PULP PLATFORM

Open Source Hardware, the way it should be!

PULP Simulator and SDK

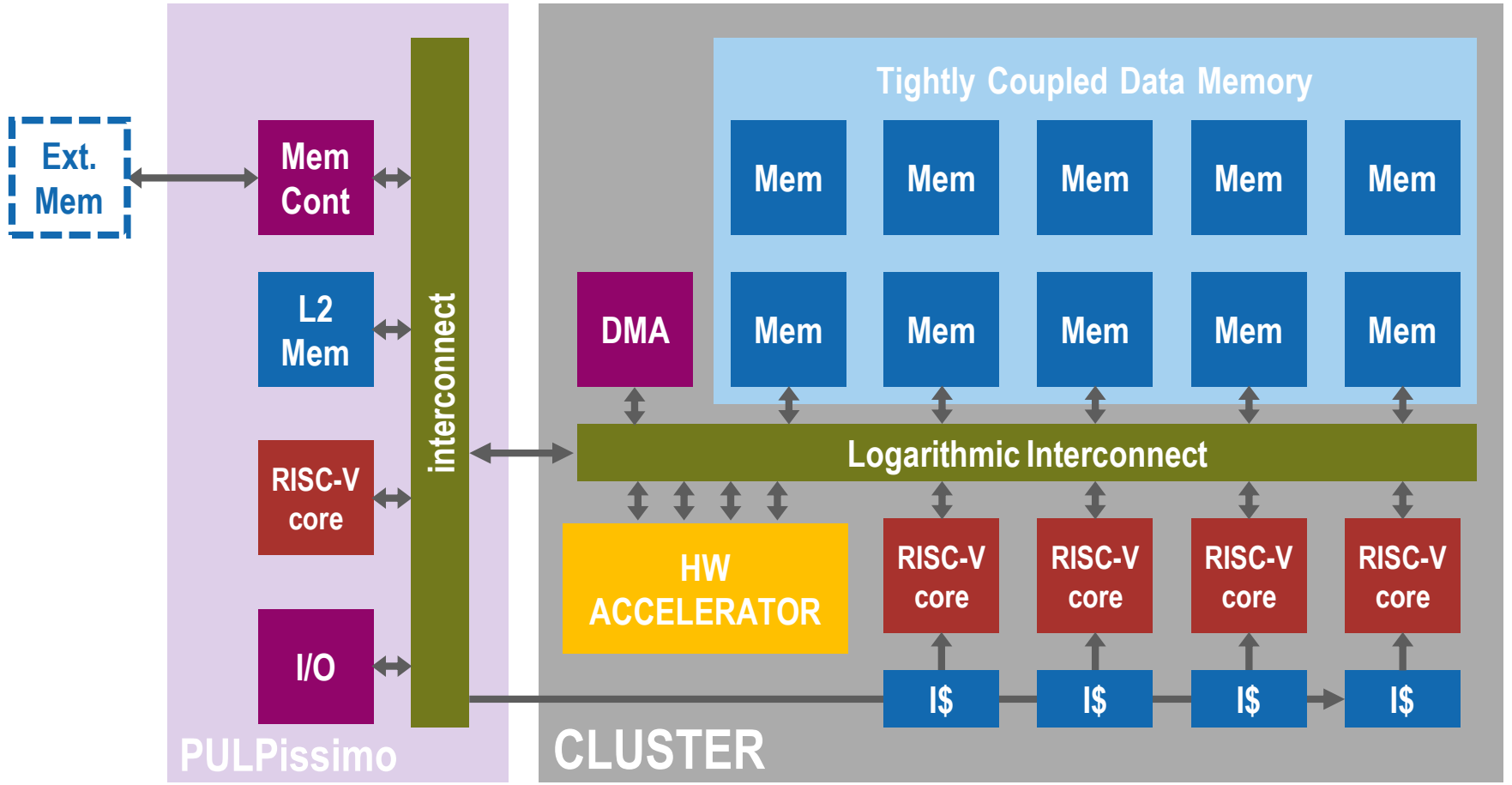
Nazareno Bruschi <nazareno.bruschi@unibo.it>

