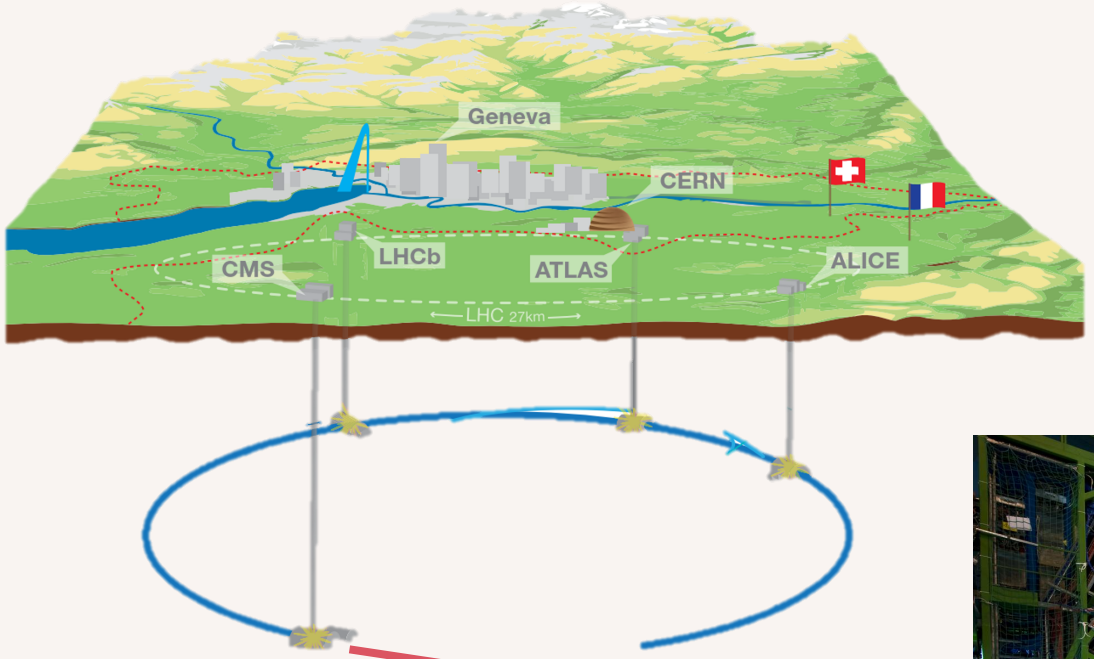
EP-ESE
ELECTRONIC SYSTEMS FOR EXPERIMENTS

WP5 IC Tech
EP R&D

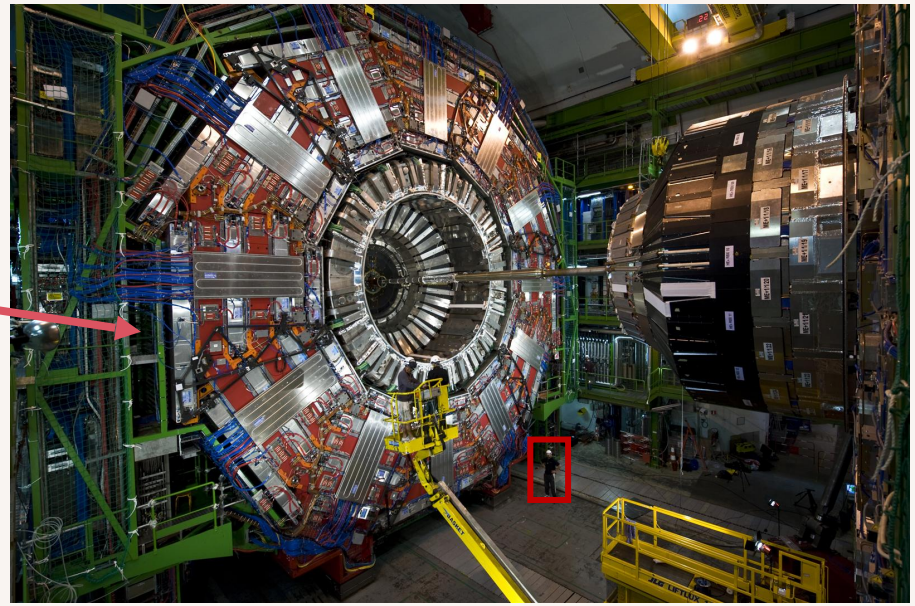
Automate Fault-Tolerant SoC Generation with the SOCRATES Platform

