CPS Summer School 2022

Tutorial C4D:
A programmable and reconfigurable FPGA overlay

Alessandro Capotondi[1], Daniel Madroñal[2]
Gianluca Bellocchi[1], Andrea Marongiu[1], Francesca Palumbo[2]
[1]Università degli Studi di Modena e Reggio Emilia
[2]Università degli Studi di Sassari

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826610. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Austria,

COMP4DRONES will provide a **framework** of key enabling technologies for **safe and autonomous drones** that will leverage their **customization and modularity** for civilian services

COMP4DRONES

ECSEL JU GA No 826610

Website: ***comp4drones.eu***

AGENDA

1 Introduction

2 Methodology overview

3 MDC tool

4 OODK overlay

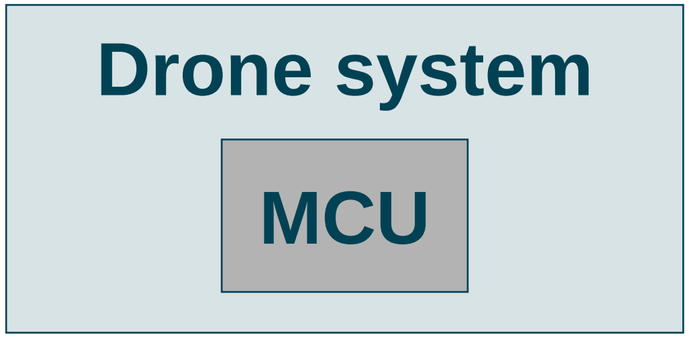
5 COMP4DRONES use case

6 Conclusions

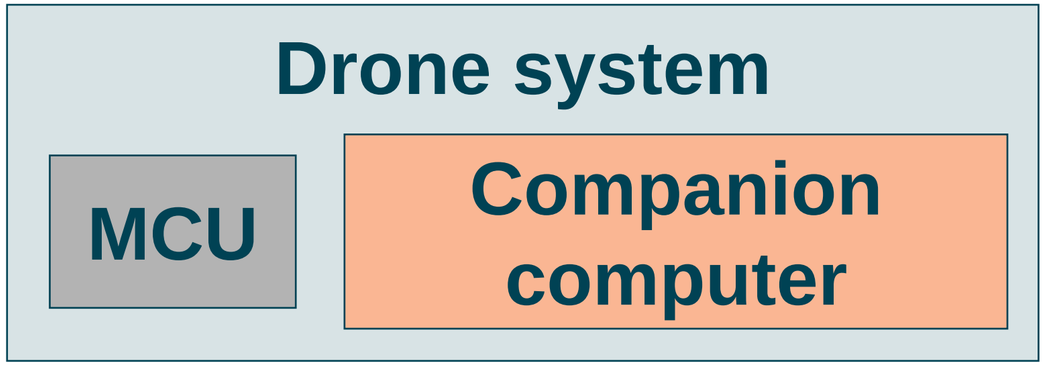
AGENDA

Introduction

Accelerator-rich paradigm



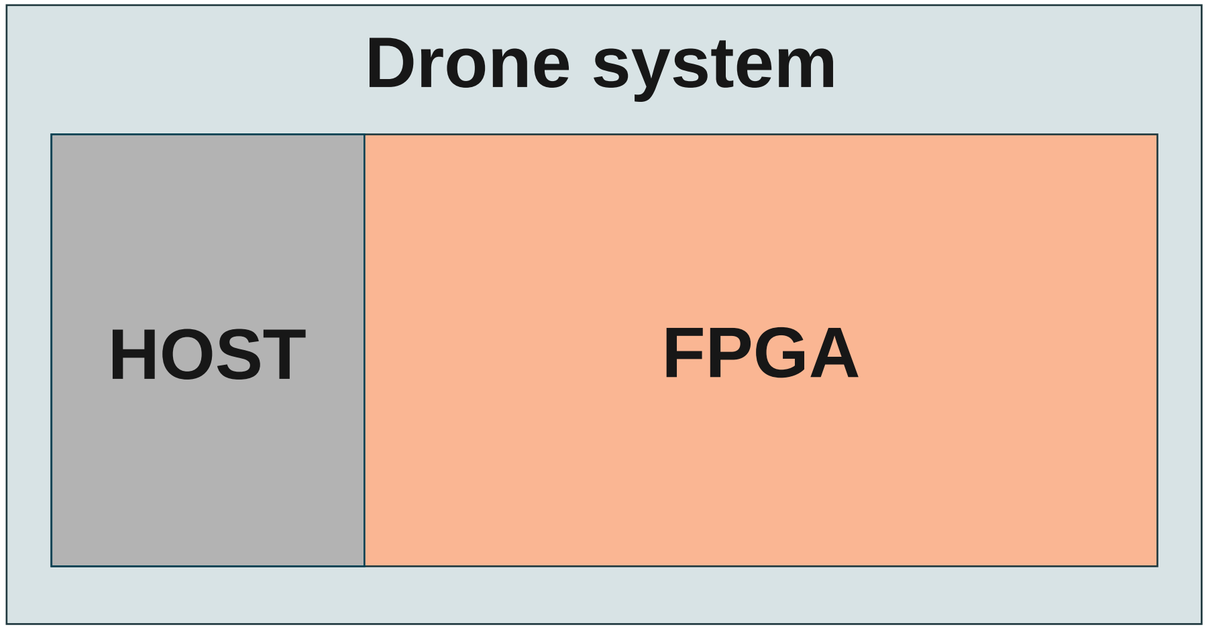
- The *“classic” set-up* comprises a **micro-controller unit (MCU)** that is used for control and actuation



- *Current paradigm* envisions coupling a **MCU** with a **companion computer**
- **Heterogeneous solutions** (Nvidia Tegra TX2, Xilinx Zynq US+, ..) are increasingly used