Germain Haugou <germain.haugou@iis.ee.ethz.ch>

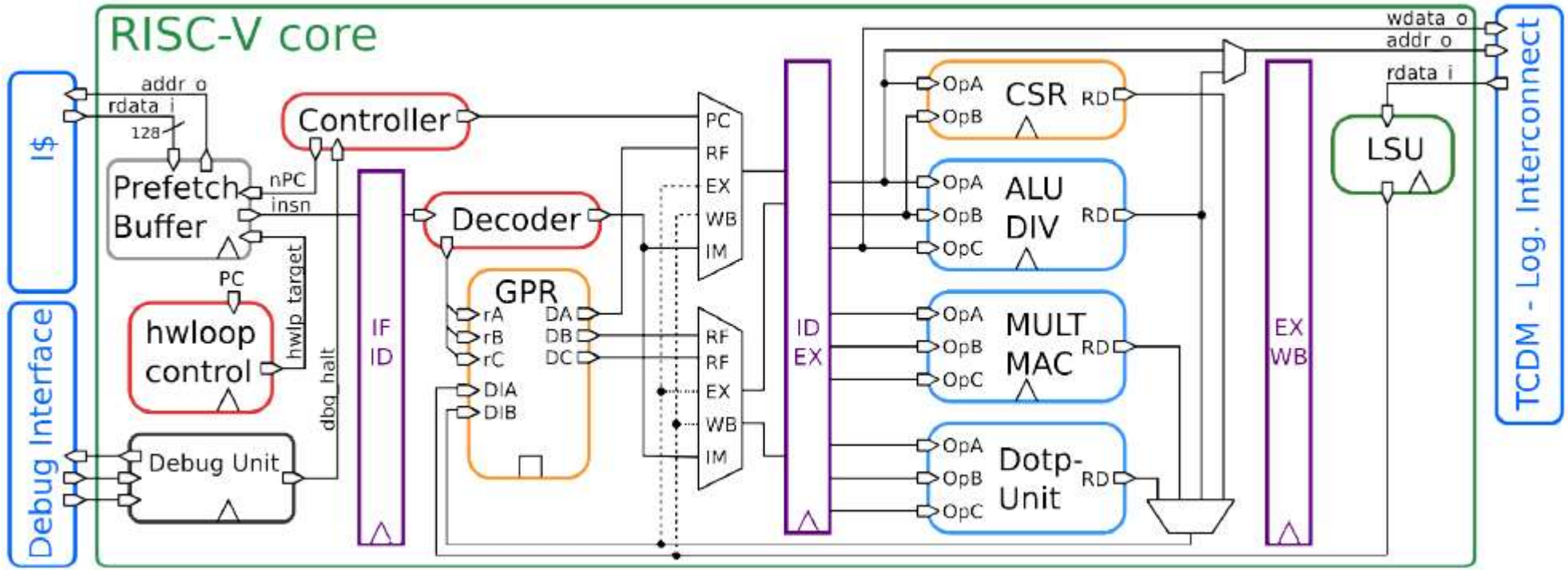
ETH zürich

 http://pulp-platform.org  @pulp_platform  https://www.youtube.com/pulp_platform

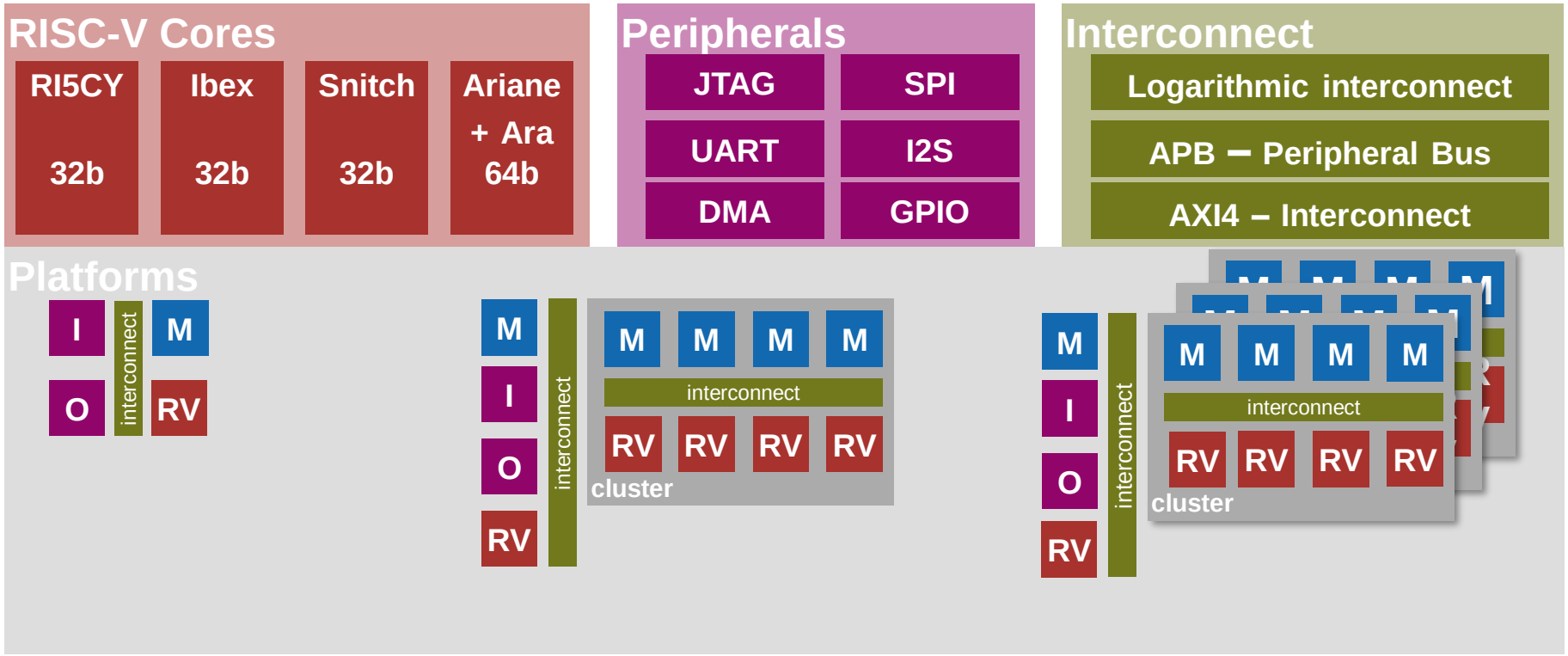
Pulp Architecture



RISCY and PULP Toolchain



GVSoC – PULP Simulator



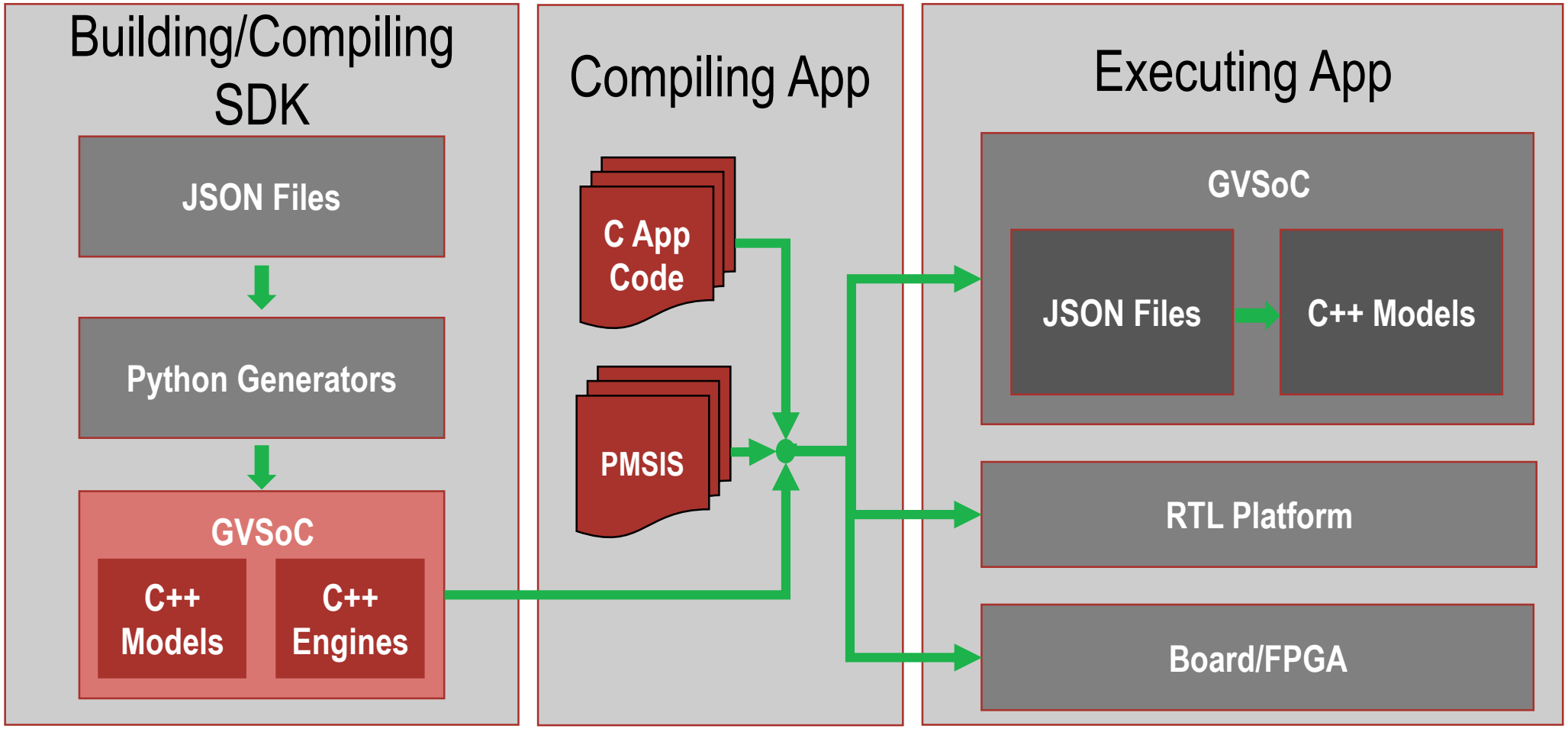
Potentially could simulate every PULP Platforms, building blocks thanks to its Instruction Set Simulator based on RISCV and PULP Extensions

