hardware circuit using Scala/Chisel • Learn how to create a new RISC-V computer system using open Scala/Chisel libraries
3	Linux on RISC-V	• Lean how to boot Linux on a RISC-V computer system → control your custom hardware after Linux boot
4	VLSI Design with RISC-V System-on-Chip (SoC)	• Make a RISC-V chip: flat layout for a simple system, hierarchy layout for a complex system
5	Cyber-security with RISC-V Computer System	• Learn how to modify a RISC-V computer system for security purposes: secure boot, cryptographic acceleration, prevent side-channel attacks, etc.

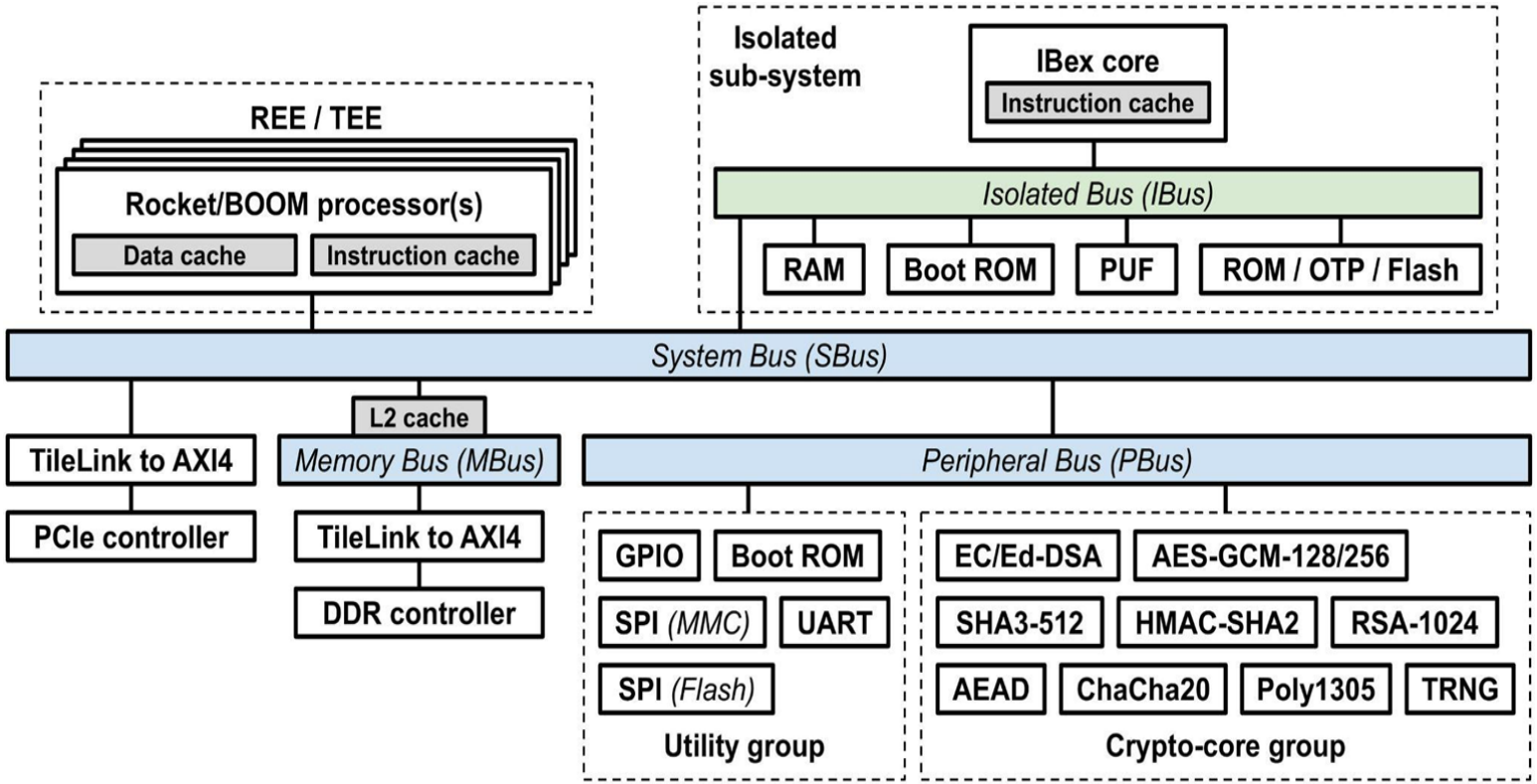
6. RISC-V courses @ UEC (2/6) Classes example