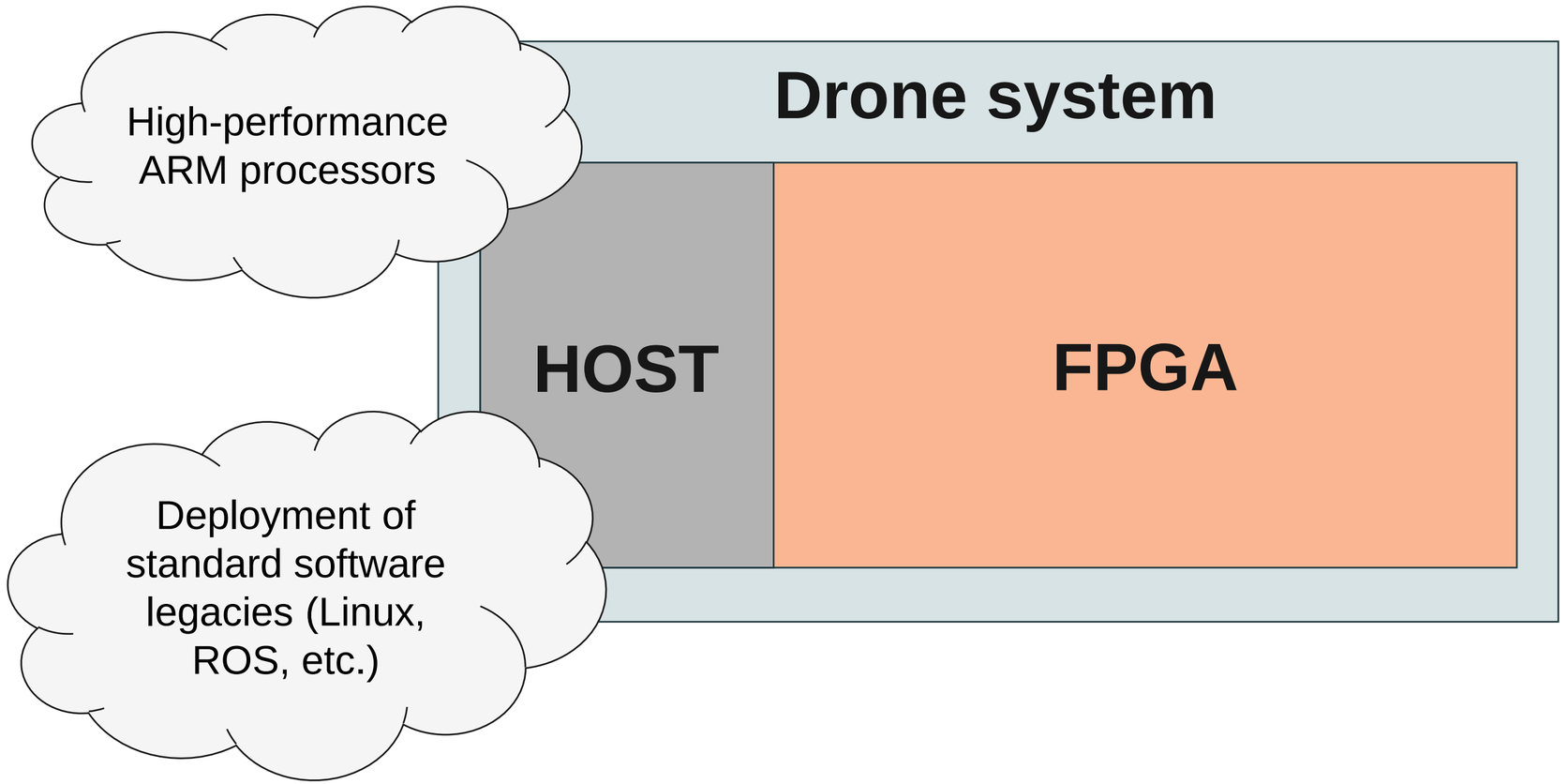
Introduction

Accelerator-rich paradigm



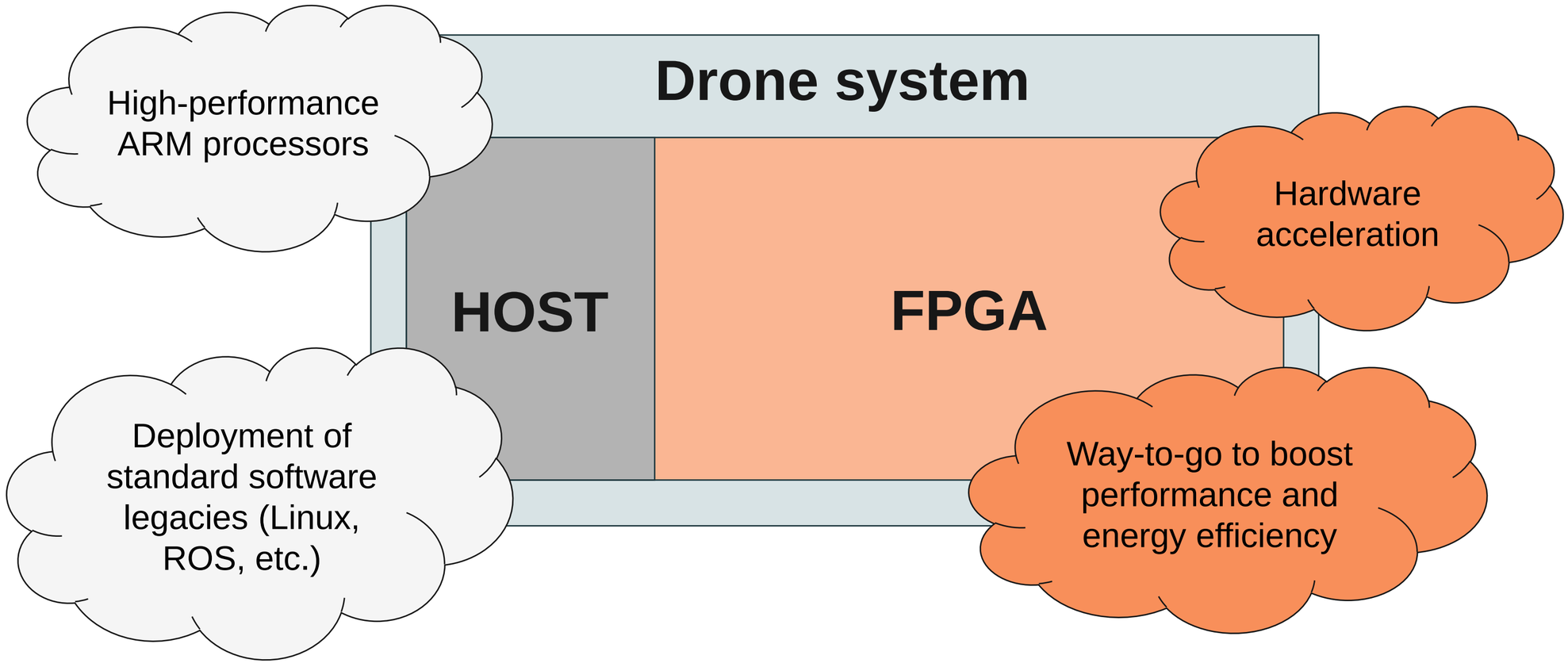
Introduction

Accelerator-rich paradigm



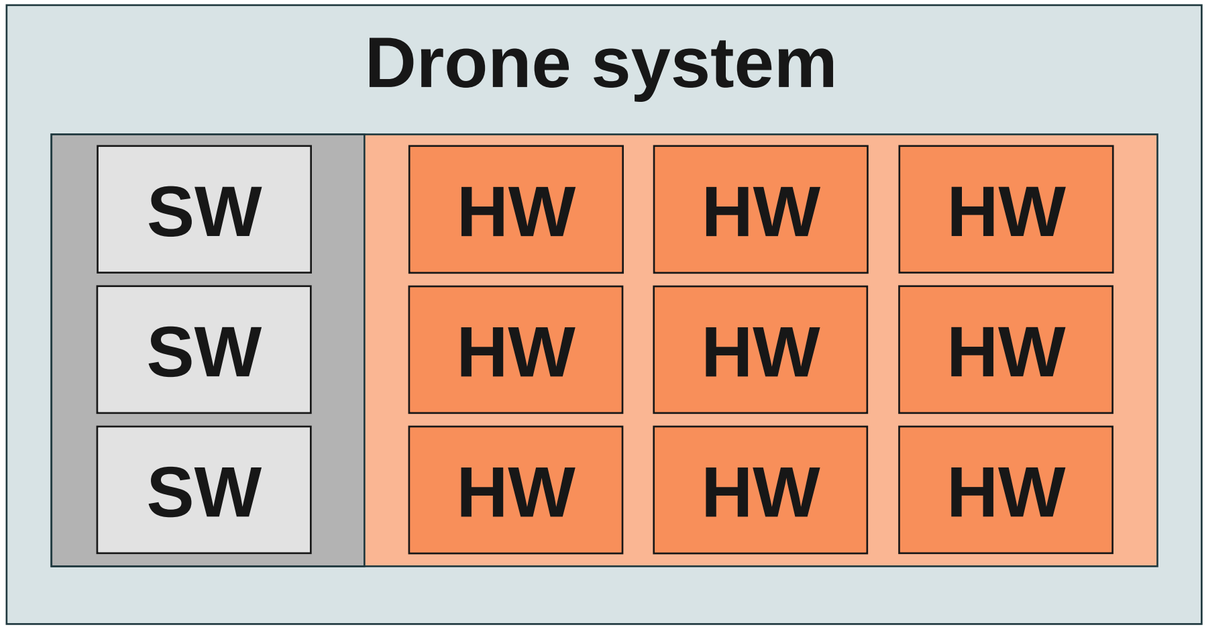
Introduction

Accelerator-rich paradigm



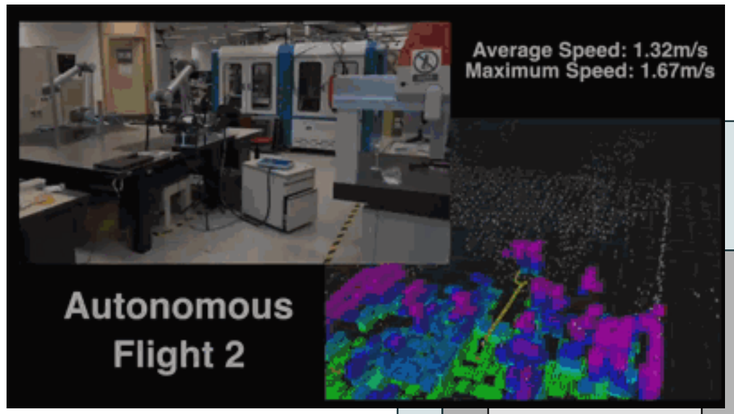
Introduction

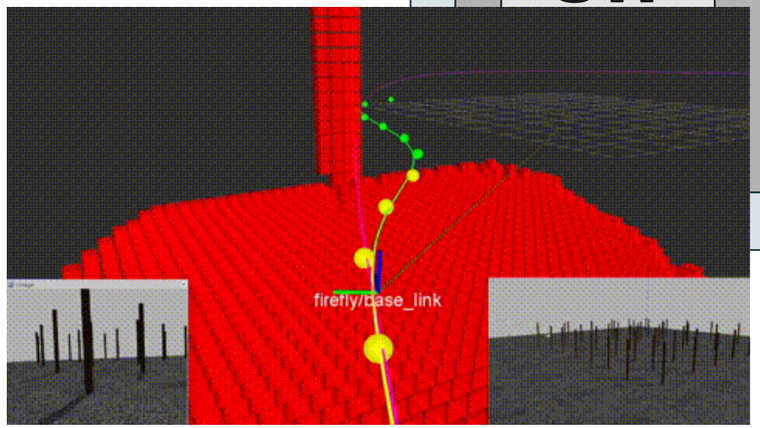
Accelerator-rich paradigm



Introduction

Accelerator-rich paradigm





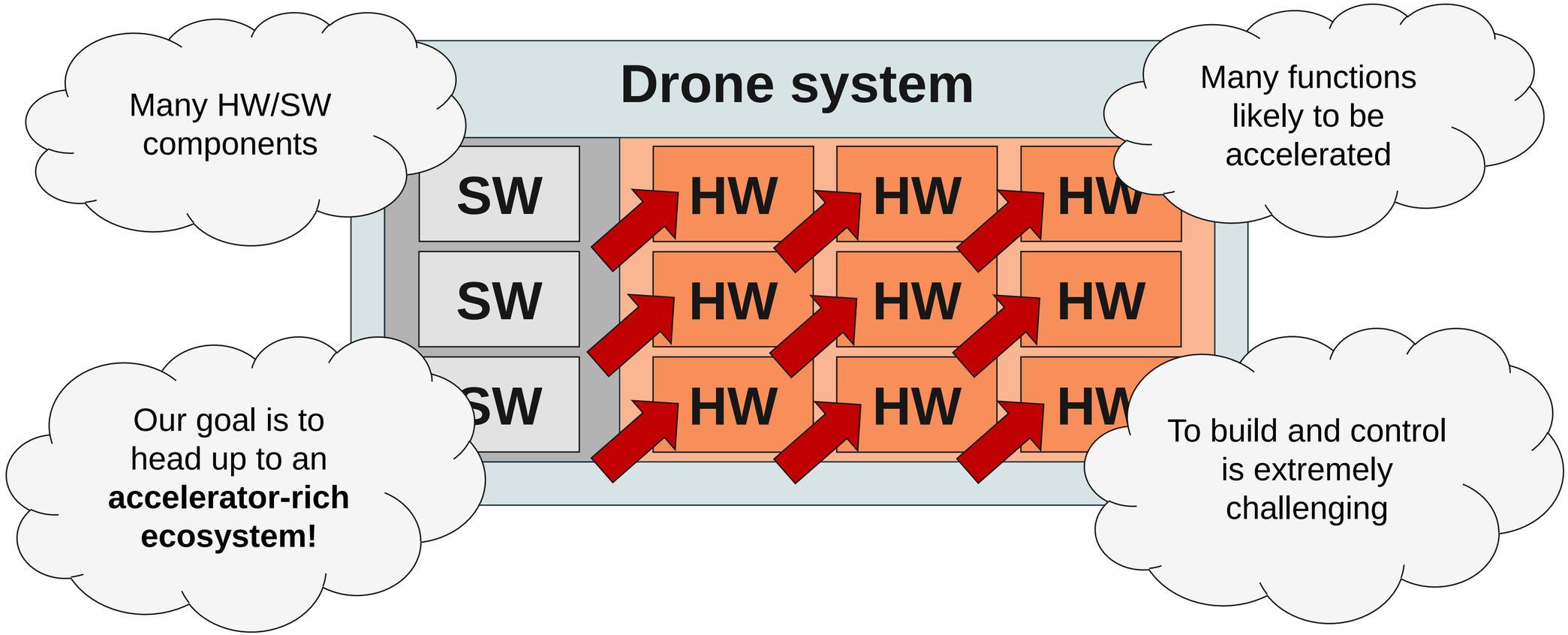
Drone system

SW

HW HW HW
HW HW
HW HW

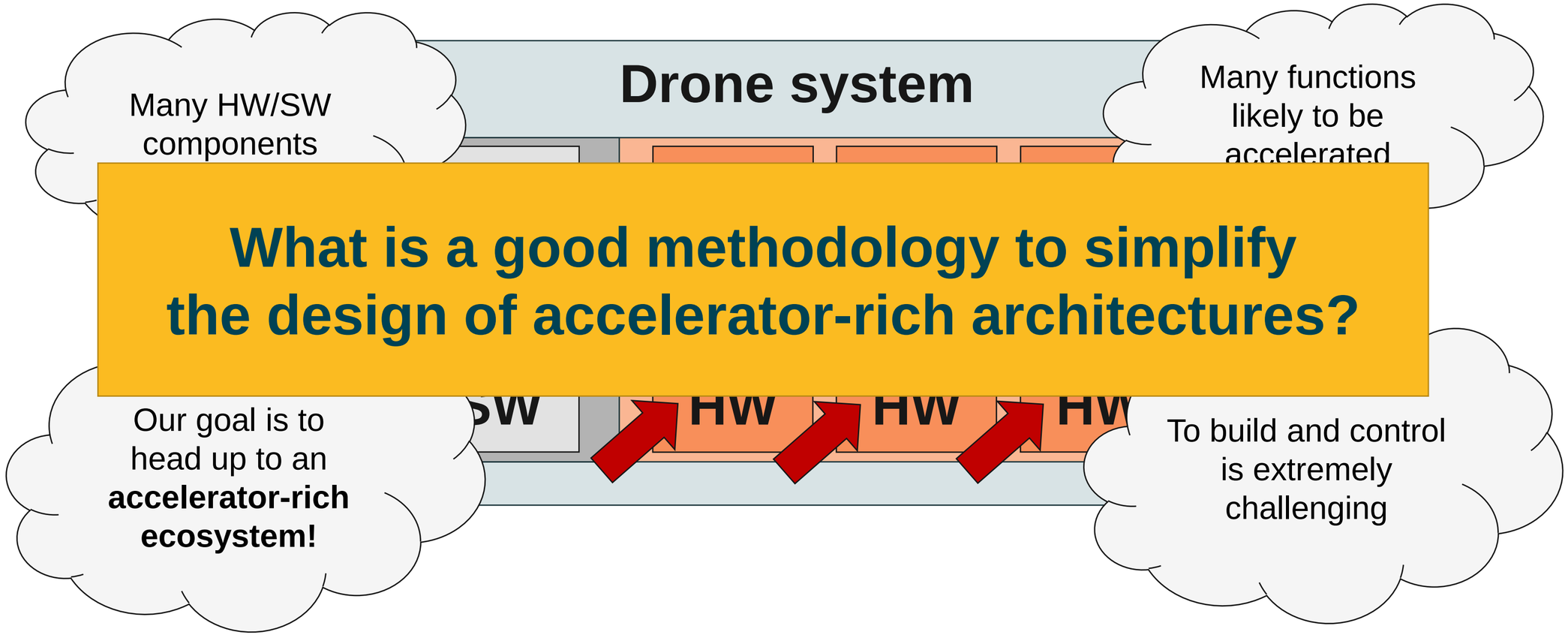
Introduction

Accelerator-rich paradigm



Introduction

Accelerator-rich paradigm



What is a good methodology to simplify the design of accelerator-rich architectures?

Introduction

Motivation

What has to be simplified?

- *System-Level Design*
  - Build and evaluate accelerator-rich systems
    - Expensive
    - Time-consuming

Introduction

Motivation

What has to be simplified?

- *System-Level Design*
  - Build and evaluate accelerator-rich systems
    - Expensive
    - Time-consuming

- *Design Space Exploration (DSE)*
  - Key effects only manifest at system-level
  - User knobs:
    - System optimization
    - Accelerator optimization

Introduction

Motivation

What has to be simplified?

- *System-Level Design*
  - Build and evaluate accelerator-rich systems
    - ❖ Expensive
    - ❖ Time-consuming

- *Design Space Exploration (DSE)*
  - Key effects only manifest at system-level
  - User knobs:
    - ❖ System optimization
    - ❖ Accelerator optimization

- *Accelerator Design*
  - Multi-functionality support
  - Multi working-point support