GVSoC – Features

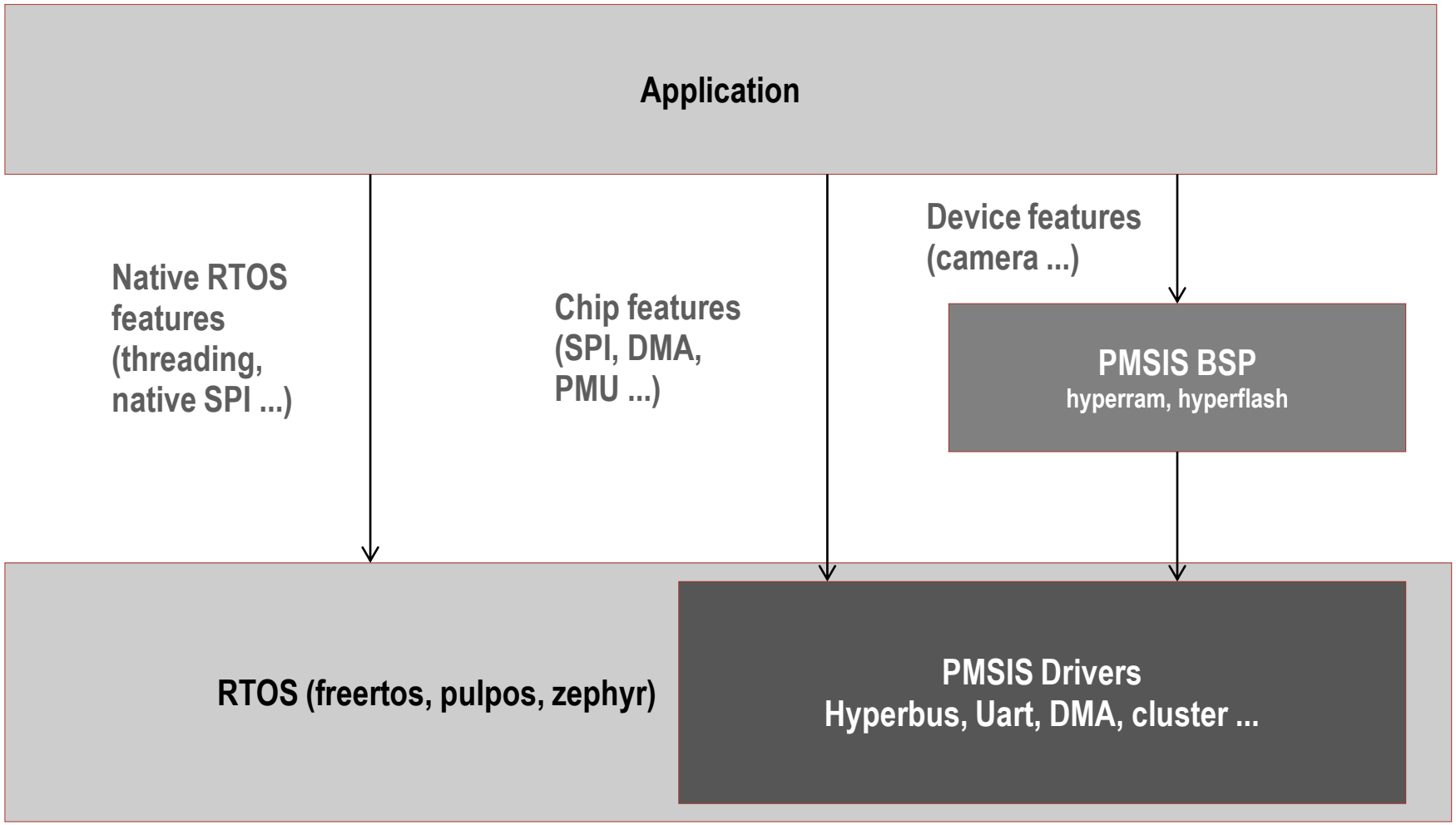
- **Virtual platform features:**
  - *C++* for fast native simulation
  - *Python* for instantiation + JSON for configuration
  - Complete set of traces to see what happen
- **Timing model:**
  - Fully-event based, instances can generate events at specific time
  - Includes timing models for interconnects, DMACs, memories…
  - Performance counters for information from the execution
- **Simulation performance:**
  - Around 1MIPS simulation speed
  - Functionally aligned and calibrated with HW
  - Timing accuracy is within 10-20% of target HW

Many more details in gvsoc documentation https://gvsoc.readthedocs.io/en/latest/ about the concepts (not updated for all the commands)

PULP Software Environment



PMSIS – PULP Runtime



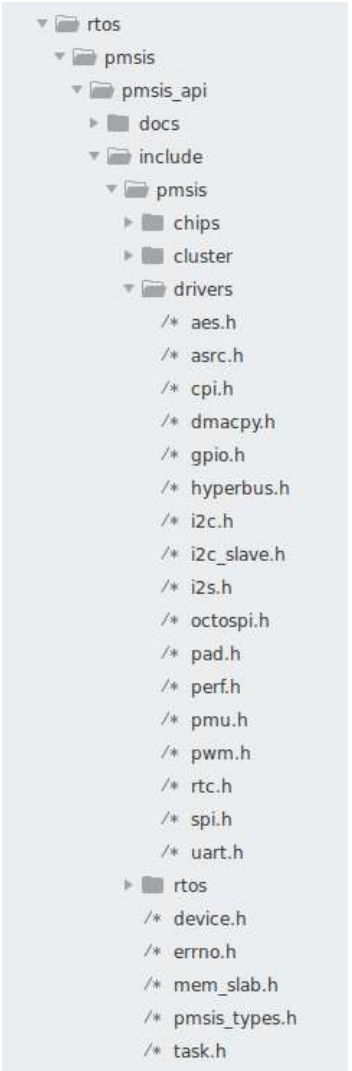
PULP-SDK – Directory organization

- rtos/
  - pmsis
  - pulpos
- tools/
  - gap-configs
  - gapy
  - gvsoc
- tests/
- applications/

- Contains runtime code and every functions of the software stack
- Contains configuration files, python generators, pulp runner and every gvsoc model and components
- Contains small examples to test few basic pulp features on GVSoC
- Contains relevant example applications such as MobileNetV1

PMSIS APIs – Documentation

- A documentation at GreenWaves-Technologies manuals web page (https://greenwaves-technologies.com/manuals/BUILD/PMSIS_API/html/md__home_yao_gap_sdk_rtos_pmsis_pmsis_api_docs_mainpage.html)
- APIs description and functionalities are also briefly explained in header files, located in the rtos/pmsis/pmsis_api/include/pmsis



Requirements – PULP Toolchain

- PULP toolchain is available at https://github.com/pulp-platform/pulp-riscv-gnu-toolchain
- On Ubuntu 18.04 this packages should be required and you can install them with:
  - $ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev

Installation – PULP Toolchain

- To install the PULP toolchain follow these steps:
  - $ git clone https://github.com/pulp-platform/pulp-riscv-gnu-toolchain
  - $ cd pulp-riscv-gnu-toolchain
  - $ git submodule update --init --recursive
  - $ export PATH=<INSTALL_DIR>/bin:$PATH
  - $ ./configure --prefix=<INSTALL_DIR> --with-arch=rv32imc --with-cmodel=medlow --enable-multilib
  - $ make
- More details at https://github.com/pulp-platform/pulp-riscv-gnu-toolchain

Requirements – PULP-SDK

- PULP-SDK is available at https://github.com/pulp-platform/pulp-sdk
- On Ubuntu 18.04 this packages should be required and you can install them with:
  - $ sudo apt-get install -y build-essential git libftdi-dev libftdi1 doxygen python3-pip libsdl2-dev curl cmake libusb-1.0-0-dev scons gtkwave libsndfile1-dev rsync autoconf automake texinfo libtool pkg-config libsdl2-ttf-dev
- You may have needed of other python3 packages and you can install them with:
  - $ pip install argcomplete pyelftools six

Installation – PULP-SDK

- To install the PULP-SDK follow these steps:
  - $ git clone https://github.com/pulp-platform/pulp-sdk
  - $ export PULP_RISCV_GCC_TOOLCHAIN=<INSTALL_DIR>
  - $ cd pulp-sdk
  - $ source configs/pulp-open.sh
  - $ make build
- More details at **https://github.com/pulp-platform/pulp-sdk**

Setup the environment for GVSoC execution

- Assuming that PULP Toolchain and PULP-SDK are correctly built and installed
  - $ cd pulp-sdk/
- Setting up PULP Toolchain path
  - $ export PULP_RISCV_GCC_TOOLCHAIN=<INSTALL_DIR>
- Setting up the environment sourcing a configuration file
  - $ source configs/pulp-open.sh
- Build and compile GVSoC (if target or GVSoC is changed)
  - $ make build

Run first simple test: An Hello from PULP!

```c
#include "pmsis.h"

#if defined(CLUSTER)
void pe_entry(void *arg)
{
    printf("\nHello from cluster_id: %d, core_id: %d\n", pi_cluster_id(), pi_core_id());
}

void cluster_entry(void *arg)
{
    pi_cl_team_fork((NUM_CORES), pe_entry, 0);
}
#endif

static int test_entry()
{
#if defined(CLUSTER)
    struct pi_device cluster_dev;
    struct pi_cluster_conf cl_conf;
    struct pi_cluster_task cl_task;

    pi_cluster_conf_init(&cl_conf);
    pi_open_from_conf(&cluster_dev, &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        return -1;
    }

    pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_entry, NULL));

    pi_cluster_close(&cluster_dev);
#endif
#if !defined(CLUSTER)
    printf("\nHello from FC\n");
#endif

    return 0;
}

static void test_kickoff(void *arg)
{
    int ret = test_entry();
    pmsis_exit(ret);
}

int main()
{
    return pmsis_kickoff((void *)test_kickoff);
}
```

TIY: > cd tests/hello

> make clean all run VERBOSE=1

Have a look at the Makefile

Make options

```
APP = test
APP_SRCS += test.c

ifdef USE_CLUSTER
APP_CFLAGS += -DCLUSTER -DNUM_CLUSTER=$(USE_CLUSTER)
ifdef NUM_CORES
APP_CFLAGS += -DNUM_CORES=$(NUM_CORES)
else
APP_CFLAGS += -DNUM_CORES=1
endif
endif

APP_CFLAGS += -Os -g
APP_LDFLAGS += -Os -g

include $(RULES_DIR)/pmsis_rules.mk
```

Compiler flags

Rules to make target

Run first simple test: An Hello from PULP!

```c
#include "pmsis.h"

#if defined(CLUSTER)
void pe_entry(void *arg)
{
    printf("\nHello from cluster_id: %d, core_id: %d\n", pi_cluster_id(), pi_core_id());
}

void cluster_entry(void *arg)
{
    pi_cl_team_fork((NUM_CORES), pe_entry, 0);
}
#endif

static int test_entry()
{
#if defined(CLUSTER)
    struct pi_device cluster_dev;
    struct pi_cluster_conf cl_conf;
    struct pi_cluster_task cl_task;

    pi_cluster_conf_init(&cl_conf);
    pi_open_from_conf(&cluster_dev, &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        return -1;
    }

    pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_entry, NULL));

    pi_cluster_close(&cluster_dev);
#endif
#if !defined(CLUSTER)
    printf("\nHello from FC\n");
#endif

    return 0;
}

static void test_kickoff(void *arg)
{
    int ret = test_entry();
    pmsis_exit(ret);
}

int main()
{
    return pmsis_kickoff((void *)test_kickoff);
}
```

Include all runtime basic functions and libraries (rtos/pulpos/common/)

An Hello from fabric controller

Run first simple test: An Hello from PULP!

```c
#include "pmsis.h"


#if defined(CLUSTER)
void pe_entry(void *arg)
{
    printf("\nHello from cluster_id: %d, core_id: %d\n", pi_cluster_id(), pi_core_id());
}

void cluster_entry(void *arg)
{
    pi_cl_team_fork((NUM_CORES), pe_entry, 0);
}
#endif

static int test_entry()
{
#if defined(CLUSTER)
    struct pi_device cluster_dev;
    struct pi_cluster_conf cl_conf;
    struct pi_cluster_task cl_task;

    pi_cluster_conf_init(&cl_conf);
    pi_open_from_conf(&cluster_dev, &cl_conf);
    if (pi_cluster_open(&cluster_dev))
    {
        return -1;
    }

    pi_cluster_send_task_to_cl(&cluster_dev, pi_cluster_task(&cl_task, cluster_entry, NULL));

    pi_cluster_close(&cluster_dev);
#endif
#if !defined(CLUSTER)
    printf("\nHello from FC\n");
#endif

    return 0;
}

static void test_kickoff(void *arg)
{
    int ret = test_entry();
    pmsis_exit(ret);
}

int main()
{
    return pmsis_kickoff((void *)test_kickoff);
}
```

An Hello from every core

Task fork on settable number of cores

Cluster call, offload and close

TIY: > cd tests/hello

> make clean all run USE_CLUSTER=1 NUM_CORES=8

Disassembled – Show the real code

- $ cd tests/hello
- $ make clean all
- $ make dis > test.s

```
Disassembly of section .text:

1c00809c <test_kickoff>:
1c00809c:   1c001537            lui a0,0x1c001
1c0080a0:   1141                    addi    sp,sp,-16
1c0080a2:   86850513            addi    a0,a0,-1944 # 1c000868 <__DTOR_END__>
1c0080a6:   c606                    sw  ra,12(sp)
1c0080a8:   2a8d                    jal 1c00821a <puts>
1c0080aa:   1c0017b7            lui a5,0x1c001
1c0080ae:   4501                    li  a0,0
1c0080b0:   0c07a023            sw  zero,192(a5) # 1c0010c0 <_edata>
1c0080b4:   2259                    jal 1c00823a <exit>

1c0080b6 <main>:
1c0080b6:   1141                    addi    sp,sp,-16
1c0080b8:   c606                    sw  ra,12(sp)
1c0080ba:   37cd                    jal 1c00809c <test_kickoff>
```

TIY: > cd tests/hello

> make clean all USE_CLUSTER=1 CORES=8

> make dis > test.s

System Traces – What is it doing?