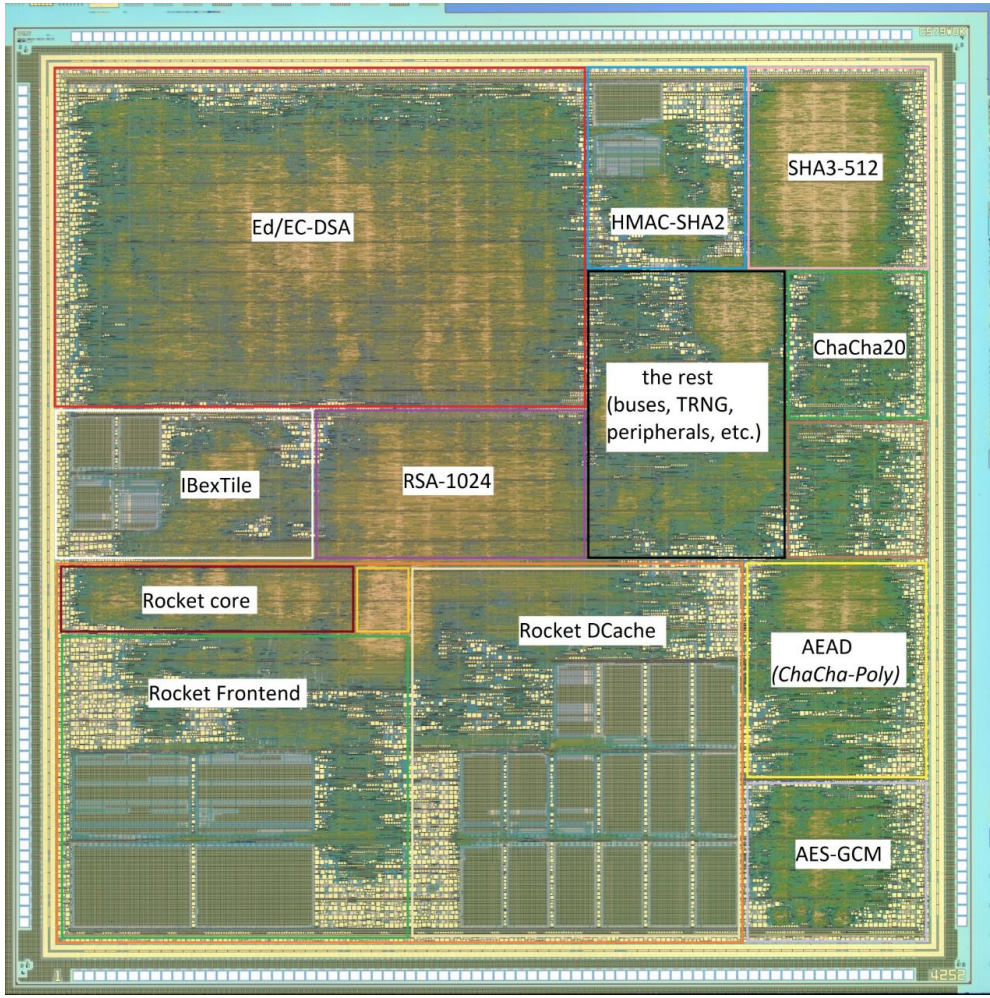
Week	Title
1	RISC-V Introduction
2	VexRiscv: a simple 32-bit MCU
3	Computer system with Rocket: Introduction, make, & program Arty-A7
4	Computer system with Rocket: System details & system modifications
5	Computer system with Rocket: Boot sequence & making your software
6	Midterm review
7	Computer system with Rocket: Make a custom hardware
8	Computer system with Rocket: Make a custom hardware (continue)
9	Computer system with Rocket: Custom hardware with external IOs
10	Make a chip in ROHM-180nm: Introduction & using templates
11	Make a chip in ROHM-180nm: Flat layout with VexRiscv
12	Make a chip in ROHM-180nm: Hierarchy layout with Rocket
13	Make a chip in ROHM-180nm: Frame integration
14	Class summary (total review)

