EP-ESE
ELECTRONIC SYSTEMS FOR EXPERIMENTS

WP5 IC Tech
EP R&D

SOCRATES
A Radiation-Tolerant SoC Generator Framework

Marco Andorno (marco.andorno@cern.ch)

On behalf of the CERN EP R&D WP5.3 team:
Marco Andorno, Alessandro Caratelli, Davide Ceresa, Benoit Denkinger, Kostas Kloukinas, Anvesh Nookala

2024/10/01 – TWEPP 2024

Outline

Introduction

- Motivations
- R&D context

The SOCRATES platform

- SoCMake build system
- Radiation-tolerance add-ons

The TriglaV prototype

- Architecture
- Fault-tolerance and observability
- Current status and future testing

Final remarks

- Possible applications
- Future outlook

The problem:

Future LHC and detector upgrades require **more complex ASICs** → They require advanced technology nodes, that come with **high development costs**

The solution we propose:

Programmability

- Retarget an ASIC for different applications
- Change the algorithm at runtime

HW/SW codesign

- Unified build system
- System bring-up from a common description

IP blocks library

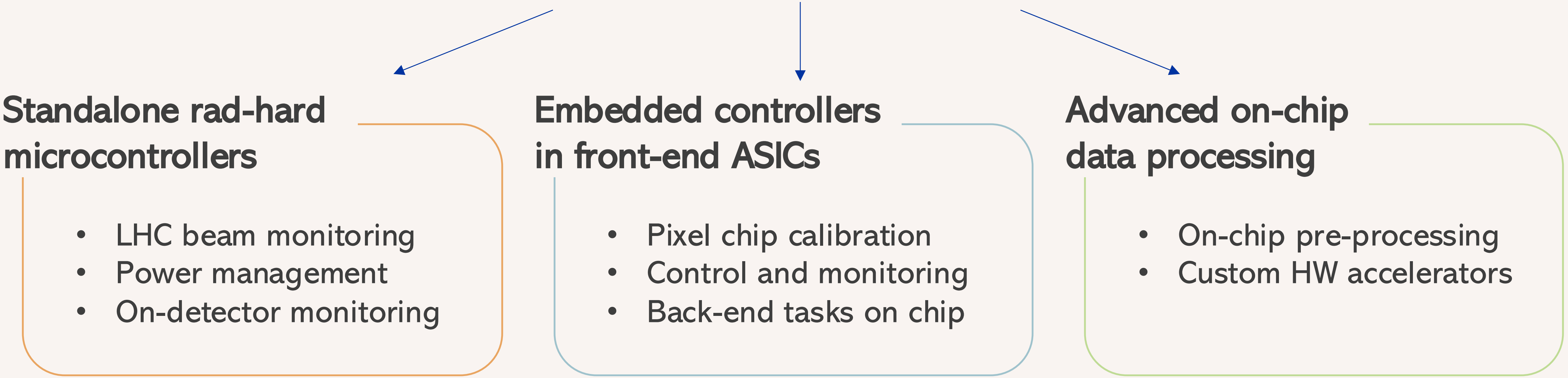
- Pre-verified building blocks
- Promote collaborative work in the community
- Compatible with a standard bus protocol

Towards System-on-Chips

Turning our custom ASICs into programmable SoCs will provide:

- Quick prototyping
- Smaller number of more capable ASICs
- Faster design and verification turnaround time
- Cost effective development

Where are SoCs applicable for the HEP community?



Standalone rad-hard microcontrollers

- LHC beam monitoring
- Power management
- On-detector monitoring

Embedded controllers in front-end ASICs

- Pixel chip calibration
- Control and monitoring
- Back-end tasks on chip

Advanced on-chip data processing

- On-chip pre-processing
- Custom HW accelerators

… but a single architecture can’t fit all these applications, so…

SoC RAdiation Tolerant Eco-System

… but a single architecture can’t fit all these applications, so…

⇩

SOCRATES

Comprehensive **toolkit** to generate highly-customizable systems that can be integrated in custom radiation-tolerant ASICs. Its main goals:

- Automatic SoC assembly and SW stack
- Library of rad-tol verified IP blocks
- Fault-tolerance support

➡ It’s going to be open-sourced for HEP and the wider open-source HW community

Development within the **CERN EP R&D WP5**:

Marco Andorno, Alessandro Caratelli, Davide Ceresa, Benoit Denkinger, Kostas Kloukinas, Risto Pejasinovic, Anvesh Nookala

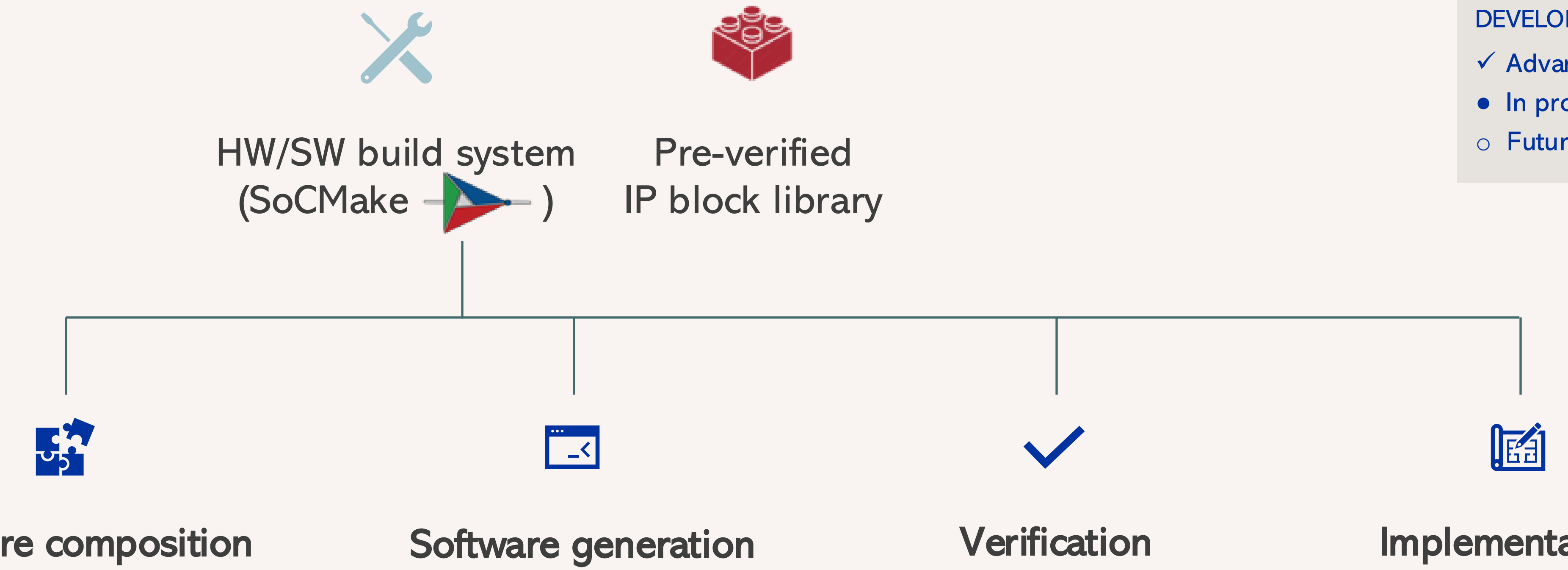
The SOCRATES platform

DEVELOPMENT STAGE:

✓ Advanced

● In progress

○ Future developments



HW/SW build system (SoCMake)

Pre-verified IP block library

Hardware composition

✓ Generate the top-level SoC

✓ Integrate IP blocks

✓ Infer interconnects (crossbars, adapters)

Software generation

✓ Invoke toolchain

✓ Generate hardware abstraction layer (HAL)

✓ Generate linker scripts

Verification

✓ Functional testing environment

● SystemC-UVM environment

✓ FPGA prototyping

Implementation

● Run synthesis and physical implementation flow on targets

○ Generate constraints

… and its rad-hardening add-ons

DEVELOPMENT STAGE:

✓ Advanced

● In progress

○ Future developments

HW/SW build system (SoCMake)

Pre-verified IP block library

Radiation hardening

Hardware composition

- ✓ Generate the top-level SoC
- ✓ Integrate IP blocks
- ✓ Infer interconnects (crossbars, adapters)
- ✓ Fault tolerant interconnects
- ✓ Automatic TMR of IP blocks

Software generation

- ✓ Invoke toolchain
- ✓ Generate hardware abstraction layer (HAL)
- ✓ Generate linker scripts
- ○ Software fault tolerance

Verification

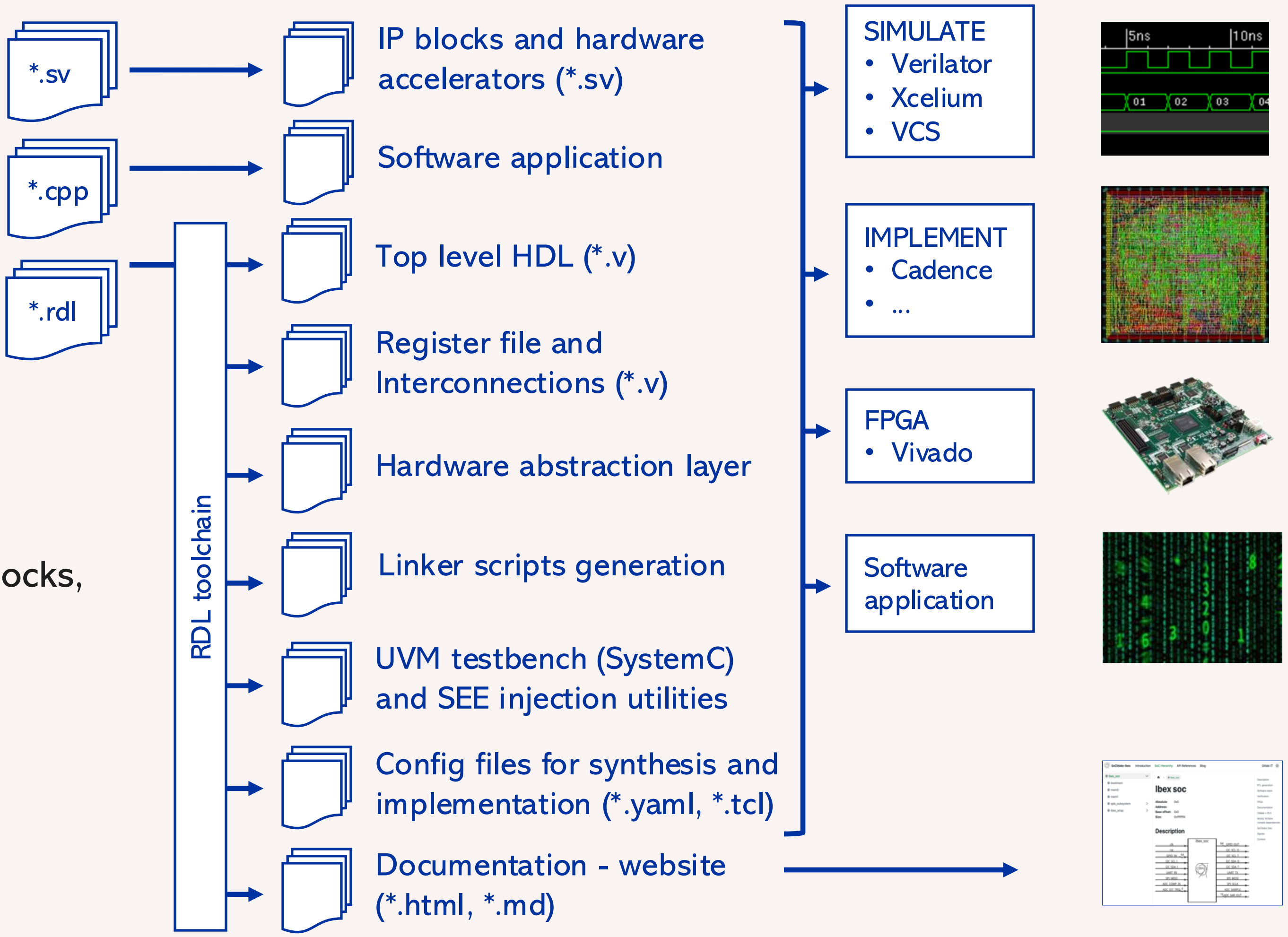
- ✓ Functional testing environment
- ● SystemC-UVM environment
- ✓ FPGA prototyping
- ✓ Fault injection framework
- ✓ Formal verification for TMR

Implementation

- ● Run synthesis and physical implementation flow on targets
- ○ Generate constraints
- ○ Constraints for fault tolerance

SoCMake

- **CMake**-based HW/SW build system generator for SoCs
- Supports **SystemRDL** as an input configuration
  - Register Description Language extended for supporting SoC description
  - Used as top-level architecture description
- Rapid prototyping of SoCs:
  - Quick composition of IP blocks into full systems
  - Quickly build different architectures with different IP blocks, hardware accelerators or different CPUs
  - Dependency management
- **Open-source**
  - SoCMake: github.com/HEP-SoC/SoCMake
  - Toolkit and implementation: gitlab.cern.ch/socrates



*.sv

*.cpp

*.rdl

RDL toolchain

IP blocks and hardware accelerators (*.sv)

Software application

Top level HDL (*.v)

Register file and Interconnections (*.v)

Hardware abstraction layer

Linker scripts generation

UVM testbench (SystemC) and SEE injection utilities

Config files for synthesis and implementation (*.yaml, *.tcl)

Documentation - website (*.html, *.md)

SIMULATE
- Verilator
- Xcelium
- VCS

IMPLEMENT
- Cadence
- …

FPGA
- Vivado

Software application

TriglaV: from SOCRATES to silicon

Goals:

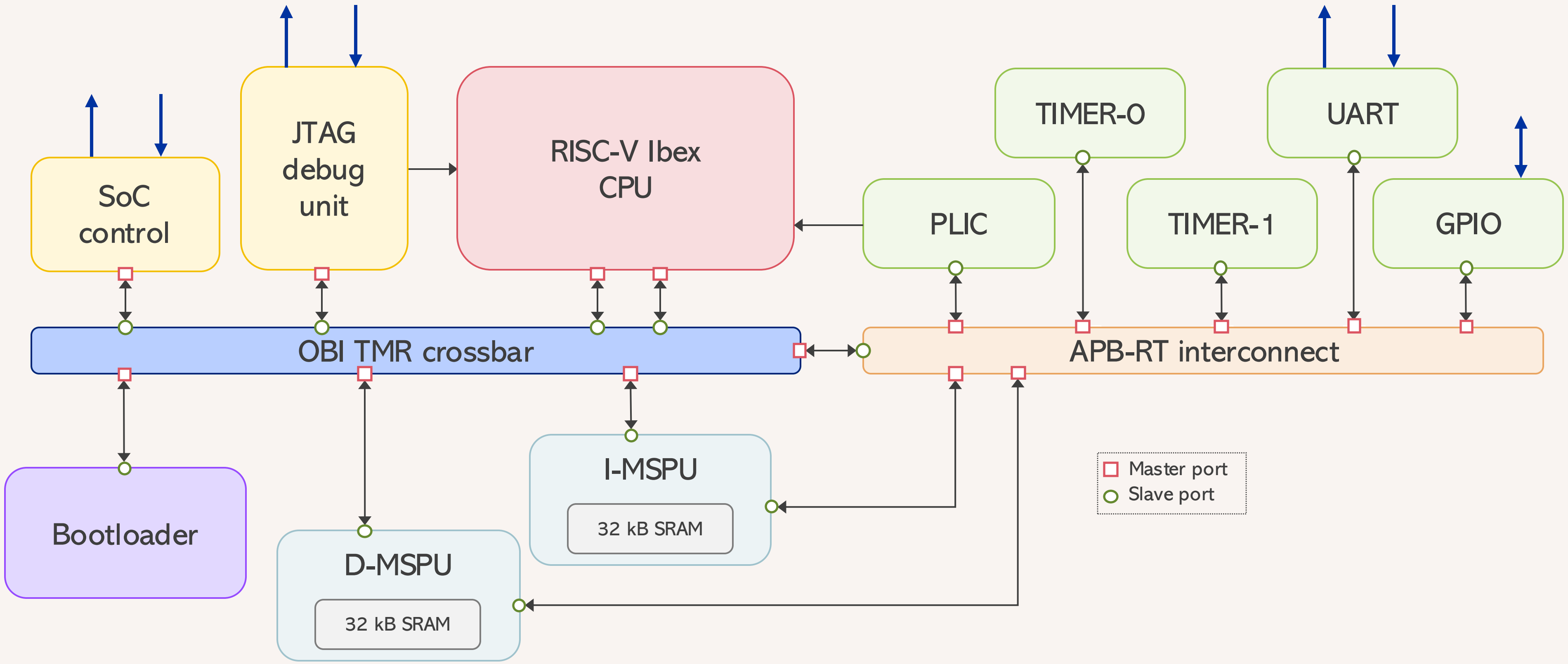
- Build a radiation-tolerant microcontroller-like SoC with the SOCRATES tools
- Validate the toolkit and demonstrate the feasibility of using programmable logic inside front-end ASICs
- Test-drive the HEP 28 nm Common Design Platform
- Test the radiation performance and gather experience on SoC design

Specifications:

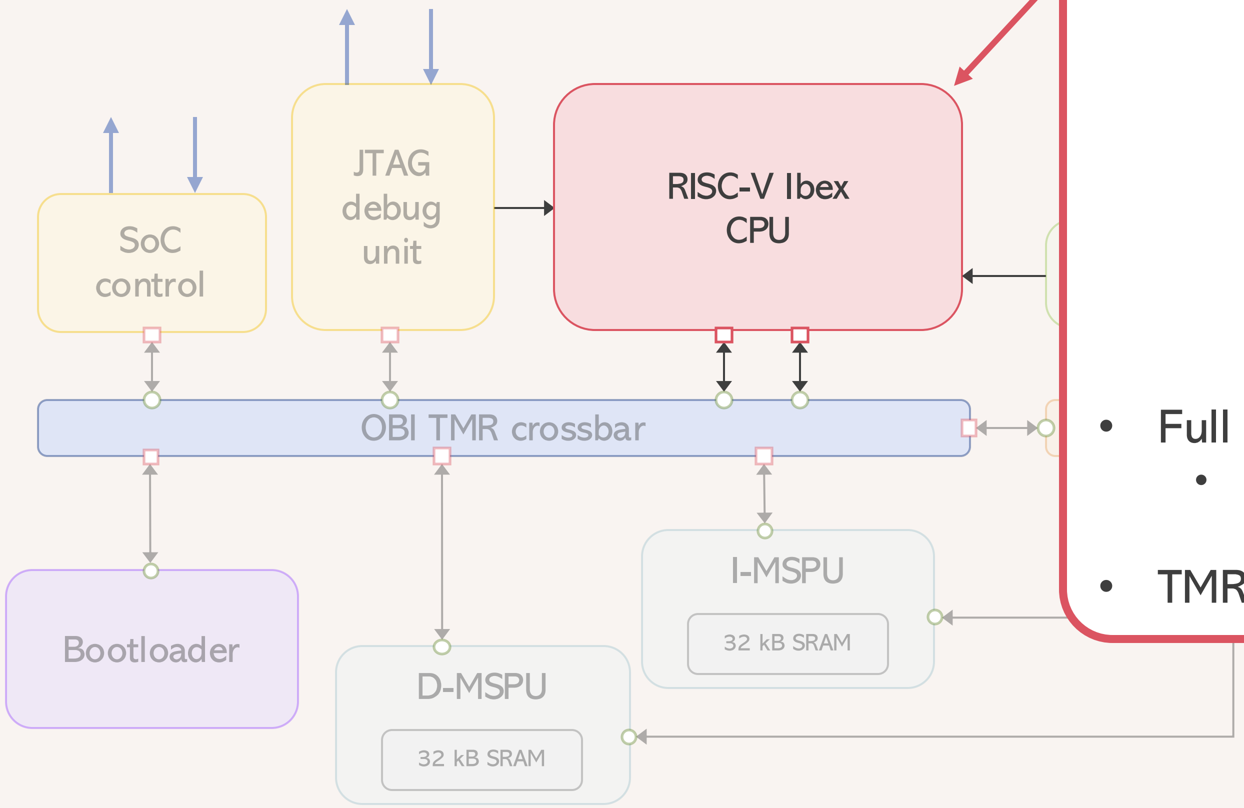
- Fully generated out of the SOCRATES platform
- Suitable peripherals and architecture for a pixel readout ASIC embedded controller applications
- Radiation-tolerant design up to ~CMS IT levels, with testability features to trace faults under irradiation
- Implemented on 28 nm bulk CMOS technology

→ Prototype tapeout on MPW scheduled for November 27th

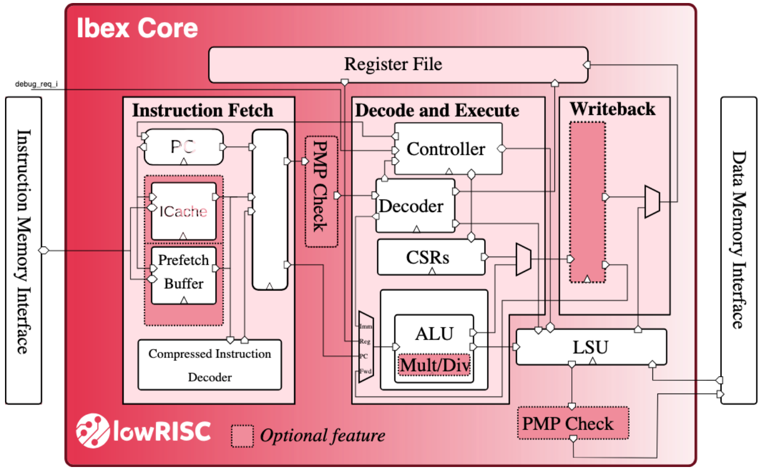
TriglaV architecture



TriglaV architecture



- Open-source high-TRL RISC-V core by lowRISC[1]
  - RV32IMC ISA
  - 2-stage pipeline