Introduction

Structure of the presentation

Introduction

Structure of the presentation

Step 1:

Overview of the proposed methodology (How to build a whole FPGA-based system starting from a dataflow specification)

Introduction

Structure of the presentation

Step 1:

Overview of the proposed methodology (How to build a whole FPGA-based system starting from a dataflow specification)

Step 2:

Accelerator definition and generation (MDC workflow)

Introduction

Structure of the presentation

Step 1:

Overview of the proposed methodology (How to build a whole FPGA-based system starting from a dataflow specification)

Step 2:

Accelerator definition and generation (MDC workflow)

Step 3:

Overlay connection and usage from SW (OODK workflow)

AGENDA

Methodology overview

High-level outline

1) Dataflow specification

2) Datapath merging and wrapper generation

3) Build the system

Methodology overview

High-level outline

1) Dataflow specification 2) Datapath merging and wrapper generation 3) Build the system

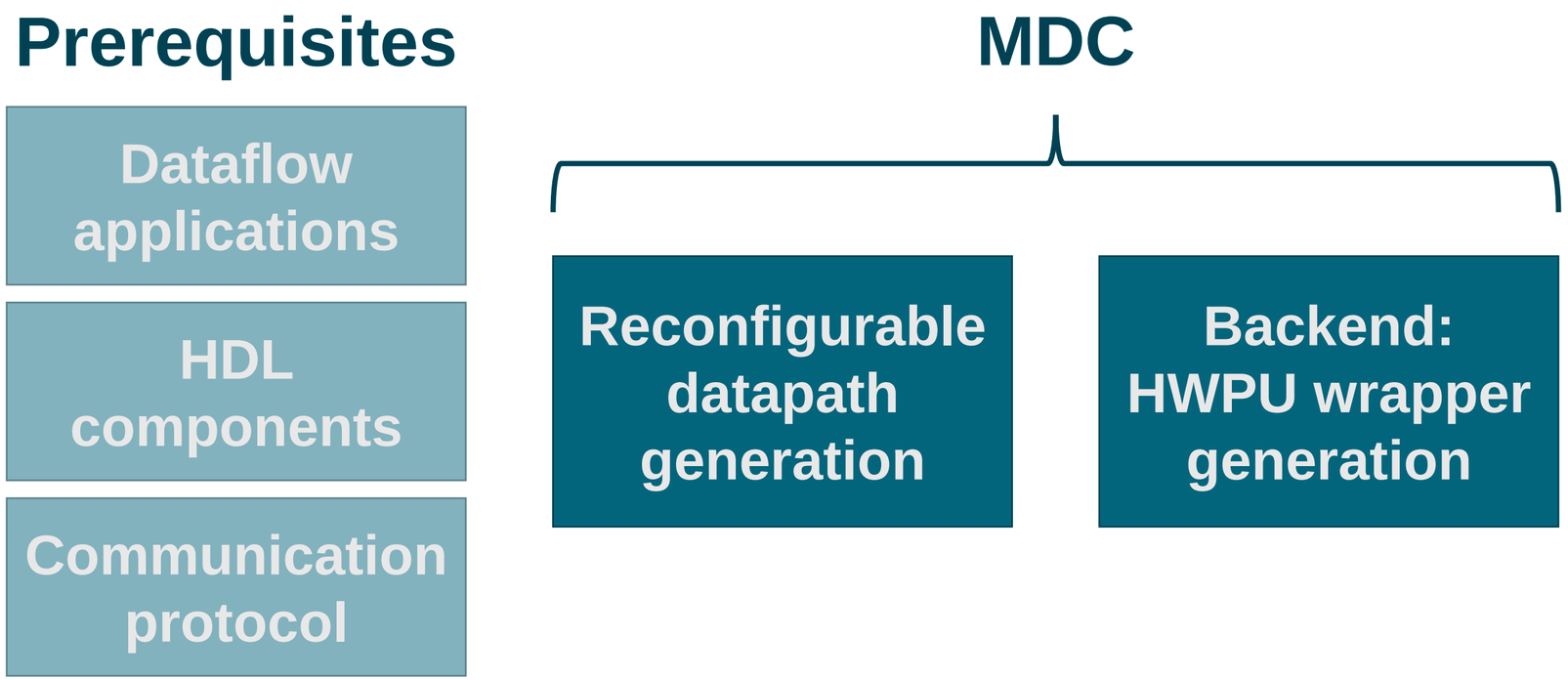
Prerequisites

- **Dataflow applications**
- **HDL components**
- **Communication protocol**

Methodology overview

High-level outline

1) Dataflow specification 2) Datapath merging and wrapper generation 3) Build the system



Methodology overview

High-level outline

1) Dataflow specification 2) Datapath merging and wrapper generation 3) Build the system

Prerequisites

- Dataflow applications
- HDL components
- Communication protocol

MDC

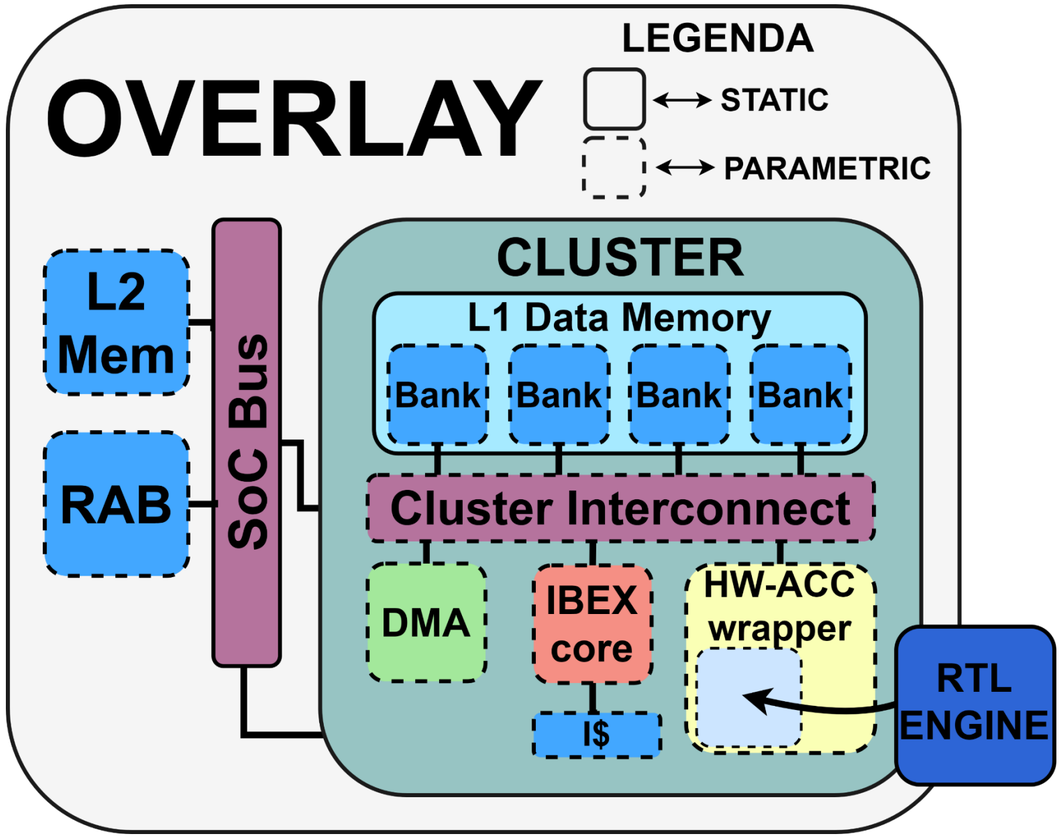
- Reconfigurable datapath generation
- Backend: HWPU wrapper generation