- We're doing the VLSI design class at UEC.
- The content was the combined course **1** *(integration)* and **4** *(VLSI design)*

6. RISC-V courses @ UEC (3/6) Example design

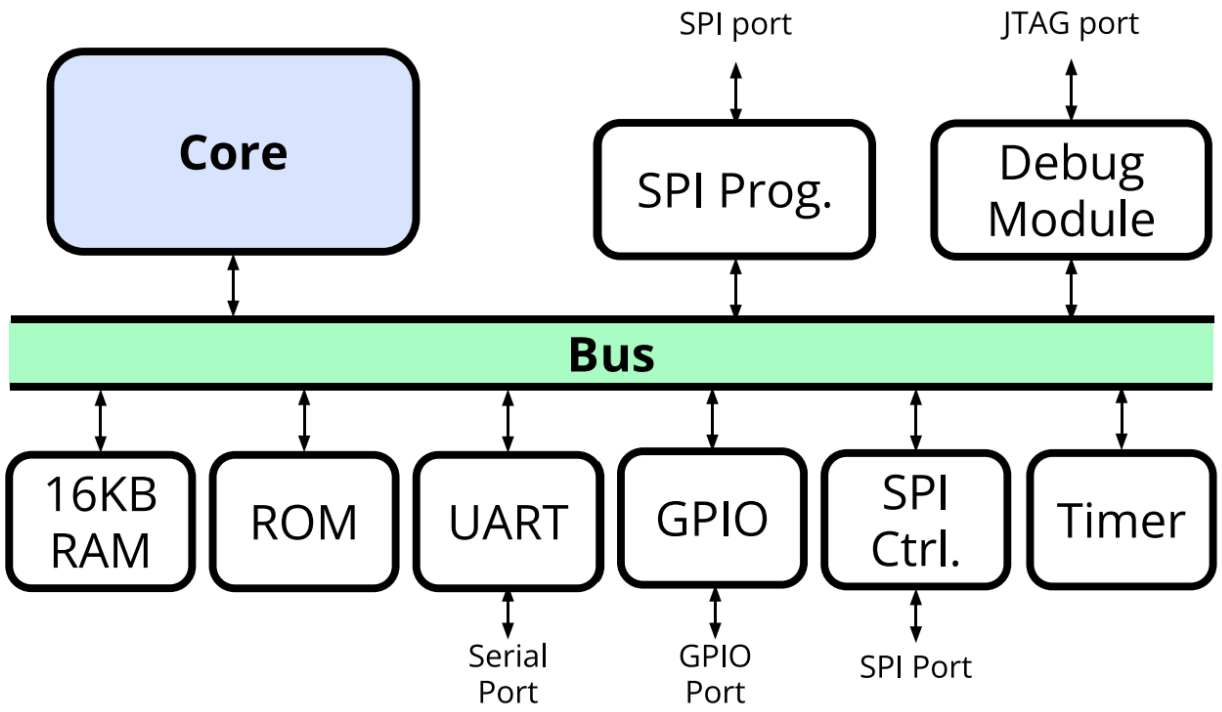
- Crypto-SoC with crypto-core accelerators, hidden MCU for secure boot, and complex key scheduling.