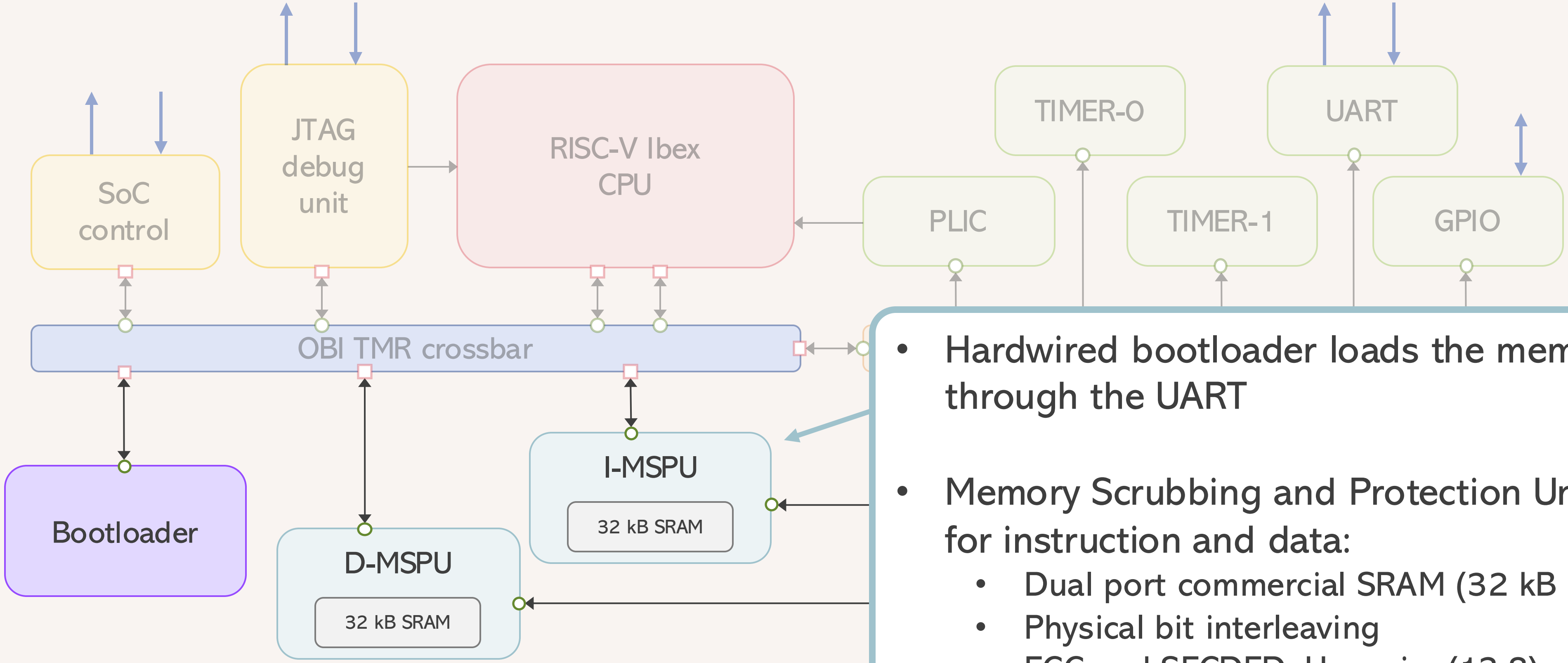
- Full TMR at FF level
  - 4x area overhead for TMR, ~6x final area with SEU observability
- TMR formal verification[2] and fault injection[3]

[1] P. Davide Schiavone et al., *Slow and steady wins the race? A comparison of ultra-low-power RISC-V69cores for Internet-of-Things applications*, PATMOS (2017) pp. 1-8.70

[2] A. Pulli, M. Lupi, *A simulation methodology for verification of transient fault tolerance of ASICs designed for high-energy physics experiments*, 2023 JINST 18 C01038

[3] A. Nookala et al., *Soft-error Reliability Analysis and Error Rate Estimation for RISC-V Processors in High Energy Physics Environments*, NSREC (2024), draft for IEEE TNS 2025