FPGA overlay

- Build the system: Overlay + HWPUs

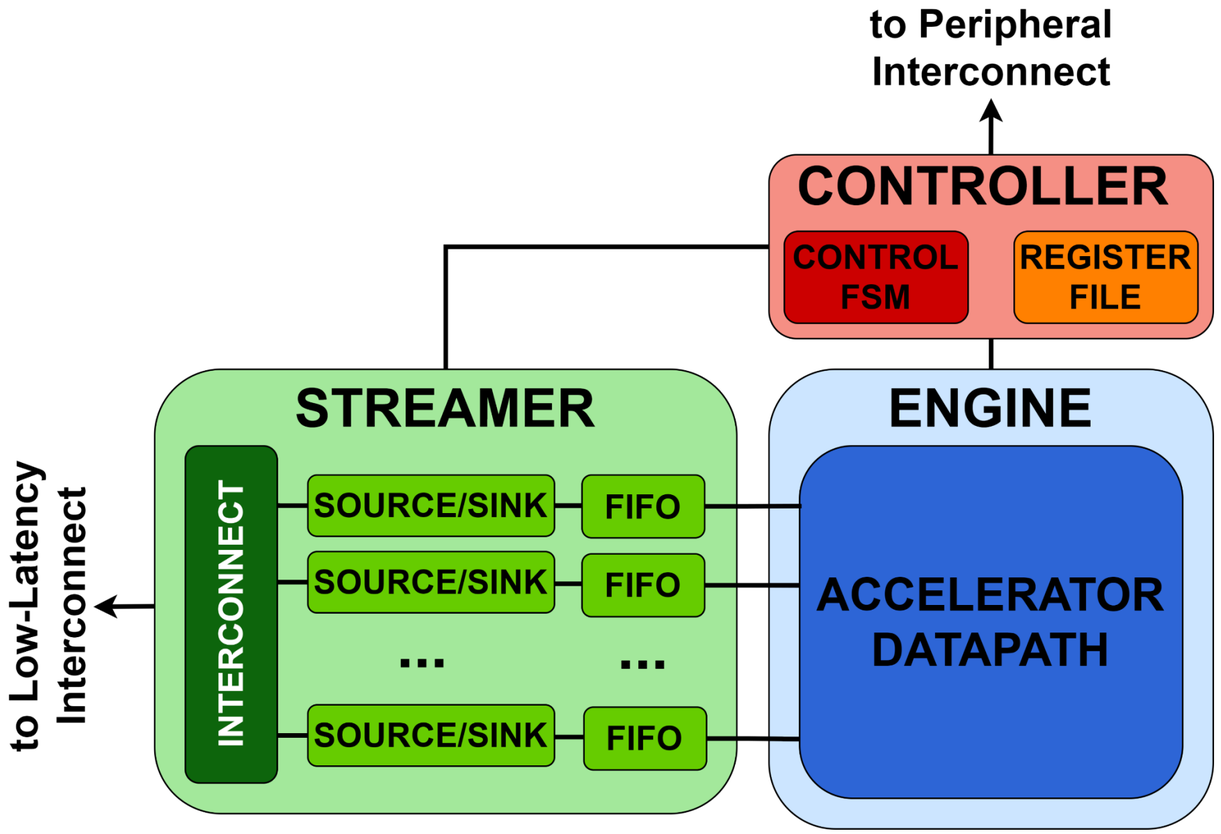
Methodology overview

FPGA overlay



Methodology overview

HWPU accelerator wrapper

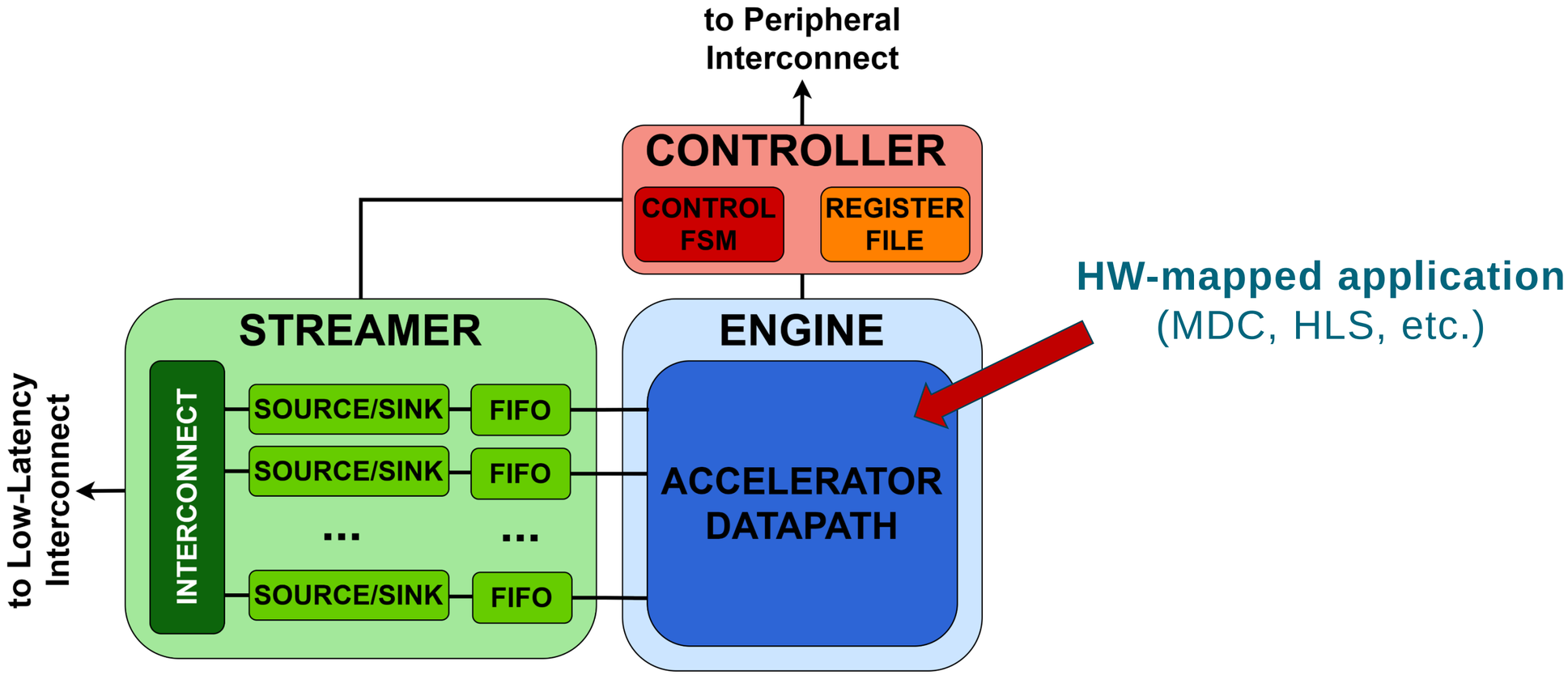


Bellocchi, Gianluca, Alessandro Capotondi, Francesco Conti, and Andrea Marongiu. "A risc-v-based fpga overlay to simplify embedded accelerator deployment." In 2021 24th Euromicro Conference on Digital System Design (DSD), pp. 9-17. IEEE, 2021.

Conti, Francesco, Pasquale Davide Schiavone, and Luca Benini. "XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018): 2940-2951.