Marco Andorno (marco.andorno@cern.ch)

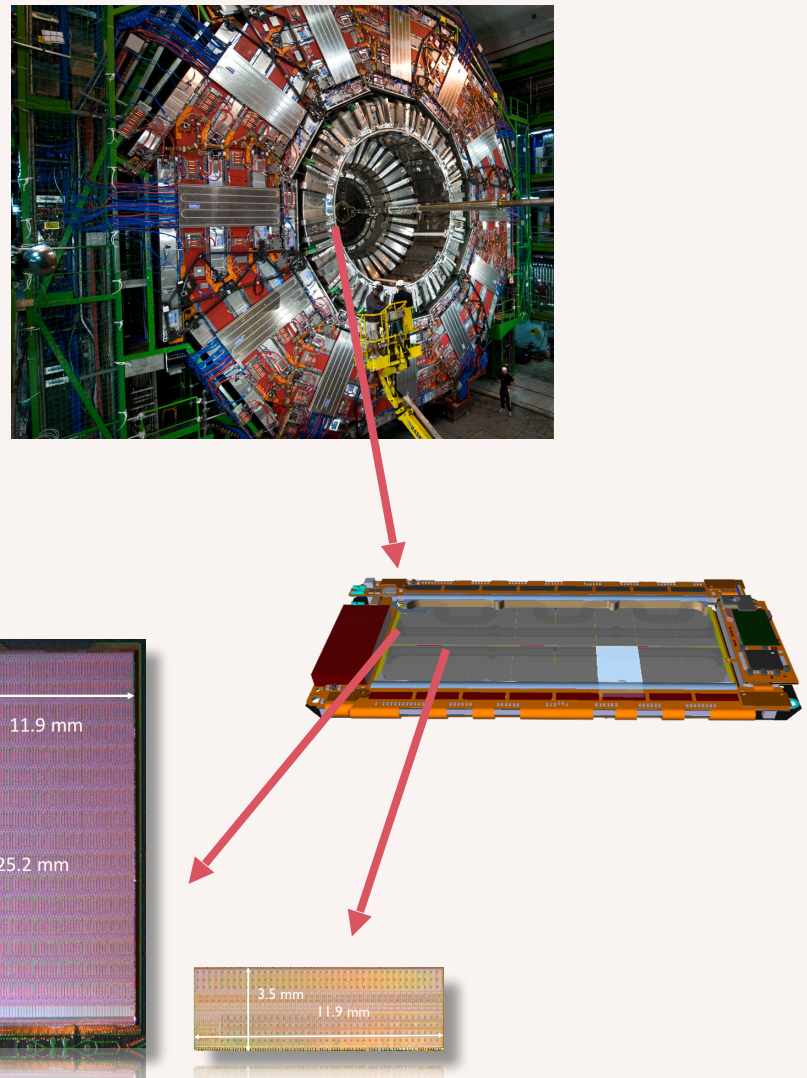
CERN and High Energy Physics (HEP) Experiments



- Accelerate and collide particles (e.g. protons) at almost the speed of light
- Take 40 million “pictures” per second of their interactions

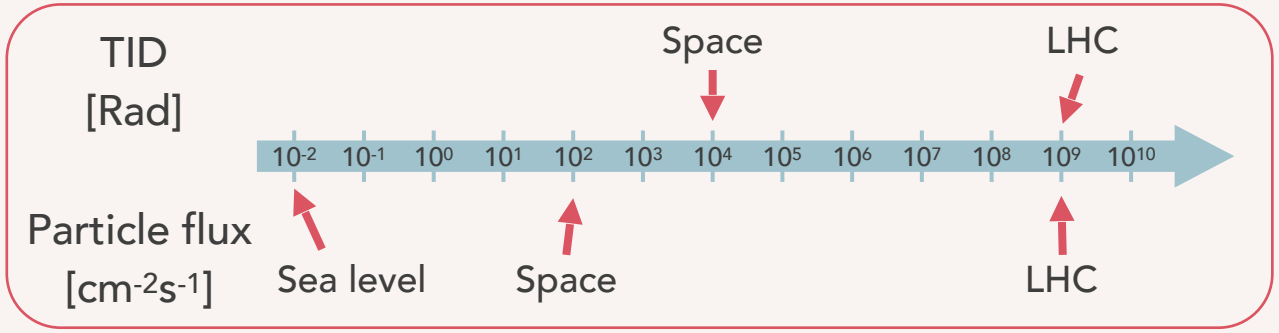


ASICs for HEP



Demanding constraints:

- Limited material and power budget
- Long lifespan of components
- Extreme radiation tolerance:



Very specific applications:

- Many fast data readout channels
- Unique signal processing techniques

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

Turn our custom ASICs into programmable fault-tolerant SoCs to get:

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

… but a single architecture can't fit all our applications, so…

SoC RAdiation Tolerant Eco-System

SOCRATES

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

- Unified build system for HW and SW
- Library of radiation-tolerant verified IP blocks
- Fault-tolerance support

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

The SOCRATES platform

HW/SW build system
(SoCMake)

Pre-verified
IP block library

DEVELOPMENT STAGE:
- ✓ Advanced
- ● In progress
- ○ Future developments

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 fault tolerance add-ons

HW/SW build system (SoCMake)

Pre-verified IP block library

Fault tolerance

DEVELOPMENT STAGE:
- ✓ Advanced
- ● In progress
- ○ Future developments

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

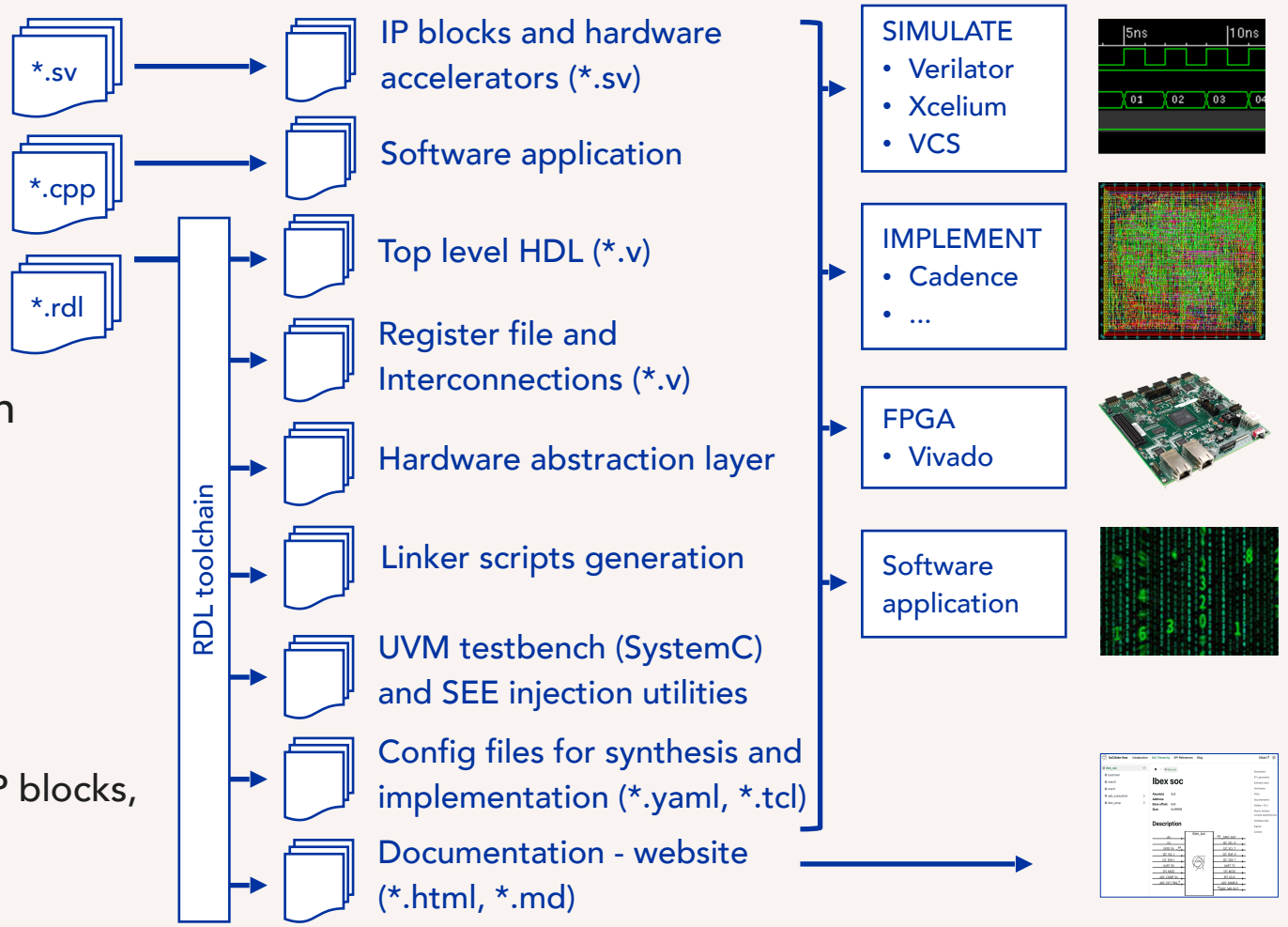
- ✓ Functional testing environment
- ● SystemC-UVM environment
- ✓ FPGA prototyping
- ✓ Fault injection framework
- ✓ Formal verification for TMR