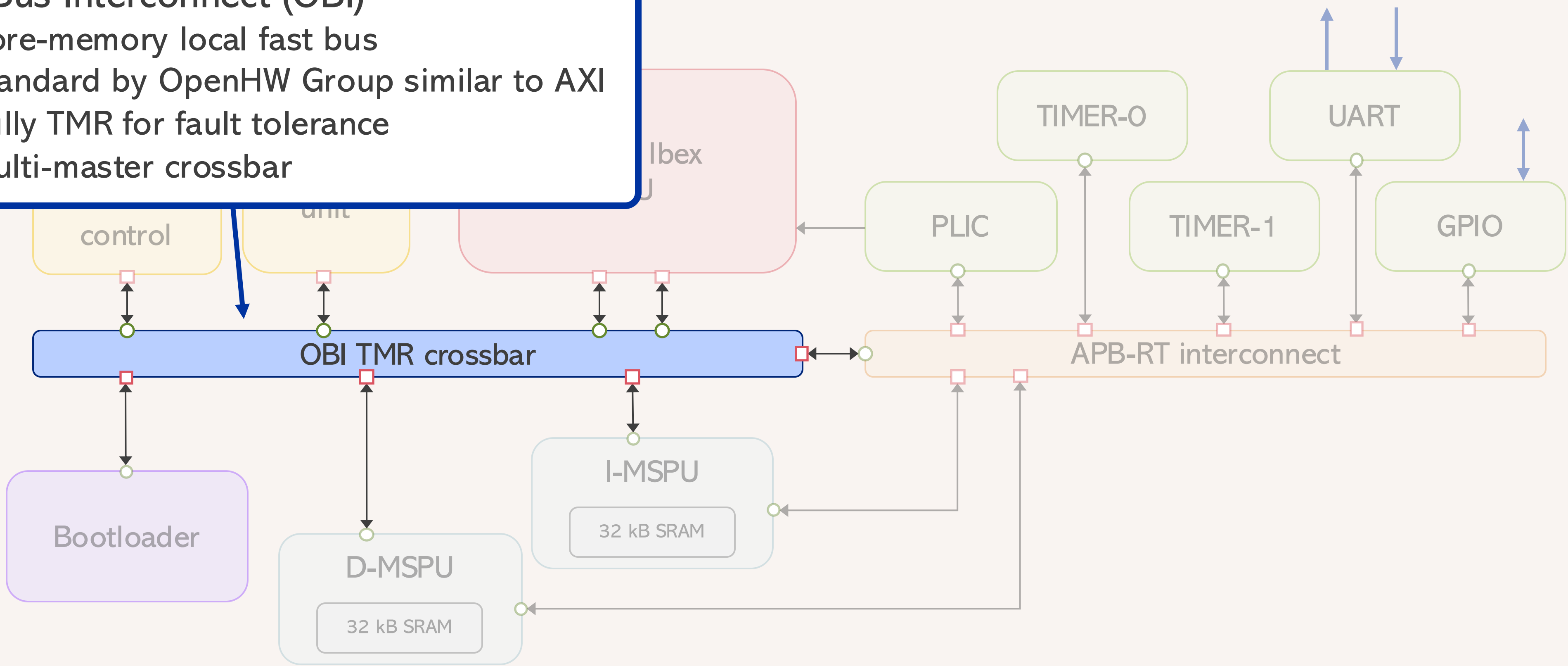
TriglaV architecture



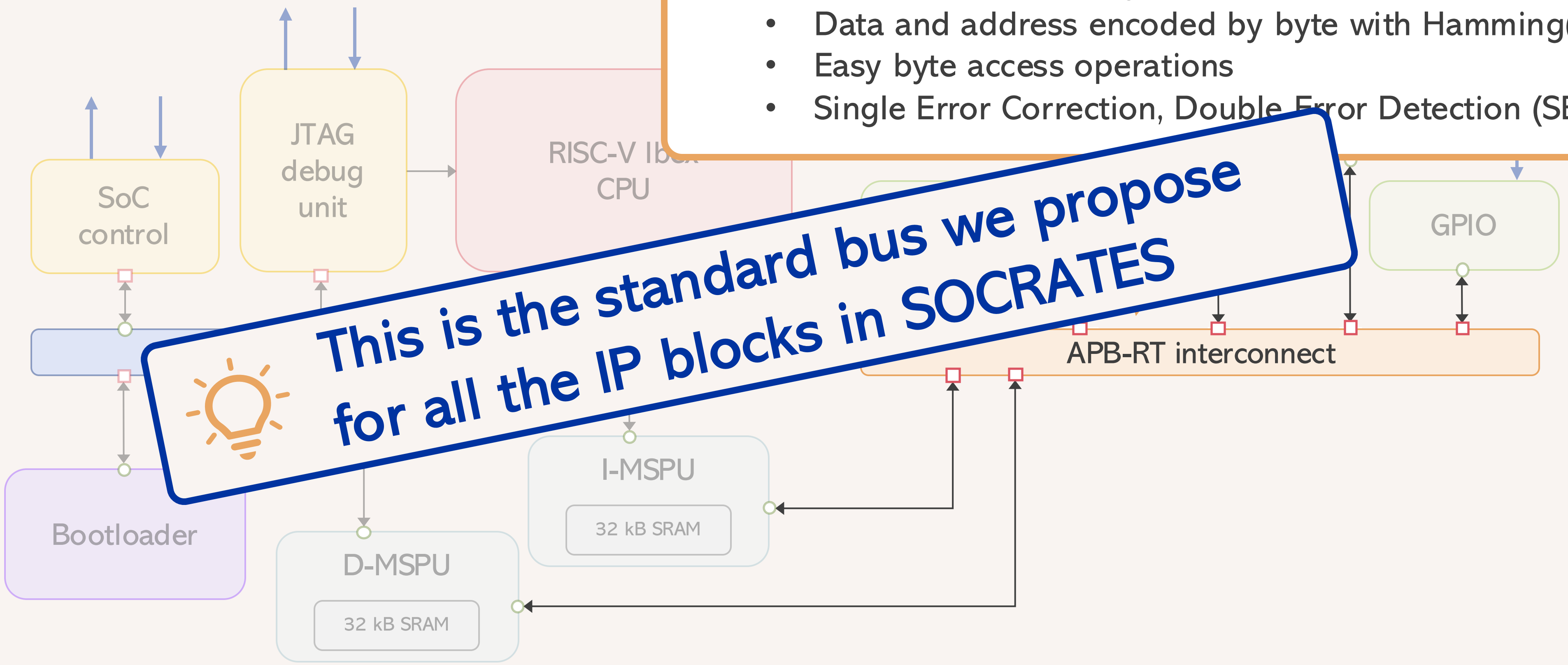
- Hardwired bootloader loads the memories through the UART
- Memory Scrubbing and Protection Unit (MSPU) for instruction and data:
  - Dual port commercial SRAM (32 kB each)
  - Physical bit interleaving
  - ECC and SECDED: Hamming(13,8) at byte-level
  - Configurable periodic scrubbing

TriglaV architecture

- Open Bus Interconnect (OBI)
  - Core-memory local fast bus
  - Standard by OpenHW Group similar to AXI
  - Fully TMR for fault tolerance
  - Multi-master crossbar



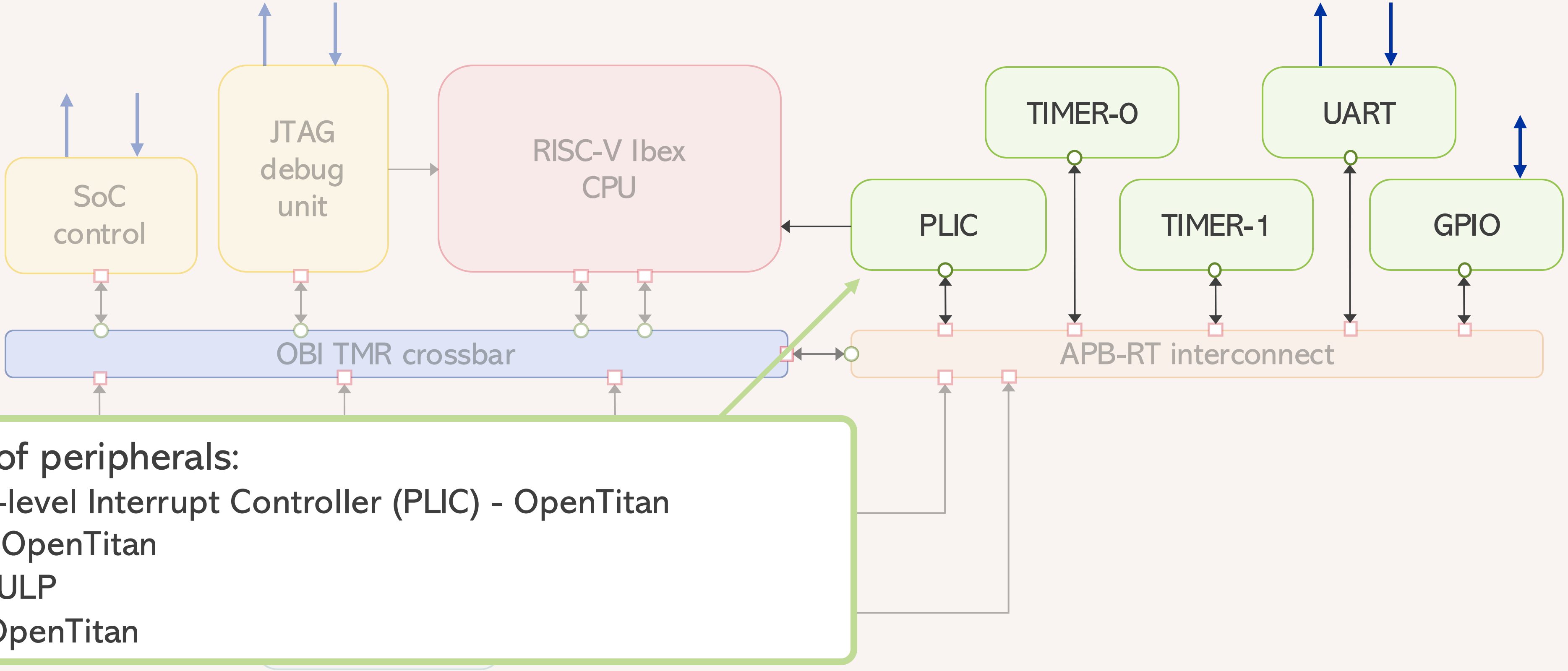
TriglaV architecture



- APB-RT
  - Peripheral bus
  - Radiation-tolerant version of the AMBA APB standard
  - Triplicated control signals
  - Data and address encoded by byte with Hamming(13,8)
  - Easy byte access operations
  - Single Error Correction, Double Error Detection (SECDED)