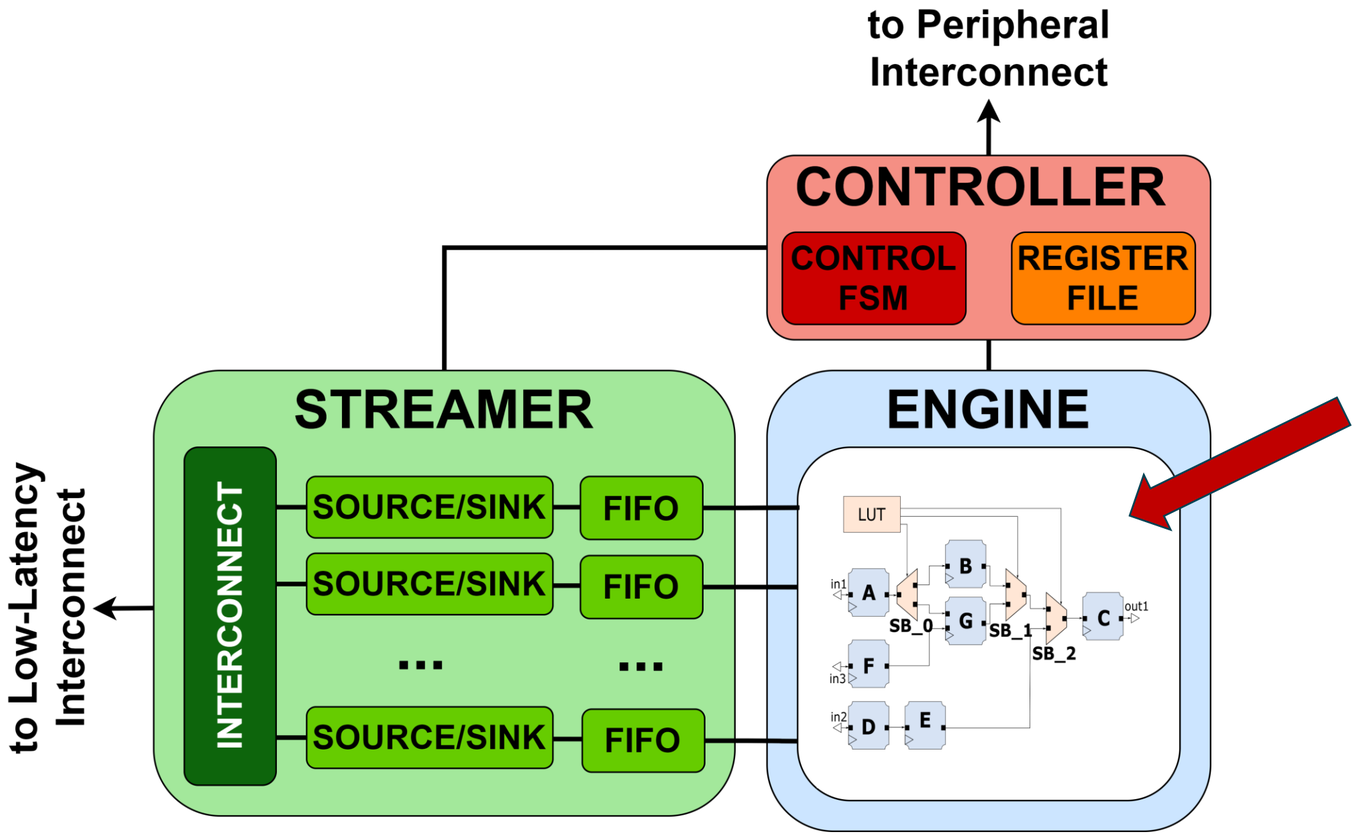
Methodology overview

HWPU accelerator wrapper



Methodology overview

HWPU accelerator wrapper



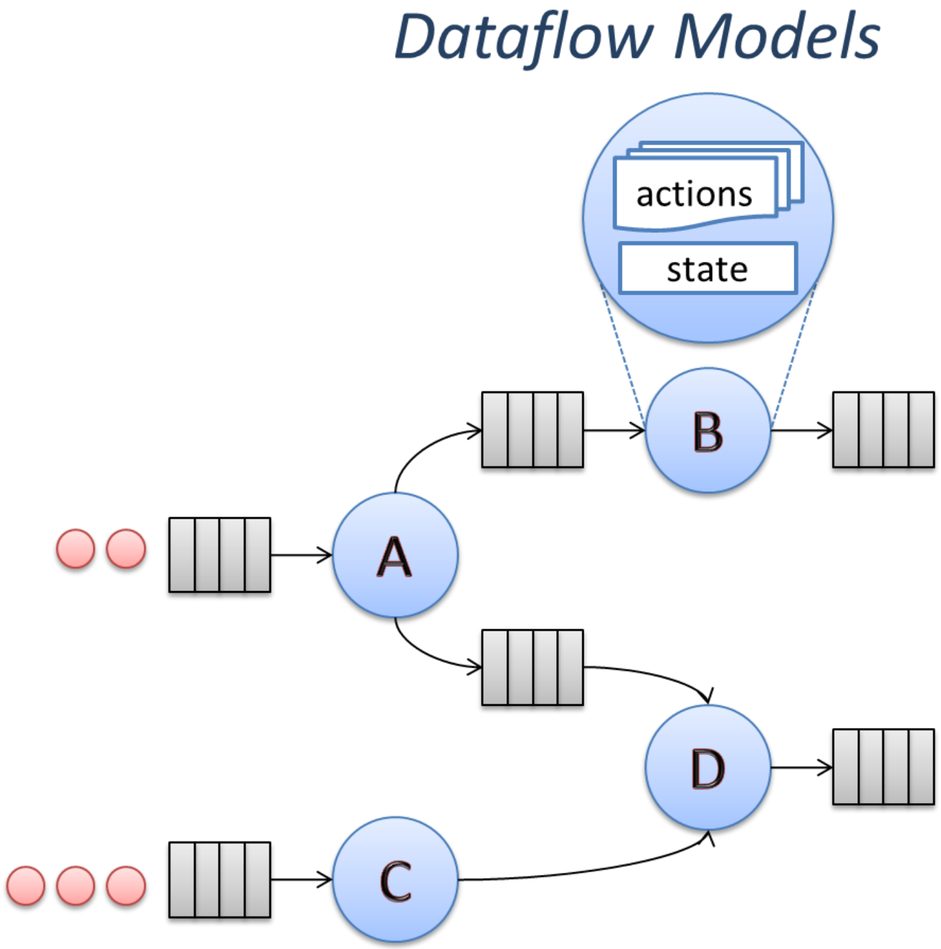
MDC-based reconfigurable application

Methodology overview

App modeling

Methodology overview

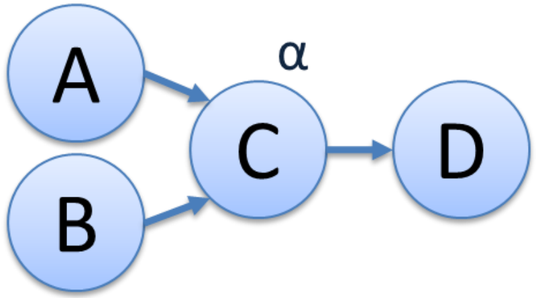
App modeling

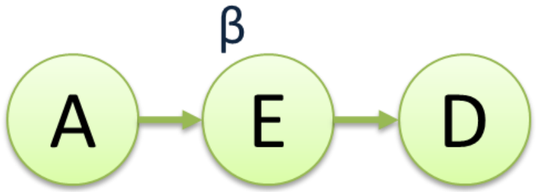


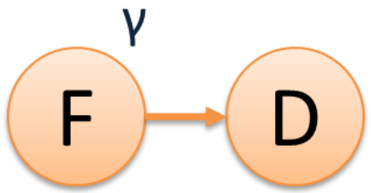
- Directed graph of actors (functional units)
- Actors exchange tokens (data packets) through dedicated channels