- $ cd tests/hello
- $ make clean all run runner_args="--trace=<PATH>:log.txt"
- If <PATH>=.*, every trace will be dumped in BUILD/PULP/GCC_RISCV/log.txt file. HUGE!

/sys/board/chip/**<PATH>**	**Description**
cluster/pe0	Processing element, useful to see the IOs made by the core, and the instruction it executes. You can add */iss* to just get instruction events
cluster/event_unit	Hardware synchronizer events, useful for debugging inter-core synchronization mechanisms
cluster/pcache	Shared program cache accesses
cluster/l1_ico	Shared L1 interconnect
cluster/l1/bankX	L1 memory banks (the X should be replaced by the bank number)
soc/l2	L2 memory accesses
cluster/dma	DMA events

Understanding GVSoC System Traces

- 4890000: 489: [/sys/board/chip/soc/cluster/pe0/insn] M 1c001252 p.sw 0, 4(a5!) a5=10000010 a5:1000000c PA:1000000c
- <timestamp> <cycles> <path> <address> <instruction> <operands> <operands info>
- **Where:**
  - <timestamp> is the timestamp of the event in picoseconds
  - <cycles> is the number of cycles
  - <path> is the path in the architecture where the event occurred
  - <address> is the address of the instruction
  - <instruction> is the instruction label
  - <operands> is the part of the decoded operands
  - <operands info> is giving details about the operands values and how they are used
- **The timestamp is absolute. The cycle count is local to the frequency domain**

Understanding GVSoC System Traces

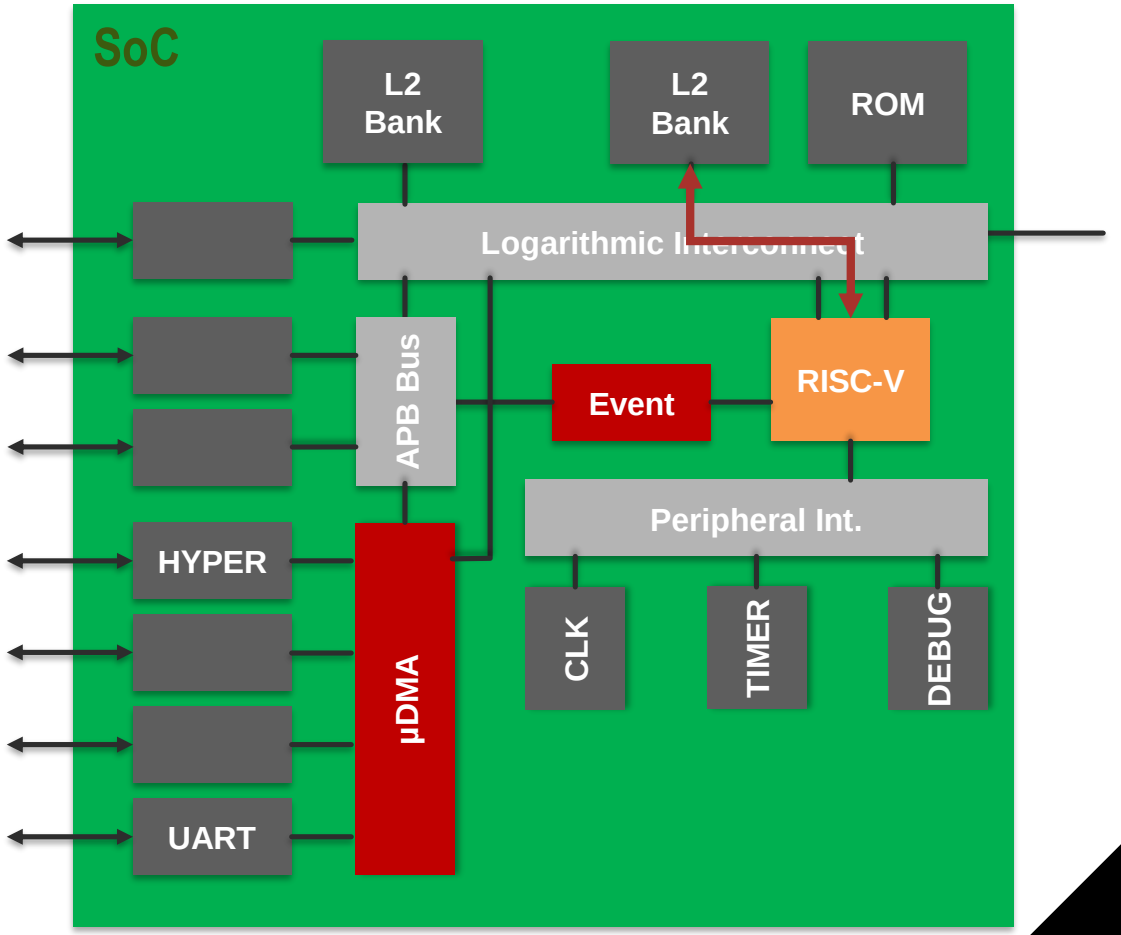
- 4890000: 489: [/sys/board/chip/soc/cluster/pe0/insn] M 1c001252 p.sw 0, 4(a5!) a5=10000010 a5:1000000c PA:1000000c
- <timestamp> <cycles> <path> <address> <instruction> <operands> <operands info>

TIY: > cd tests/hello

> make clean all run runner_args="--trace=insn"

A little bit complex: Vector x Vector

- Compute the partial output results in the core, taking the input data from L2 memory and then store them back in L2



Performance counters

```c
int main()
{
    // initialize matrix operands
    task_initMat();

#ifdef PRINT_MATRIX
    printf("\n\nThis is the Matrix A\n");
    print_matrix(A, N);
    printf("\n\nThis is the Matrix B\n");
    print_matrix(B, N);
#endif
#ifndef STATS
    //initialize performance counters
    pi_perf_conf(
        1 << PI_PERF_CYCLES |
        1 << PI_PERF_INSTR
    );

    // measure statistics on matrix operations
    pi_perf_reset();
    pi_perf_start();
#else
    INIT_STATS();

    PRE_START_STATS();
    START_STATS();
#endif

    for(int i=0; i<N;i++){
        task_VectProdScalar(A[i], B, tempC, N);
    }

#ifndef STATS
    pi_perf_stop();
    uint32_t instr_cnt = pi_perf_read(PI_PERF_INSTR);
    uint32_t cycles_cnt = pi_perf_read(PI_PERF_CYCLES);

    printf("Number of Instructions: %d\nClock Cycles: %d\nCPI: %f%f\n",
        instr_cnt, cycles_cnt, (float) cycles_cnt/instr_cnt);
#else
    STOP_STATS();
#endif
}
```