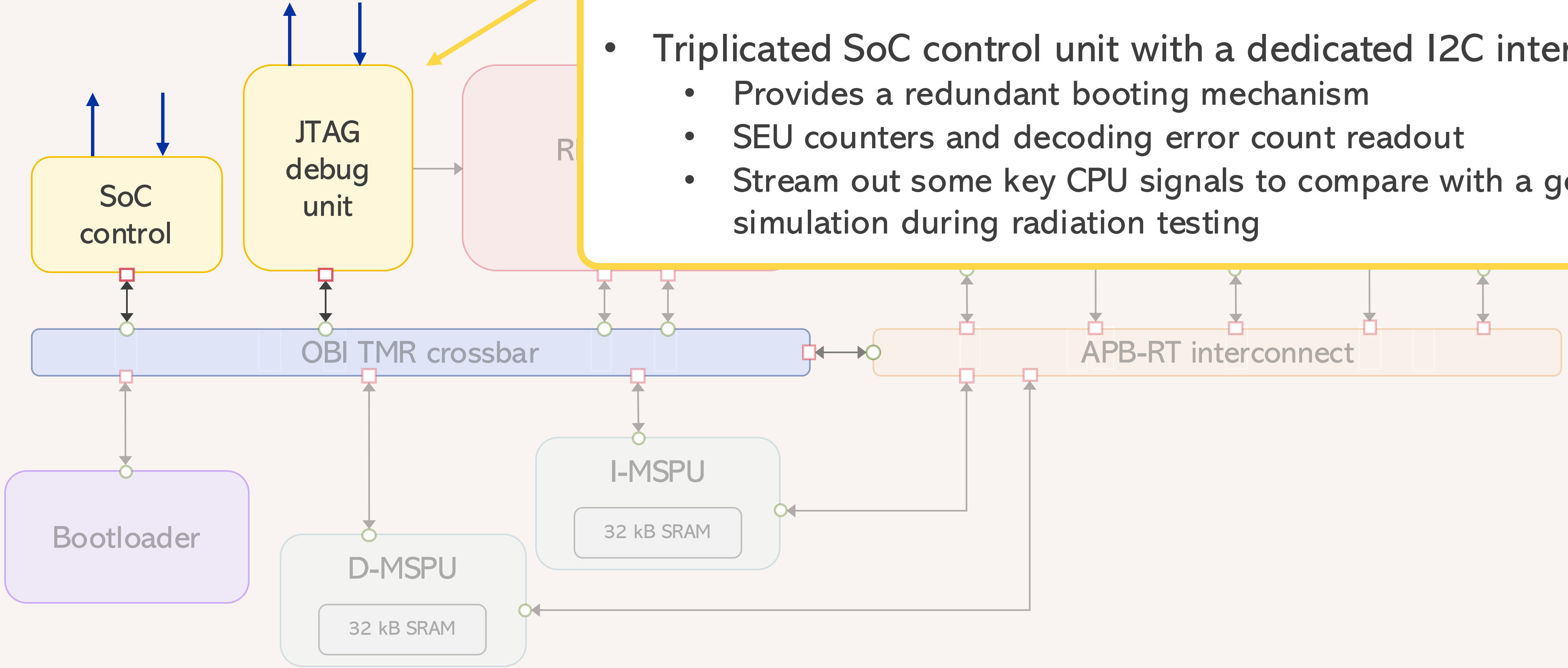
This is the standard bus we propose for all the IP blocks in SOCRATES

TriglaV architecture



- Minimal set of peripherals:
  - Platform-level Interrupt Controller (PLIC) - OpenTitan
  - Timers - OpenTitan
  - GPIO - PULP
  - UART - OpenTitan

TriglaV architecture



- JTAG port for general software debug (non-TMR)
- Triplicated SoC control unit with a dedicated I2C interface
  - Provides a redundant booting mechanism
  - SEU counters and decoding error count readout
  - Stream out some key CPU signals to compare with a golden simulation during radiation testing

What does SoCMake generate?

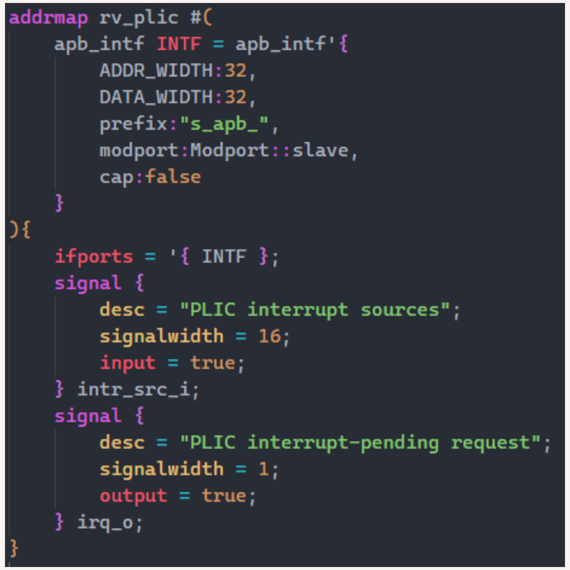
UART

```
uart.sv
uart.rdl
CMakeLists.txt
```

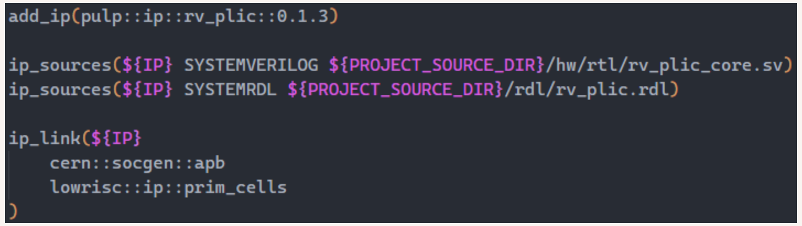
- any other IP block

PLIC

```
plic.sv
plic.rdl
CMakeLists.txt
```



```
addrmap rv_plic #(
    apb_intf INTF = apb_intf'{
        ADDR_WIDTH:32,
        DATA_WIDTH:32,
        prefix:"s_apb_",
        modport:Modport::slave,
        cap:false
    }
){
    ifports = '{ INTF };
    signal {
        desc = "PLIC interrupt sources";
        signalwidth = 16;
        input = true;
    } intr_src_i;
    signal {
        desc = "PLIC interrupt-pending request";
        signalwidth = 1;
        output = true;
    } irq_o;
}
```