Methodology overview

App modeling

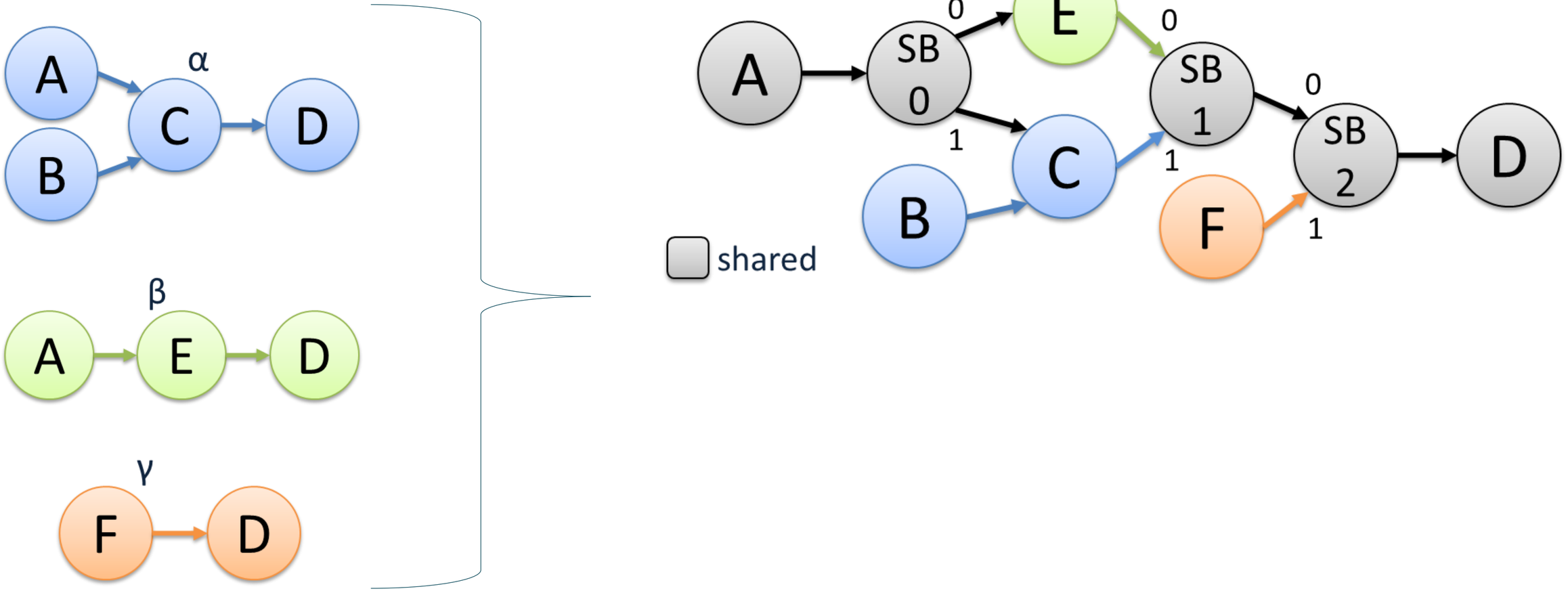






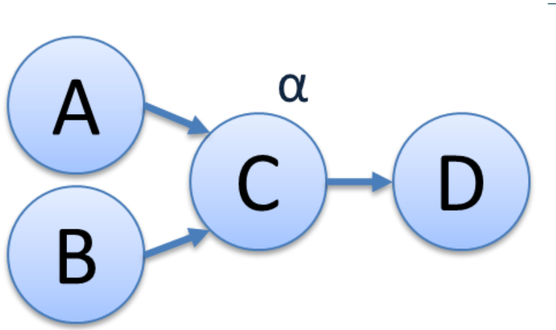
Methodology overview

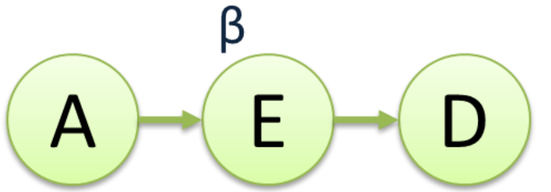
App modeling

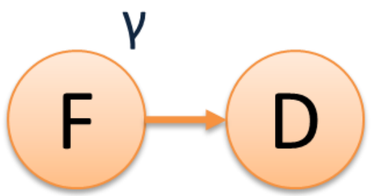


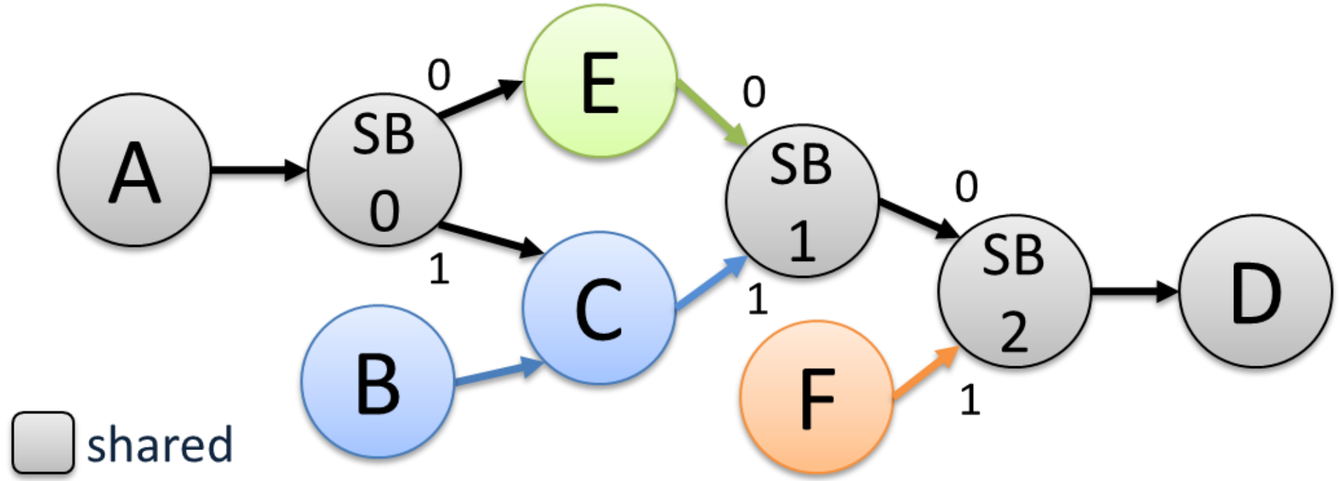
Methodology overview

App modeling





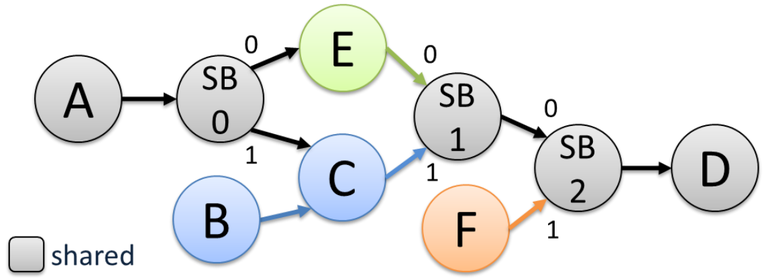




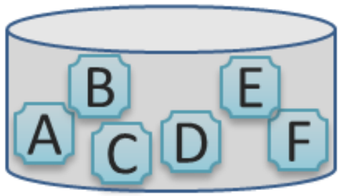
SB	0	1	2
α	1	1	0
β	0	0	0
γ	x	x	1

Methodology overview

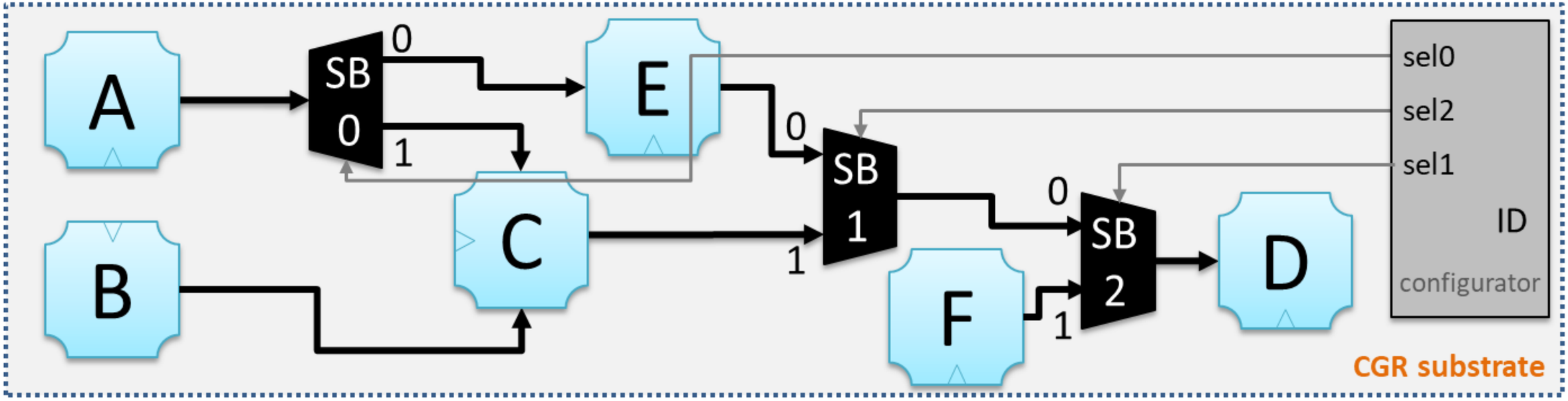
HW accelerator generation



HDL components library

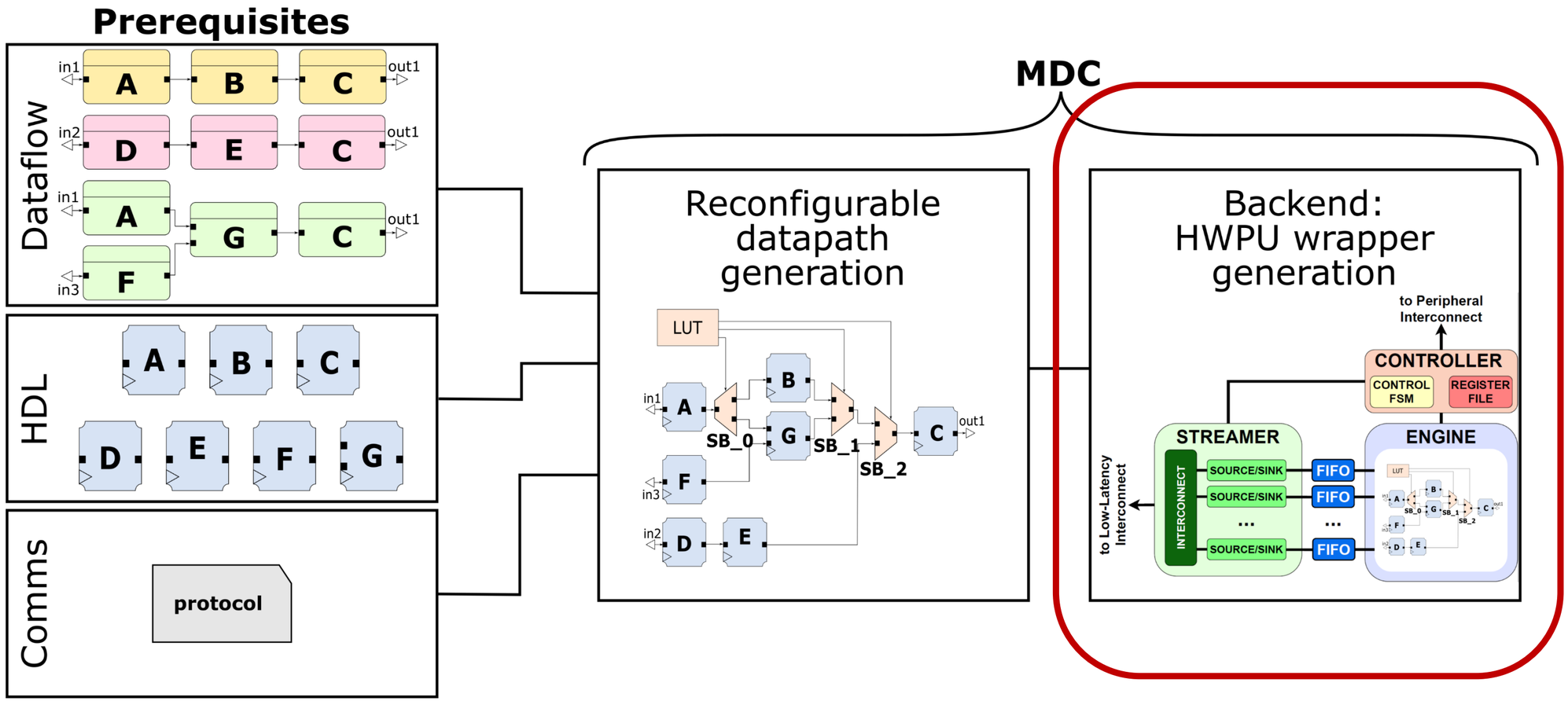


hardware communication protocol (XML)



Methodology overview

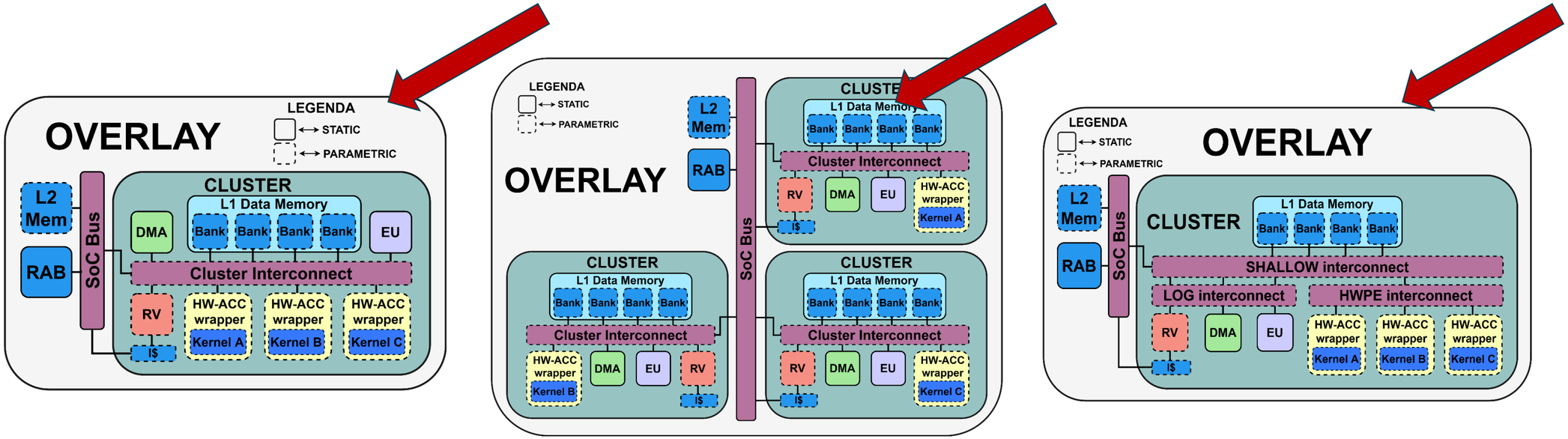
HW accelerator integration



Methodology overview

System generation

A subset of the generable accelerator-rich systems



Agile system-level design and exploration methodology

AGENDA

MDC

What application are we using in this tutorial?

MDC

What application are we using in this tutorial?

Edge detection using different kernels

INPUT IMAGE

MDC

What application are we using in this tutorial?

Edge detection using different kernels

INPUT IMAGE

SOBEL

MDC

What application are we using in this tutorial?