Select type of performance to measure

Start the counting

Stop and read

Performance counters

```
int main()
{
    // initialize matrix operands
    task_initMat();

#ifdef PRINT_MATRIX
    printf("\n\nThis is the Matrix A\n");
    print_matrix(A, N);
    printf("\n\nThis is the Matrix B\n");
    print_matrix(B, N);
#endif
#ifndef STATS
    //initialize performance counters
    pi_perf_conf(
        1 << PI_PERF_CYCLES |
        1 << PI_PERF_INSTR
    );

    // measure statistics on matrix operations
    pi_perf_reset();
    pi_perf_start();
#else
    INIT_STATS();

    PRE_START_STATS();
    START_STATS();
#endif

    for(int i=0; i<N;i++){
        task_VectProdScalar(A[i], B, tempC, N);
    }

#ifndef STATS
    pi_perf_stop();
    uint32_t instr_cnt = pi_perf_read(PI_PERF_INSTR);
    uint32_t cycles_cnt = pi_perf_read(PI_PERF_CYCLES);

    printf("Number of Instructions: %d\nClock Cycles: %d\nCPI: %f%f\n",
        instr_cnt, cycles_cnt, (float) cycles_cnt/instr_cnt);
#else
    STOP_STATS();
#endif
}
```

TIY: > cd tests/perf/matmult
> make clean all run

Application to evaluate

Performance counters

```
typedef enum {
  PI_PERF_CYCLES        = 17, /*!< Total number of cycles (also includes the
    cycles where the core is sleeping). Be careful that this event is using a
    timer shared within the cluster, so resetting, starting or stopping it on
    one core will impact other cores of the same cluster. */
  PI_PERF_ACTIVE_CYCLES = 0,  /*!< Counts the number of cycles the core was
    active (not sleeping). */
  PI_PERF_INSTR         = 1,  /*!< Counts the number of instructions executed.
  */
  PI_PERF_LD_STALL      = 2,  /*!< Number of load data hazards. */
  PI_PERF_JR_STALL      = 3,  /*!< Number of jump register data hazards. */
  PI_PERF_IMISS         = 4,  /*!< Cycles waiting for instruction fetches, i.e.
    number of instructions wasted due to non-ideal caching. */
  PI_PERF_LD            = 5,  /*!< Number of data memory loads executed.
    Misaligned accesses are counted twice. */
  PI_PERF_ST            = 6,  /*!< Number of data memory stores executed.
    Misaligned accesses are counted twice. */
  PI_PERF_JUMP          = 7,  /*!< Number of unconditional jumps (j, jal, jr,
    jalr). */
  PI_PERF_BRANCH        = 8,  /*!< Number of branches. Counts both taken and
    not taken branches. */
  PI_PERF_BTAKEN        = 9, /*!< Number of taken branches. */
  PI_PERF_RVC           = 10, /*!< Number of compressed instructions
    executed. */
  PI_PERF_LD_EXT        = 12, /*!< Number of memory loads to EXT executed.
    Misaligned accesses are counted twice. Every non-TCDM access is considered
    external (cluster only). */
  PI_PERF_ST_EXT        = 13, /*!< Number of memory stores to EXT executed.
    Misaligned accesses are counted twice. Every non-TCDM access is considered
    external (cluster only). */
  PI_PERF_LD_EXT_CYC    = 14, /*!< Cycles used for memory loads to EXT.
  Every non-TCDM access is considered external (cluster only). */
  PI_PERF_ST_EXT_CYC    = 15, /*!< Cycles used for memory stores to EXT.
  Every non-TCDM access is considered external (cluster only). */
  PI_PERF_TCDM_CONT     = 16, /*!< Cycles wasted due to TCDM/log-interconnect
  contention (cluster only). */
} pi_perf_event_e;
```

/rtos/pmsis/pmsis_api/include/pmsis/chips/default.h

Real chips have only 1 counter to be activated at the same time while other platforms could have one per event

Performance counters

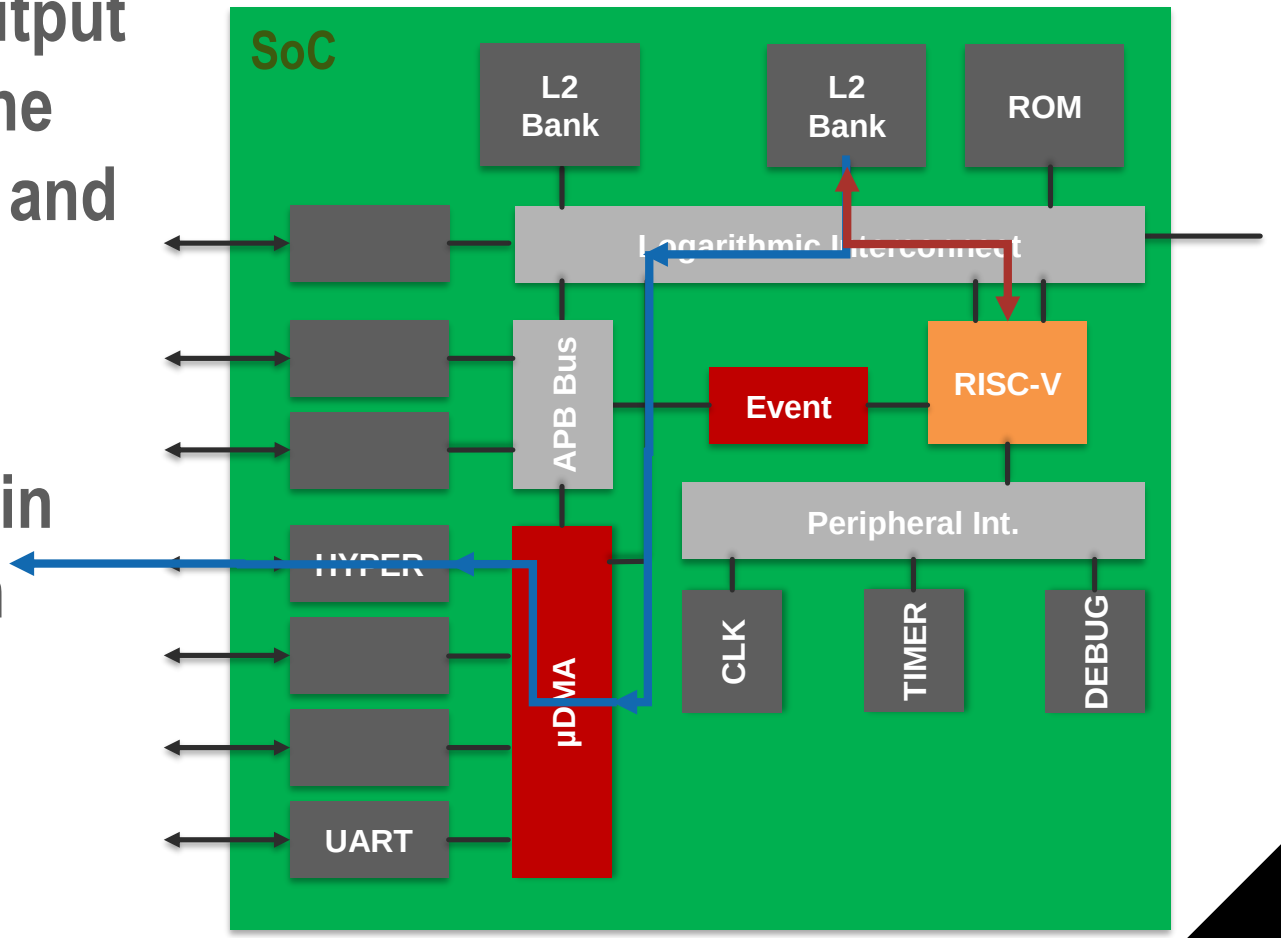
```c
int main()
{
    // initialize matrix operands
    task_initMat();

#ifdef PRINT_MATRIX
    printf("\n\nThis is the Matrix A\n");
    print_matrix(A, N);
    printf("\n\nThis is the Matrix B\n");
    print_matrix(B, N);
#endif
#ifndef STATS
    //initialize performance counters
    pi_perf_conf(
        1 << PI_PERF_CYCLES |
        1 << PI_PERF_INSTR
    );

    // measure statistics on matrix operations
    pi_perf_reset();
    pi_perf_start();
#else
    INIT_STATS();

    PRE_START_STATS();
    START_STATS();
#endif

    for(int i=0; i<N;i++){
        task_VectProdScalar(A[i], B, tempC, N);
    }

#ifndef STATS
    pi_perf_stop();
    uint32_t instr_cnt = pi_perf_read(PI_PERF_INSTR);
    uint32_t cycles_cnt = pi_perf_read(PI_PERF_CYCLES);

    printf("Number of Instructions: %d\nClock Cycles: %d\nCPI: %f%f\n",
        instr_cnt, cycles_cnt, (float) cycles_cnt/instr_cnt);
#else
    STOP_STATS();
#endif
}
```