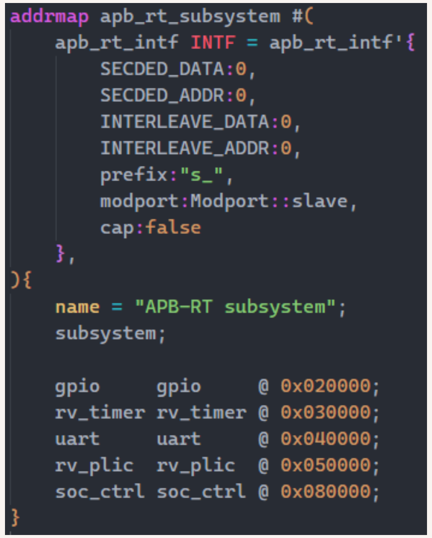


```
add_ip(pulp::ip::rv_plic::0.1.3)

ip_sources(${IP} SYSTEMVERILOG ${PROJECT_SOURCE_DIR}/hw/rtl/rv_plic_core.sv)
ip_sources(${IP} SYSTEMRDL ${PROJECT_SOURCE_DIR}/rdl/rv_plic.rdl)

ip_link(${IP}
    cern::socgen::apb
    lowrisc::ip::prim_cells
)
```

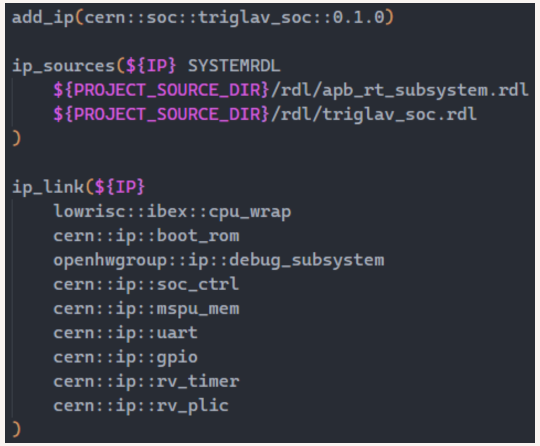
+

Top SoC

```
apb_subsystem.rdl
triglav_soc.rdl
CMakeLists.txt
```

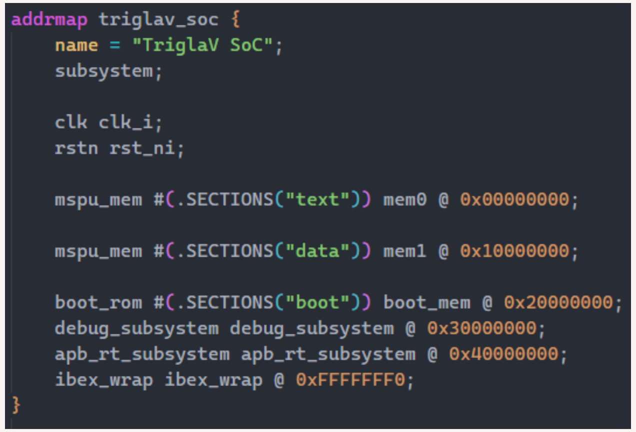


```
addrmap apb_rt_subsystem #(
    apb_rt_intf INTF = apb_rt_intf'{
        SECDED_DATA:0,
        SECDED_ADDR:0,
        INTERLEAVE_DATA:0,
        INTERLEAVE_ADDR:0,
        prefix:"s_",
        modport:Modport::slave,
        cap:false
    },
){
    name = "APB-RT subsystem";
    subsystem;

    gpio     gpio     @ 0x020000;
    rv_timer rv_timer @ 0x030000;
    uart     uart     @ 0x040000;
    rv_plic  rv_plic  @ 0x050000;
    soc_ctrl soc_ctrl @ 0x080000;
}
```



```
add_ip(cern::soc::triglav_soc::0.1.0)

ip_sources(${IP} SYSTEMRDL
    ${PROJECT_SOURCE_DIR}/rdl/apb_rt_subsystem.rdl
    ${PROJECT_SOURCE_DIR}/rdl/triglav_soc.rdl
)

ip_link(${IP}
    lowrisc::ibex::cpu_wrap
    cern::ip::boot_rom
    openhwgroup::ip::debug_subsystem
    cern::ip::soc_ctrl
    cern::ip::mspu_mem
    cern::ip::uart
    cern::ip::gpio
    cern::ip::rv_timer
    cern::ip::rv_plic
)
```

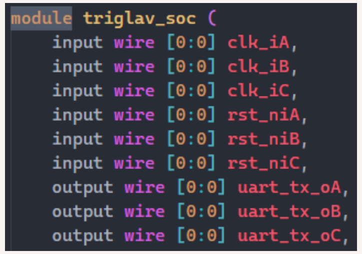
=



```
addrmap triglav_soc {
    name = "TriglaV SoC";
    subsystem;

    clk clk_i;
    rstn rst_ni;

    mspu_mem #(.SECTIONS("text")) mem0 @ 0x00000000;

    mspu_mem #(.SECTIONS("data")) mem1 @ 0x10000000;

    boot_rom #(.SECTIONS("boot")) boot_mem @ 0x20000000;
    debug_subsystem debug_subsystem @ 0x30000000;
    apb_rt_subsystem apb_rt_subsystem @ 0x40000000;
    ibex_wrap ibex_wrap @ 0xFFFFFFF0;
}
```

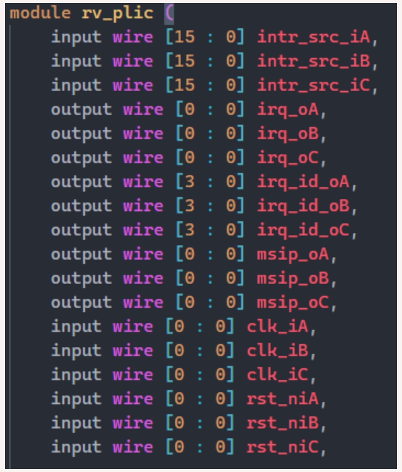
```
cmake ../
```

What you get in output from the tools



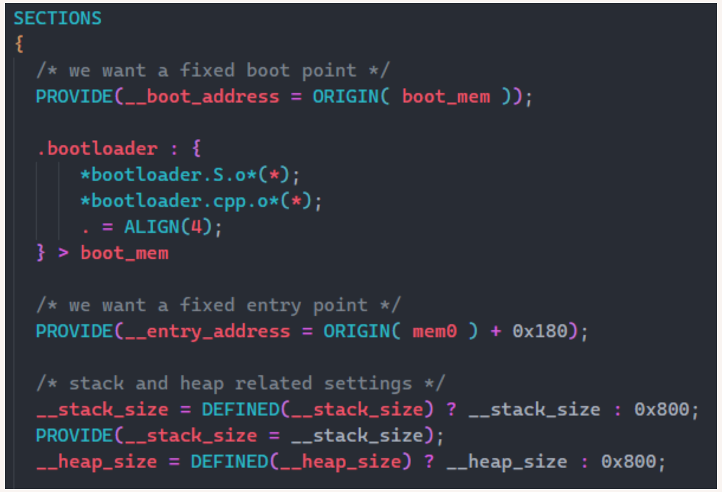
```
module triglav_soc (
    input wire [0:0] clk_iA,
    input wire [0:0] clk_iB,
    input wire [0:0] clk_iC,
    input wire [0:0] rst_niA,
    input wire [0:0] rst_niB,
    input wire [0:0] rst_niC,
    output wire [0:0] uart_tx_oA,
    output wire [0:0] uart_tx_oB,
    output wire [0:0] uart_tx_oC,
```