Implementation

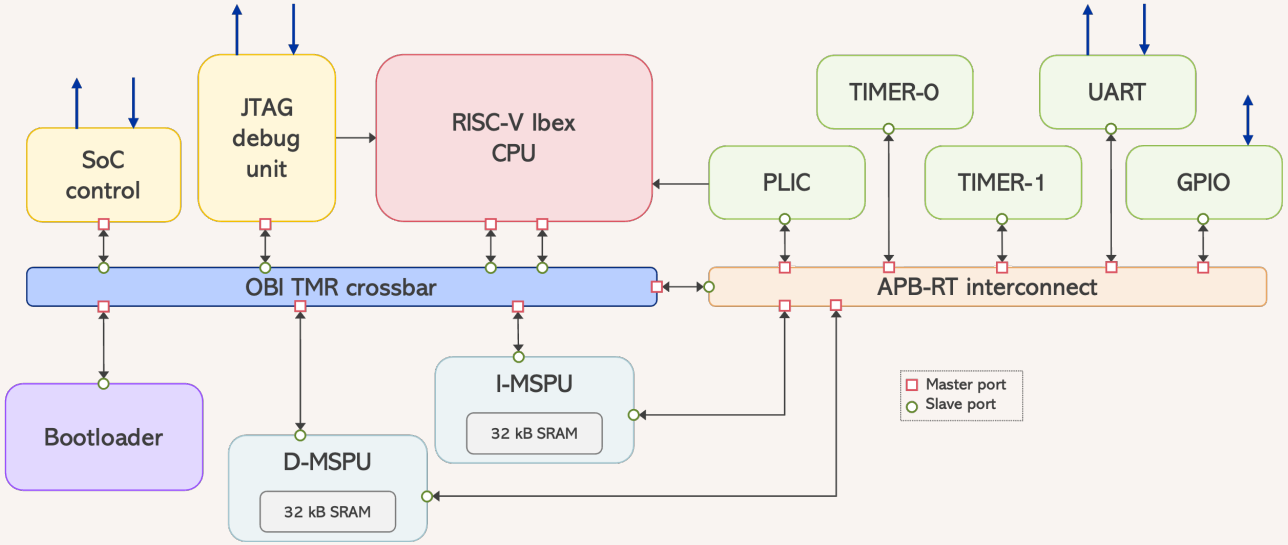
- ● Run synthesis and physical implementation flow on targets
- ○ Generate constraints
- ○ Constraints for fault tolerance