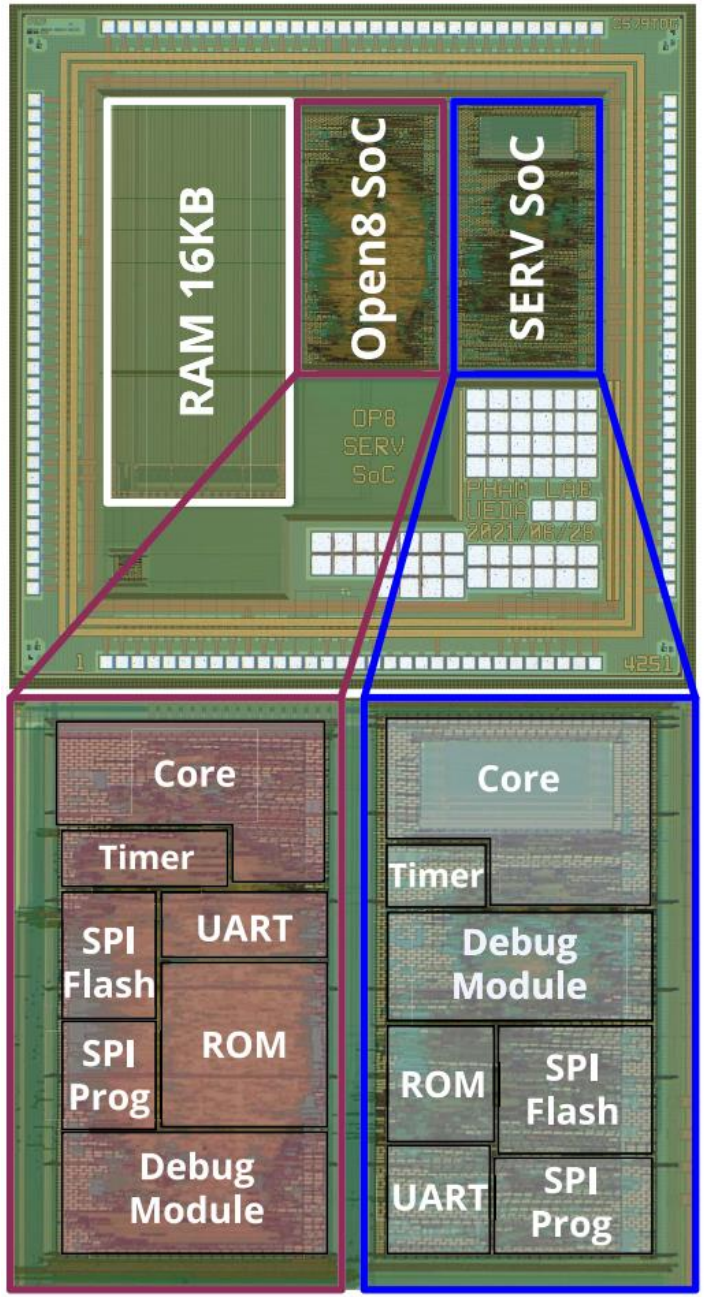




6. RISC-V courses @ UEC (4/6) Example design

- ULP-SoC with small processor(s) and simple peripherals to achive sub-μW power consumption.





6. RISC-V courses @ UEC (5/6) Oversea courses

Sep. 2022

Place

Academy of Cryptography Techniques (ACT), Hanoi, Vietnam

Content

Course **1**: *RISC-V Computer System Integration*

6. RISC-V courses @ UEC (6/6) Oversea courses

Potential future classes

Date	Place	Content
Aug. 2023	University of Electro-Communications (UEC) *Tokyo, Japan*	Course **1**: *RISC-V Computer System Integration* Course **5**: *Cyber-security with RISC-V Computer System*
Summer/Fall 2023	Le Quy Don Technical University (LQDTU) *Hanoi, Vietnam*	N/A
Summer/Fall 2023	Ho Chi Minh University of Technology (HCMUT) *HCM city, Vietnam*	N/A

THANK YOU