TIY: > cd tests/perf/matmult
> make clean all run VERBOSE_PERF=1

SPOILER: Load stalls are the problem of the differences between instructions and cycles!

A little bit complex: Ram Transfers

- First compute the partial output results in the core, taking the input data from L2 memory and then store them back in L2
- Copy partial output results in HYPERRAM, going through the LIC, µDMA and then HyperBus



Ram Transfers – Sync or Async

```
#ifndef DOUBLE_BUFFERING
    for(int i=0; i<N_WORD;i++){
        task_VectProdScalar(A[i], B, tempC, N_WORD);
        pi_ram_write(&ram, hyper_buff+i*N_BYTE, tempC, (uint32_t) N_BYTE);
    }
#else
    int i_curr=1;
    int i_prev=0;
    int buffer_id;
    task_VectProdScalar(A[0], B, tempC, N_WORD);
    for(i_curr; i_curr<N_WORD;i_curr++){
        buffer_id = i_curr & 0x1;
        pi_ram_write_async(&ram, hyper_buff+i_prev*N_BYTE, &tempC[N_WORD*(1-buffer_id)], (uint32_t) N_BYTE, pi_task_callback(&ram_write_tasks[i_prev], end_of_tx, NULL));
        task_VectProdScalar(A[i_curr], B, &tempC[N_WORD*buffer_id], N_WORD);
        i_prev++;
    }
    pi_ram_write_async(&ram, hyper_buff+i_prev*N_BYTE, &tempC[N_WORD*buffer_id], (uint32_t) N_BYTE, pi_task_callback(&ram_write_tasks[i_prev], end_of_tx, NULL));

    while(ram_returns != i_curr) {
        pi_yield();
    }
#endif
```

synchronous

asynchronous

```
static struct pi_task ram_write_tasks[N];
static int count = 0;
// Callback for asynchronous ram write
static void end_of_tx(void *arg)
{
  printf("End of %d TX \n", count);
  count++;
}
```

How can we choose between them?

- Let's see again the performance

TIY: > cd tests/perf/double_buffering
> make clean all run DB=1

µDMA Messages

```
1480197236: 85826: [34m/sys/board/chip/soc/udma/trace                [0m] Writing clock-enable register (current_value: 0x2, new_value: 0x0)
1480197236: 85826: [34m/sys/board/chip/soc/udma/trace                [0m] Dectivating periph (periph: 1)
3613852428: 87173: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x0, size: 0x4, is_write: 0)
3614086176: 87185: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x0, size: 0x4, is_write: 1)
3614086176: 87185: [34m/sys/board/chip/soc/udma/trace                [0m] Writing clock-enable register (current_value: 0x0, new_value: 0x80)
3614086176: 87185: [34m/sys/board/chip/soc/udma/trace                [0m] Activating periph (periph: 7)
3615644496: 87265: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x608, size: 0x4, is_write: 1)
3615839286: 87275: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x620, size: 0x4, is_write: 1)
3616053555: 87286: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x60c, size: 0x4, is_write: 1)
3616267824: 87297: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x604, size: 0x4, is_write: 1)
3618780615: 87426: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x620, size: 0x4, is_write: 1)
3619053321: 87440: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x624, size: 0x4, is_write: 0)
3619326027: 87454: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x428, size: 0x4, is_write: 1)
3619501338: 87463: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x42c, size: 0x4, is_write: 1)
3619676649: 87472: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x430, size: 0x4, is_write: 1)
3619851960: 87481: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x434, size: 0x4, is_write: 1)
3620027271: 87490: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x438, size: 0x4, is_write: 1)
3620222061: 87500: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x43c, size: 0x4, is_write: 1)
3620533725: 87516: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x418, size: 0x4, is_write: 1)
3620533725: 87516: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing CA setup register (value: 0x1, id: 0)
3620709036: 87525: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x41c, size: 0x4, is_write: 1)
3621118095: 87546: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x40c, size: 0x4, is_write: 1)
3621118095: 87546: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing TX start address register (value: 0x1c0011c4, id: 0)
3621312885: 87556: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x410, size: 0x4, is_write: 1)
3621312885: 87556: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing TX size register (value: 0x2, id: 0)
3621527154: 87567: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x414, size: 0x4, is_write: 1)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing TX cfg register (value: 0x10, id: 0)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0               [0m] Setting cfg register (continuous: 0, size: 8bits, enable: 1, clear: 0)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0               [0m] Fetching new request from 0 (cfg_setup: 0, nb_tran: 1)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0_tx/trace      [0m] Enqueueing new transfer (req: 0x55d2ac944820, addr: 0x1c0011c4, size: 0x2, transfer_size: 8bits, continuous: 0)
```

From the activation of the hyperbus module to the enqueueing of the first request are passed 382 µDMA cycles and 7,441 µs