Top SoC Verilog



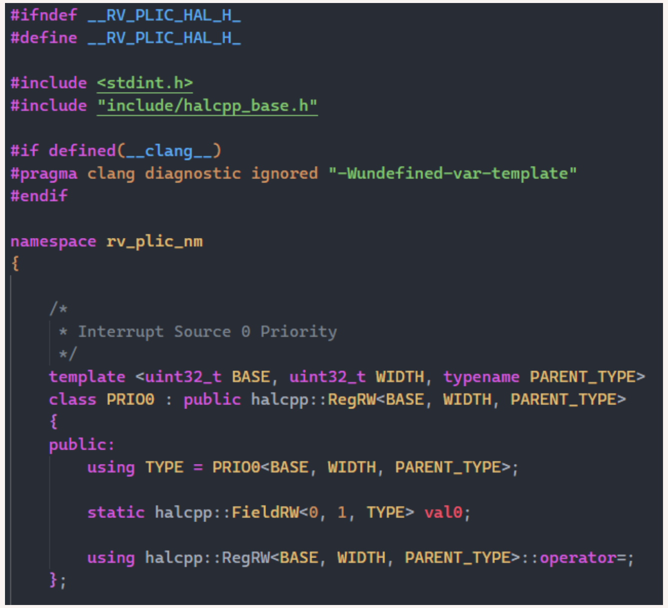
```
module rv_plic (
    input wire [15 : 0] intr_src_iA,
    input wire [15 : 0] intr_src_iB,
    input wire [15 : 0] intr_src_iC,
    output wire [0 : 0] irq_oA,
    output wire [0 : 0] irq_oB,
    output wire [0 : 0] irq_oC,
    output wire [3 : 0] irq_id_oA,
    output wire [3 : 0] irq_id_oB,
    output wire [3 : 0] irq_id_oC,
    output wire [0 : 0] msip_oA,
    output wire [0 : 0] msip_oB,
    output wire [0 : 0] msip_oC,
    input wire [0 : 0] clk_iA,
    input wire [0 : 0] clk_iB,
    input wire [0 : 0] clk_iC,
    input wire [0 : 0] rst_niA,
    input wire [0 : 0] rst_niB,
    input wire [0 : 0] rst_niC,
```

TMR peripherals



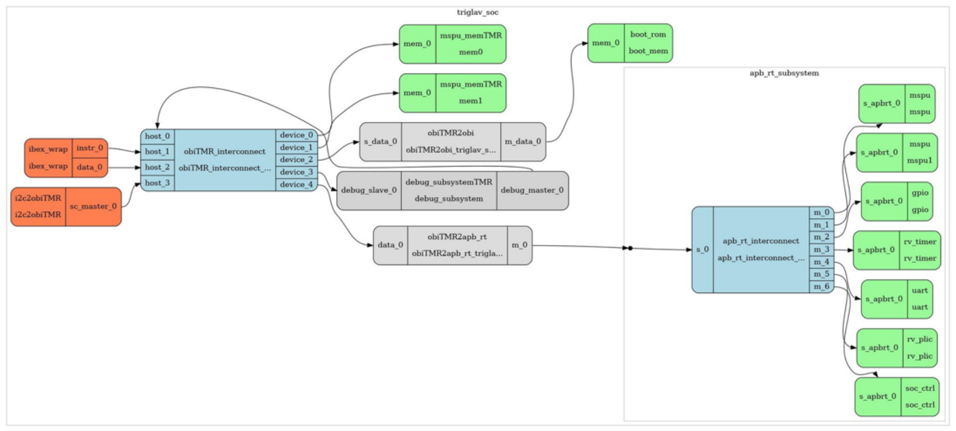
```
SECTIONS
{
  /* we want a fixed boot point */
  PROVIDE(__boot_address = ORIGIN( boot_mem ));

  .bootloader : {
      *bootloader.S.o*(*);
      *bootloader.cpp.o*(*);
      . = ALIGN(4);
  } > boot_mem

  /* we want a fixed entry point */
  PROVIDE(__entry_address = ORIGIN( mem0 ) + 0x180);

  /* stack and heap related settings */
  __stack_size = DEFINED(__stack_size) ? __stack_size : 0x800;
  PROVIDE(__stack_size = __stack_size);
  __heap_size = DEFINED(__heap_size) ? __heap_size : 0x800;
```

Linker script



```
#ifndef __RV_PLIC_HAL_H_
#define __RV_PLIC_HAL_H_

#include <stdint.h>
#include "include/halcpp_base.h"

#if defined(__clang__)
#pragma clang diagnostic ignored "-Wundefined-var-template"
#endif

namespace rv_plic_nm
{

    /*
     * Interrupt Source 0 Priority
     */
    template <uint32_t BASE, uint32_t WIDTH, typename PARENT_TYPE>
    class PRIO0 : public halcpp::RegRW<BASE, WIDTH, PARENT_TYPE>
    {
    public:
        using TYPE = PRIO0<BASE, WIDTH, PARENT_TYPE>;

        static halcpp::FieldRW<0, 1, TYPE> val0;

        using halcpp::RegRW<BASE, WIDTH, PARENT_TYPE>::operator=;
    };
```

Peripheral HAL



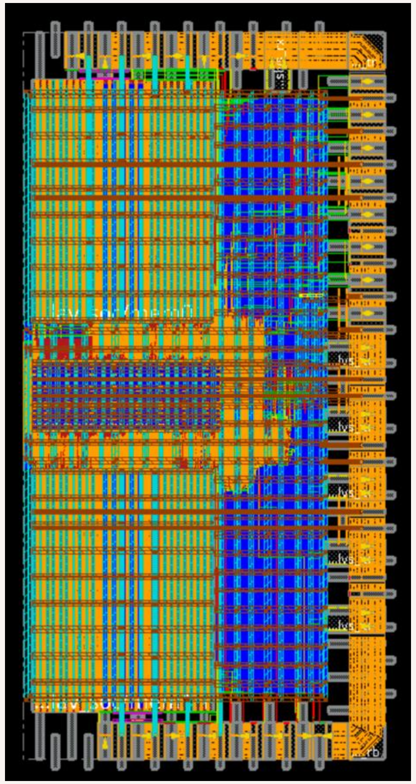
A nice block diagram

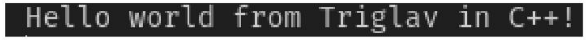
TriglaV design status

- Physical implementation in progress
  - Area: 1 mm x 2 mm
  - ~300k instances
  - Target operating frequency: 320 MHz

- FPGA emulation setup (credits to Georgios Nikolaidis)
  - ~10^3 faster than simulation
  - Increased confidence in the RTL to real HW

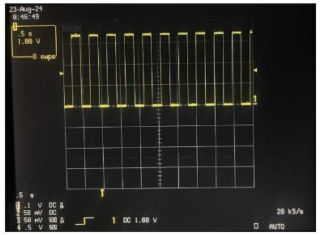
- Functional verification
  - Random verification of peripherals
  - Simulation of the top level





```
Hello world from Triglav in C++!
```

Messages from TriglaV printed in UART console



GPIO test Results

Future outlook

1. Tape out the TriglaV prototype and test its radiation performance to get a baseline

2. Evaluate alternative fault-tolerance solutions (e.g. selective triplication, core lockstepping) for better/multiple PPA/radiation resistance tradeoff

3. Polish the toolkit to make it more user friendly and open source it

4. Demonstrate the use of a similar SoC embedded in a pixel readout chip (e.g Picopix)

Thank you!

CERN

home.cern