Edge detection using different kernels

INPUT IMAGE

SOBEL

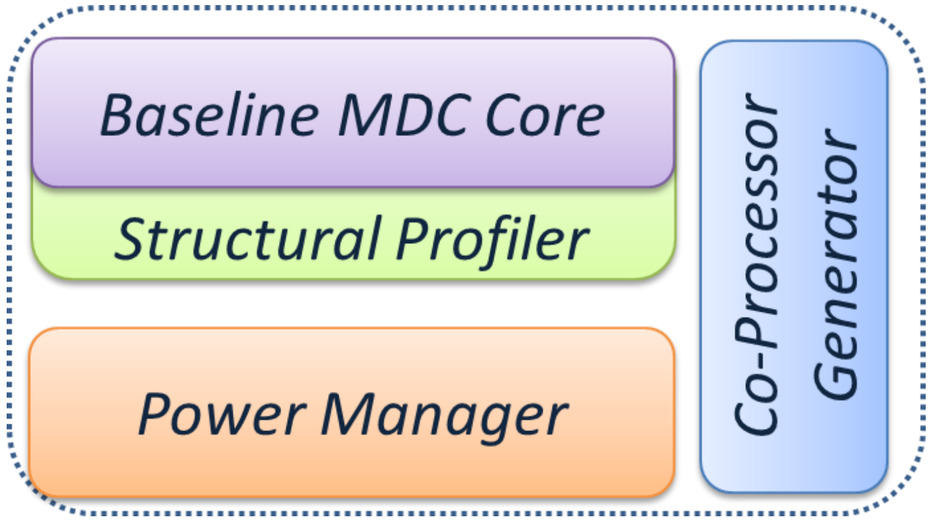
ROBERTS

MDC

Multi-Dataflow Composer concepts

MDC

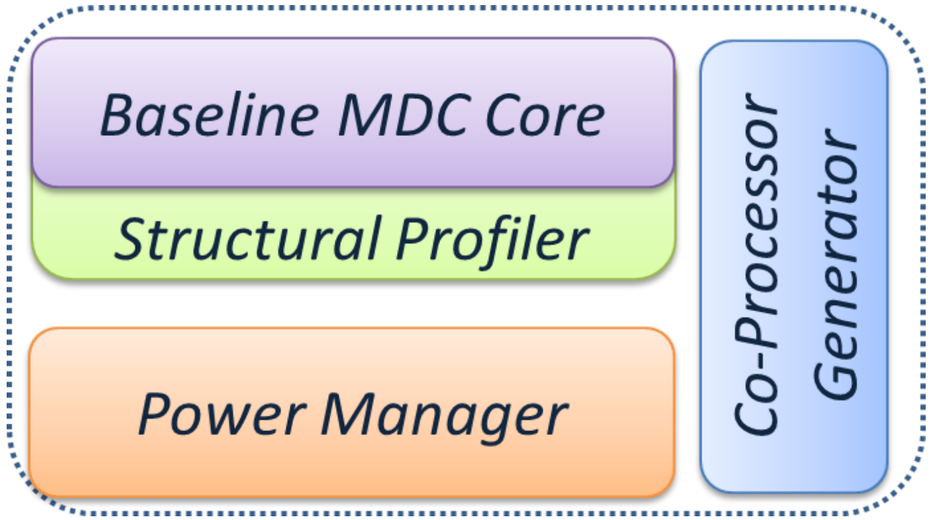
Multi-Dataflow Composer concepts



MDC *design suite:*
https://github.com/mdc-suite

MDC

Multi-Dataflow Composer concepts

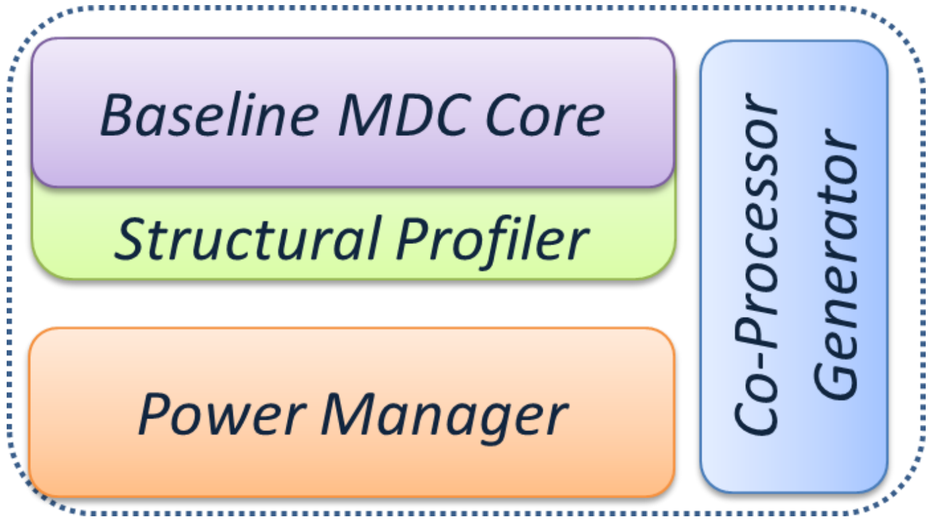


Baseline MDC Core: Datapath merging and CGR generation

MDC *design suite:*
https://github.com/mdc-suite

MDC

Multi-Dataflow Composer concepts



Baseline MDC Core: Datapath merging and CGR generation

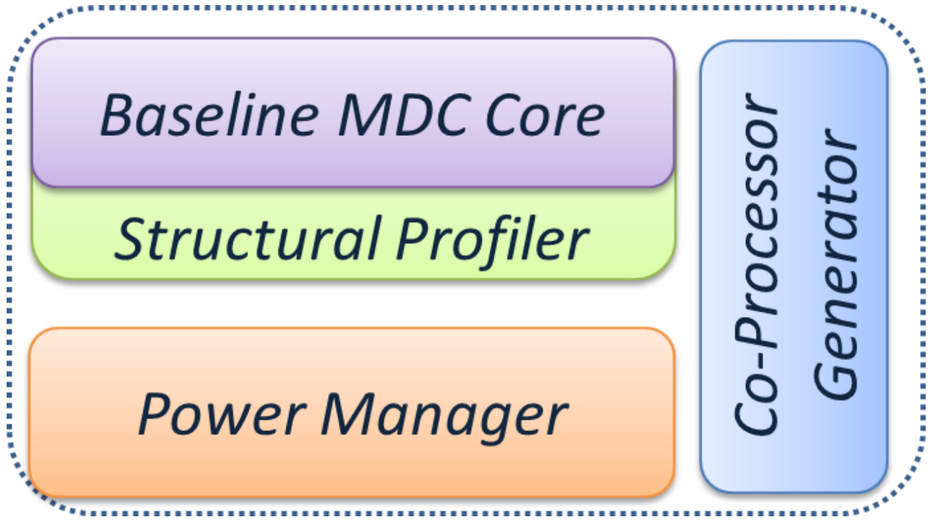
Structural Profiler: DSE for optimal CGR composition

Power Manager: Clock and power gating by regions

MDC *design suite:*
https://github.com/mdc-suite

MDC

Multi-Dataflow Composer concepts



Baseline MDC Core: Datapath merging and CGR generation

Structural Profiler: DSE for optimal CGR composition

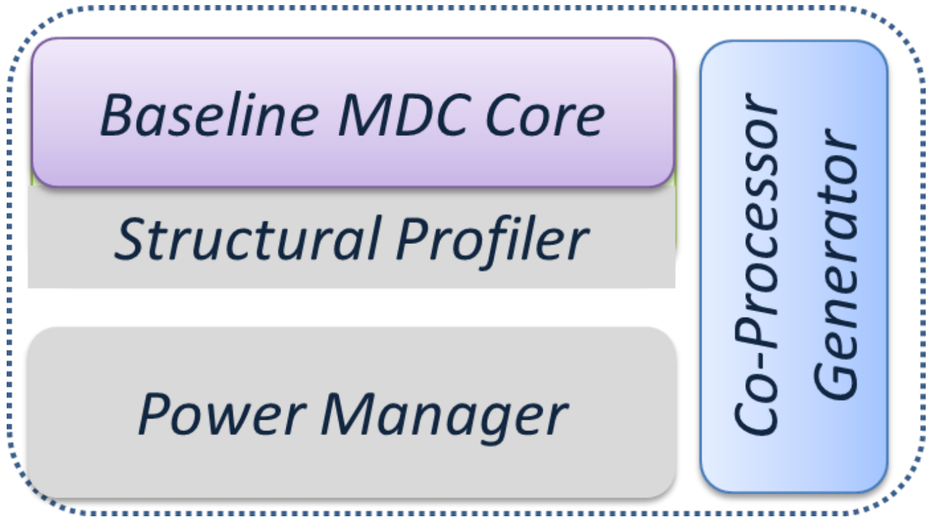
Power Manager: Clock and power gating by regions

Co-Processor Generator: Wrapper to connect accelerator and processor

MDC *design suite:*
https://github.com/mdc-suite

MDC

Multi-Dataflow Composer concepts



Baseline MDC Core: Datapath merging and CGR generation

Structural Profiler: DSE for optimal CGR composition

Power Manager: Clock and power gating by regions

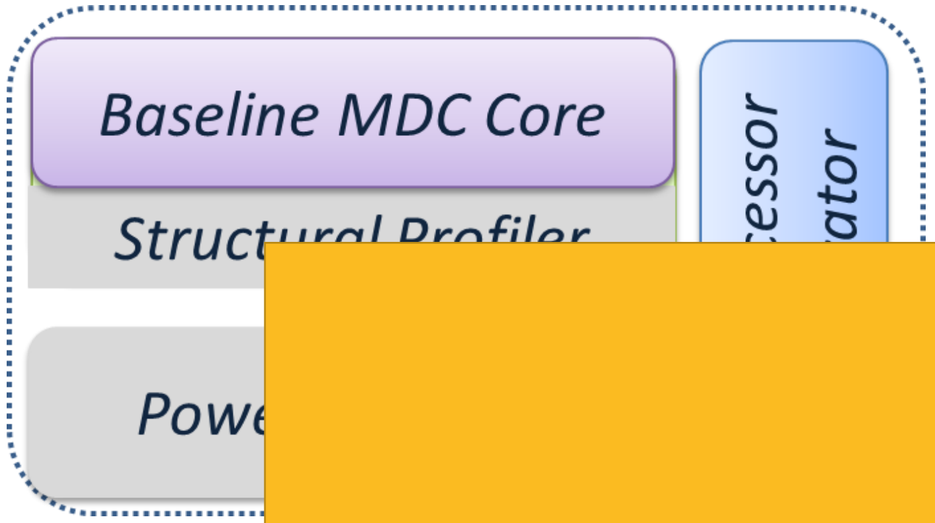
Co-Processor Generator: Wrapper to connect accelerator and processor

MDC *design suite:*
https://github.com/mdc-suite

Relevant for this tutorial

MDC

Multi-Dataflow Composer concepts



Baseline MDC Core: Datapath merging and CGR generation

Structural Profiler: DSE for optimal CGR composition

regions

ect accelerator

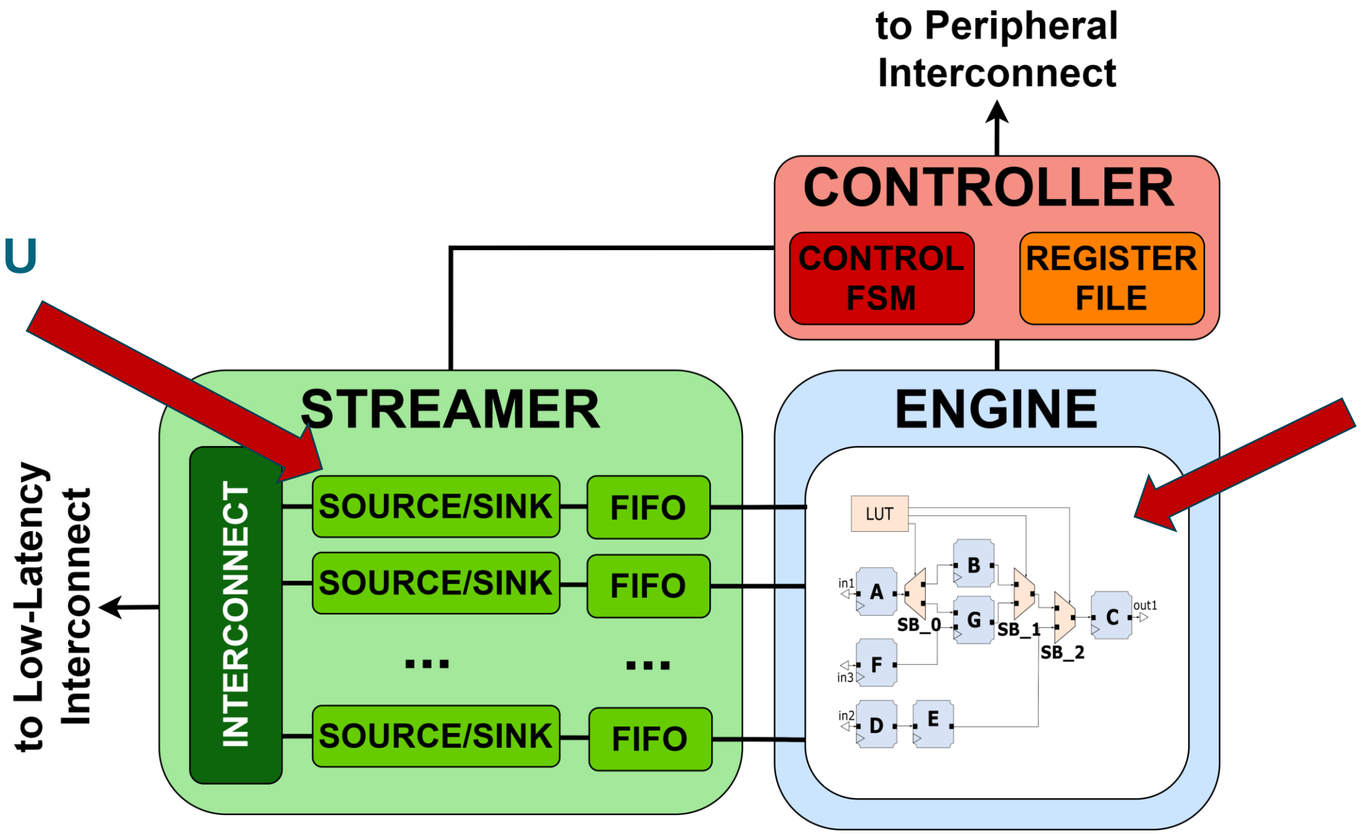
Let's try it!