µDMA Messages

```
1480197236: 85826: [34m/sys/board/chip/soc/udma/trace                [0m] Writing clock-enable register (current_value: 0x2, new_value: 0x0)
1480197236: 85826: [34m/sys/board/chip/soc/udma/trace                [0m] Dectivating periph (periph: 1)
3613852428: 87173: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x0, size: 0x4, is_write: 0)
3614086176: 87185: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x0, size: 0x4, is_write: 1)
3614086176: 87185: [34m/sys/board/chip/soc/udma/trace                [0m] Writing clock-enable register (current_value: 0x0, new_value: 0x80)
3614086176: 87185: [34m/sys/board/chip/soc/udma/trace                [0m] Activating periph (periph: 7)
3615644496: 87265: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x608, size: 0x4, is_write: 1)
3615839286: 87275: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x620, size: 0x4, is_write: 1)
3616053555: 87286: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x60c, size: 0x4, is_write: 1)
3616267824: 87297: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x604, size: 0x4, is_write: 1)
3618780615: 87426: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x620, size: 0x4, is_write: 1)
3619053321: 87440: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x624, size: 0x4, is_write: 0)
3619326027: 87454: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x428, size: 0x4, is_write: 1)
3619501338: 87463: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x42c, size: 0x4, is_write: 1)
3619676649: 87472: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x430, size: 0x4, is_write: 1)
3619851960: 87481: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x434, size: 0x4, is_write: 1)
3620027271: 87490: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x438, size: 0x4, is_write: 1)
3620222061: 87500: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x43c, size: 0x4, is_write: 1)
3620533725: 87516: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x418, size: 0x4, is_write: 1)
3620533725: 87516: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing CA setup register (value: 0x1, id: 0)
3620709036: 87525: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x41c, size: 0x4, is_write: 1)
3621118095: 87546: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x40c, size: 0x4, is_write: 1)
3621118095: 87546: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing TX start address register (value: 0x1c0011c4, id: 0)
3621312885: 87556: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x410, size: 0x4, is_write: 1)
3621312885: 87556: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing TX size register (value: 0x2, id: 0)
3621527154: 87567: [34m/sys/board/chip/soc/udma/trace                [0m] UDMA access (offset: 0x414, size: 0x4, is_write: 1)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0               [0m] Accessing TX cfg register (value: 0x10, id: 0)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0               [0m] Setting cfg register (continuous: 0, size: 8bits, enable: 1, clear: 0)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0               [0m] Fetching new request from 0 (cfg_setup: 0, nb_tran: 1)
3621527154: 87567: [34m/sys/board/chip/soc/udma/hyper0_tx/trace      [0m] Enqueueing new transfer (req: 0x55d2ac944820, addr: 0x1c0011c4, size: 0x2, transfer_size: 8bits, continuous: 0)
```

BUILD/PULP/RISCV_GCC/udma.txt

TIY: > cd tests/perf/double_buffering

> make clean all run DB=1 runner_args="--trace=soc/udma:udma.txt"

VCD Traces – More human readable

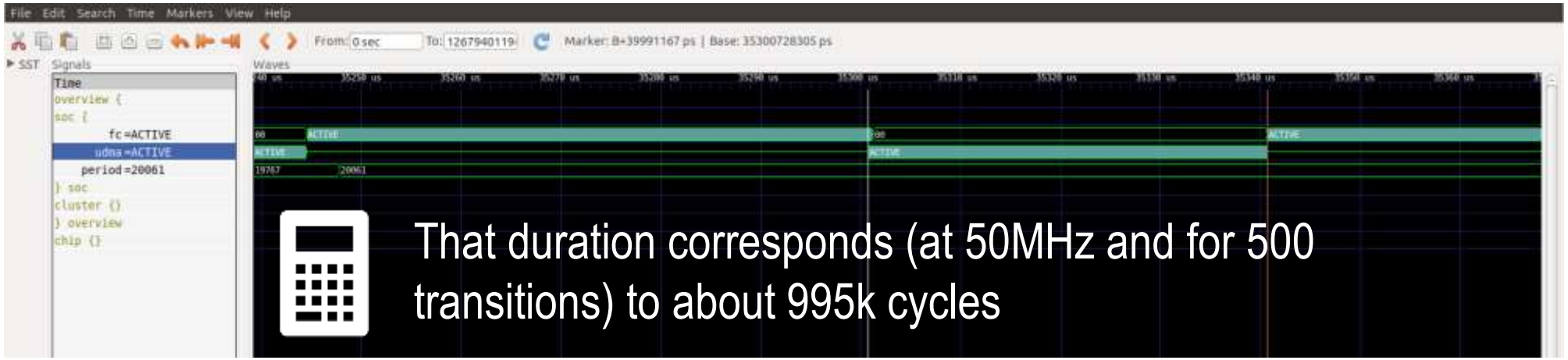
- Many more details in the gvsoc documentation https://gvsoc.readthedocs.io/en/latest/vcd_traces.html# about the concepts (not updated for all the commands)
  - $ cd tests/perf/double_buffering
  - $ make clean all run runner_args="--vcd"
- This command will create a VCD file at BUILD/PULP/RISCV_GCC/all.vcd and a file at BUILD/PULP/RISCV_GCC/view.gtkw
- Terminal will print out the command to open the latter with Gtkwaves

VCD Traces – More human readable

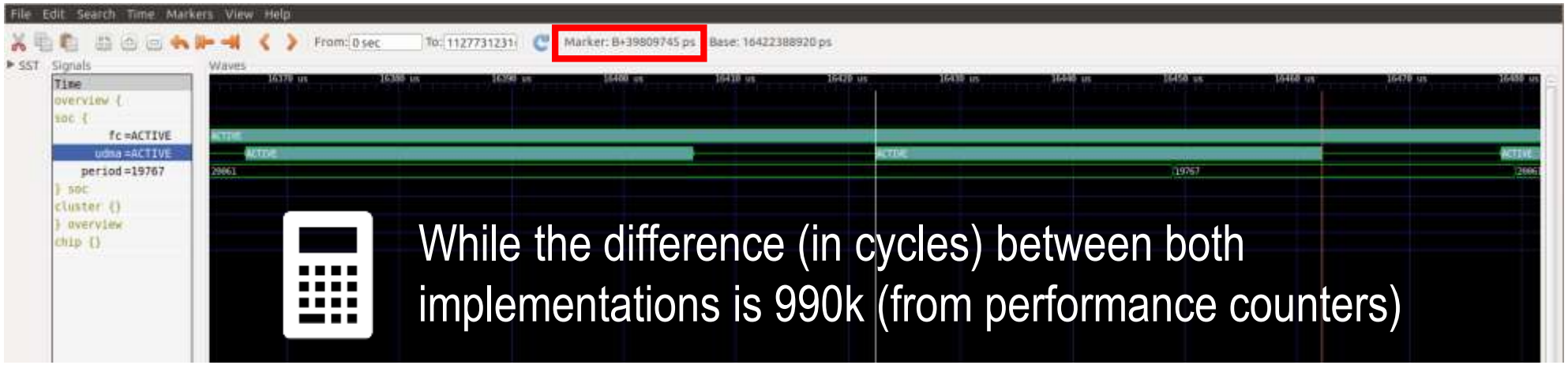
- Many more details in the gvsoc documentation https://gvsoc.readthedocs.io/en/latest/vcd_traces.html# about the concepts (not updated for all the commands)
  - $ cd tests/perf/double_buffering
  - $ make clean all run runner_args="--vcd"

TIY: > cd tests/perf/double_buffering
> make clean all run DB=1 runner_args="--vcd"
> gtkwave <INSTALLATION_PATH>/pulp-sdk/tests/hello/BUILD/PULP/GCC_RISCV/view.gtkw

VCD Traces - µDMA vs. CORE



That duration corresponds (at 50MHz and for 500 transitions) to about 995k cycles



While the difference (in cycles) between both implementations is 990k (from performance counters)