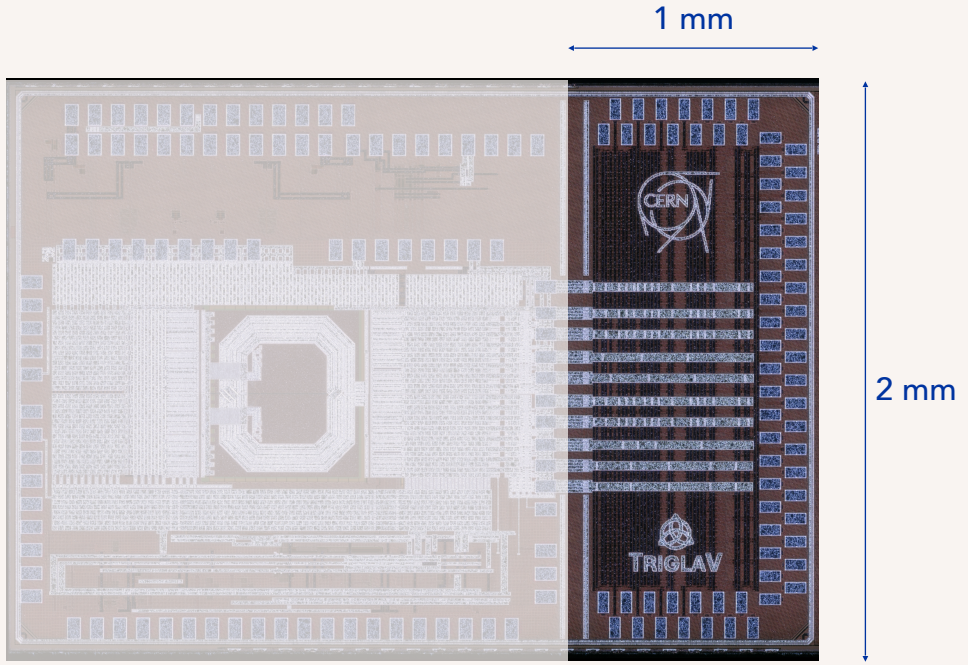
SoCMake

- **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**
  - github.com/HEP-SoC/SoCMake



TriglaV: a silicon demonstrator for SOCRATES





- Commercial 28 nm bulk CMOS technology, 2 mm², 250 MHz
- Full TMR Ibex core
- ECC protected memory and peripheral bus
- Redundant booting mechanisms
- Error counters, dedicated debug outputs

What does SoCMake generate?

```
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;
}
```

UART

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

any other IP block

PLIC

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

```
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

```
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

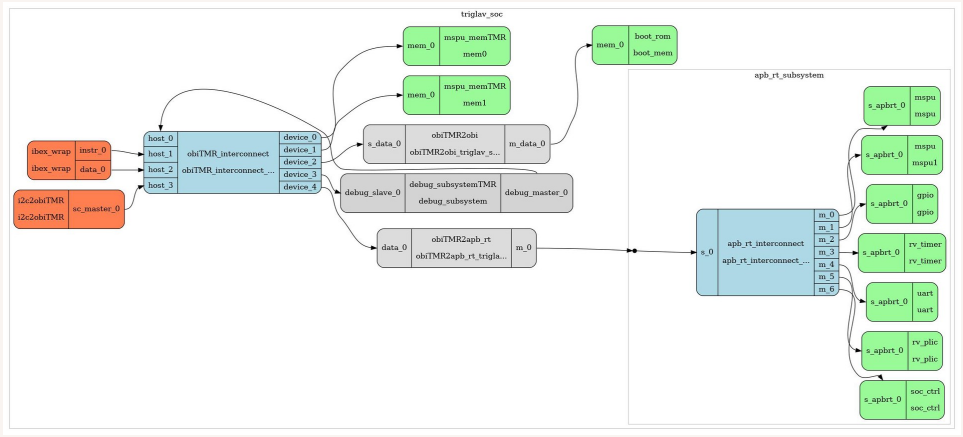
```
#ifndef __RV_PLIC_HAL_H_
#define __RV_PLIC_HAL_H_

#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

=

```
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



A nice block diagram

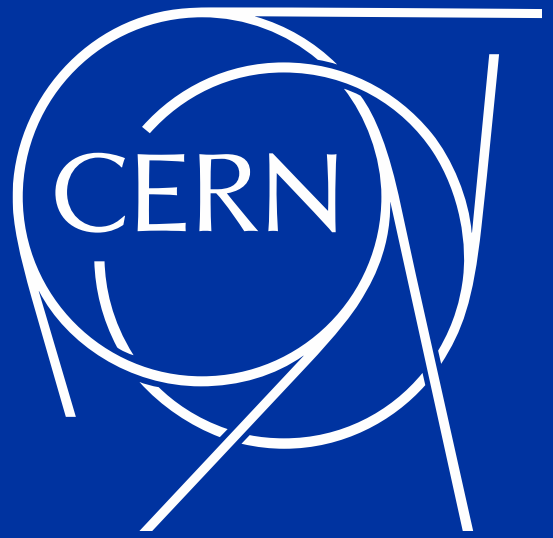
Future developments

1. Test the TriglaV prototype and its radiation performance to get a baseline

2. Polish the toolkit to make it more user friendly and get the community involved

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

Thank you!



home.cern