MDC *design suite:*
https://github.com/mdc-suite

Relevant for this tutorial

MDC

Co-processor generation: HWPU generated by MDC



Specialized HWPU automatically generated

MDC-based reconfigurable application

MDC + OODK

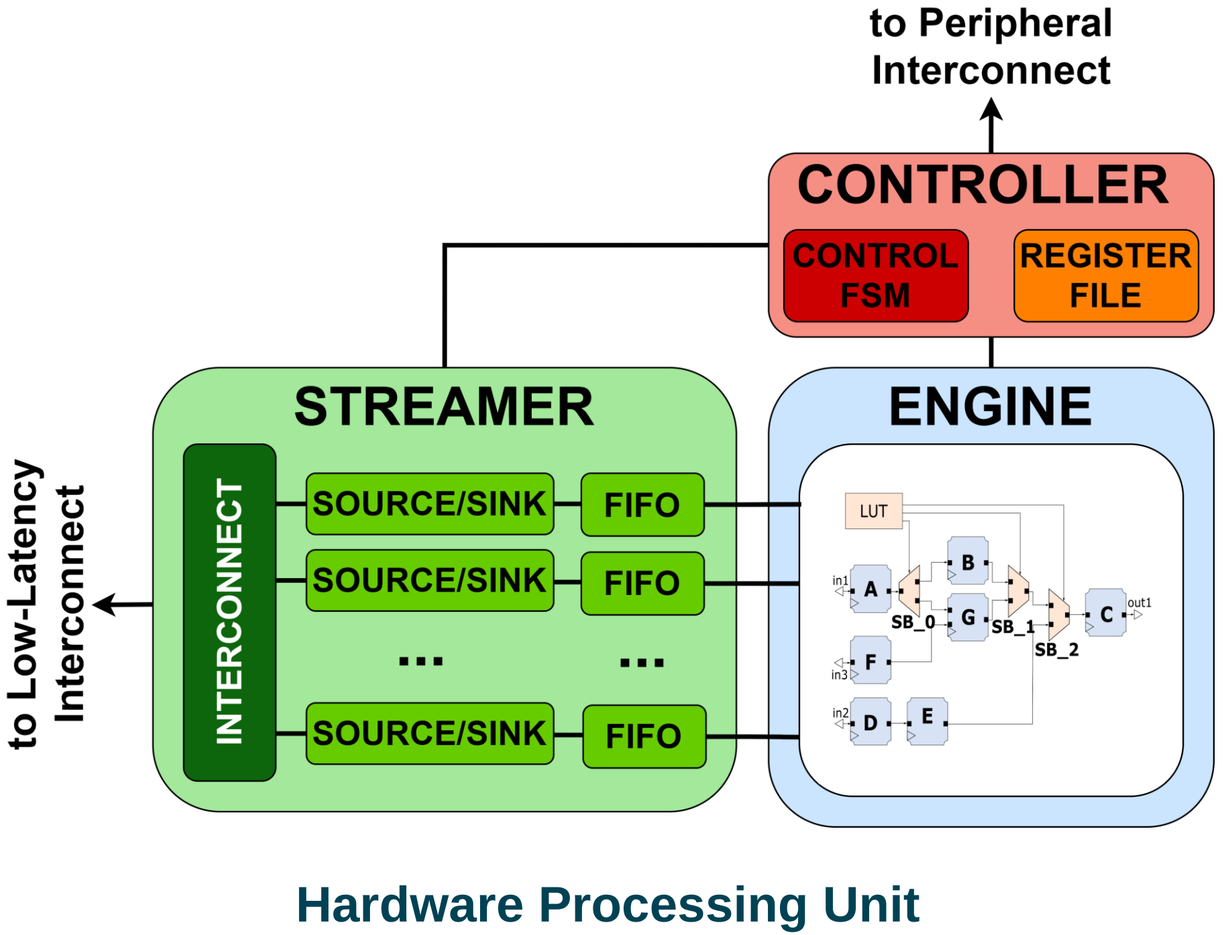
HWPU accelerator wrapper

Streamer

- Specialized DMA controller that transforms streams into memory accesses

Controller

- Register file to host runtime parameters
- Control FSM for coarse-grained control/(re-)configuration



Hardware Processing Unit

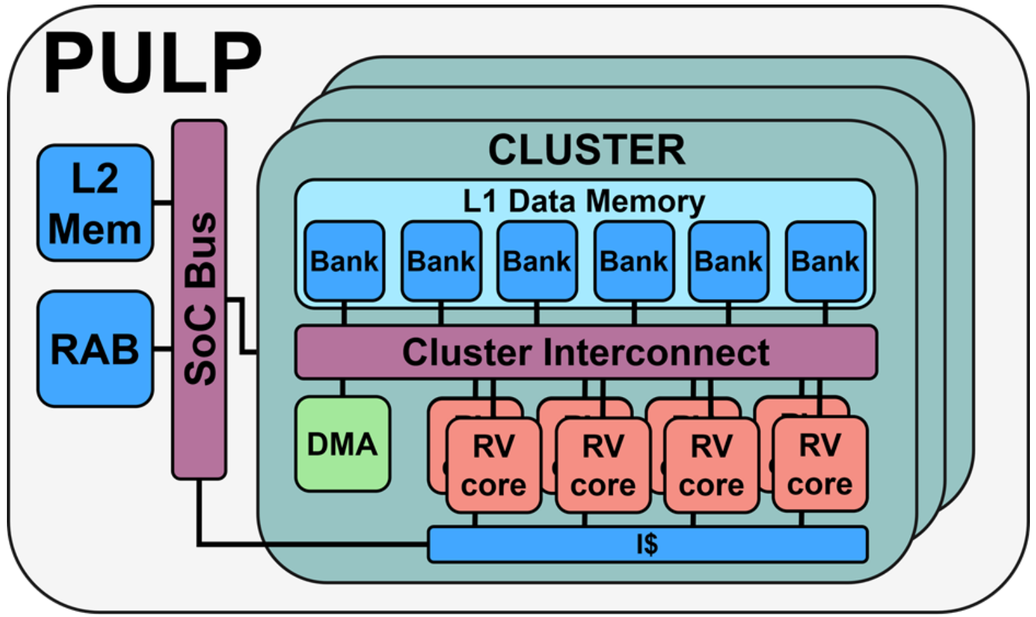
AGENDA

Overlay Development Kit

Starting point

PULP architecture

- PULP stands for «Parallel Ultra Low Power»
- Open and Scalable HW/SW research and development platform
- Cluster-based architecture
- RISC-V ISA compliant



Website: ***pulp-platform.org***

What does PULP Ecosystem include?

COMP4DRONES

RISC-V Cores

RI5CY 32b	Micro riscy 32b	Zero riscy 32b	Ariane 64b

Peripherals

JTAG	SPI
UART	I2S
DMA	GPIO

Interconnect

- Logarithmic interconnect
- APB – Peripheral Bus
- AXI4 – Interconnect

Accelerators

HWCE (convolution)	Neurostream (ML)	HWCrypt (crypto)	PULPO (1st order opt)

What does PULP Ecosystem include?

COMP4DRONES

RISC-V Cores

- RI5CY 32b
- Micro riscy 32b
- Zero riscy 32b
- Ariane 64b

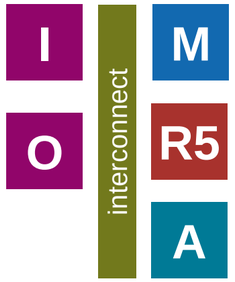
Peripherals

- JTAG
- SPI
- UART
- I2S
- DMA
- GPIO

Interconnect

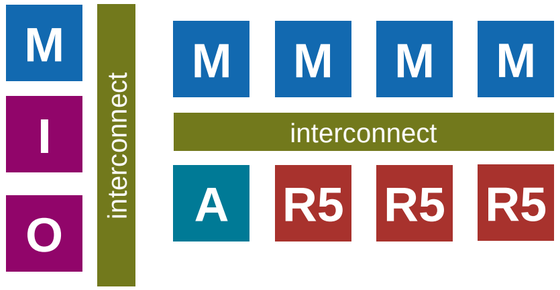
- Logarithmic interconnect
- APB – Peripheral Bus
- AXI4 – Interconnect

Platforms



Single Core

- PULPino
- PULPissimo



Multi-core

- Fulmine
- Mr. Wolf

Accelerators

- HWCE (convolution)
- Neurostream (ML)
- HWCrypt (crypto)
- PULPO (1st order opt)

What does PULP Ecosystem include?

COMP4DRONES

RISC-V Cores

- RI5CY 32b
- Micro riscy 32b
- Zero riscy 32b
- Ariane 64b

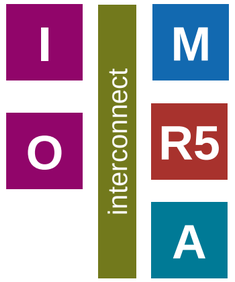
Peripherals

- JTAG
- SPI
- UART
- I2S
- DMA
- GPIO

Interconnect

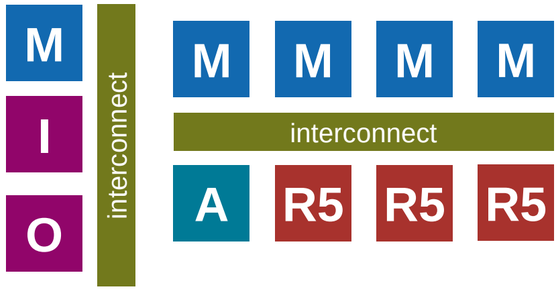
- Logarithmic interconnect
- APB – Peripheral Bus
- AXI4 – Interconnect

Platforms



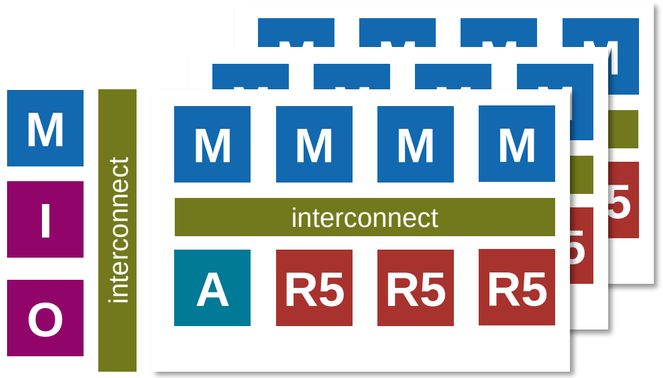
Single Core

- PULPino
- PULPissimo



Multi-core

- Fulmine
- Mr. Wolf



Multi-cluster

- Hero

IOT → **HPC**

Accelerators

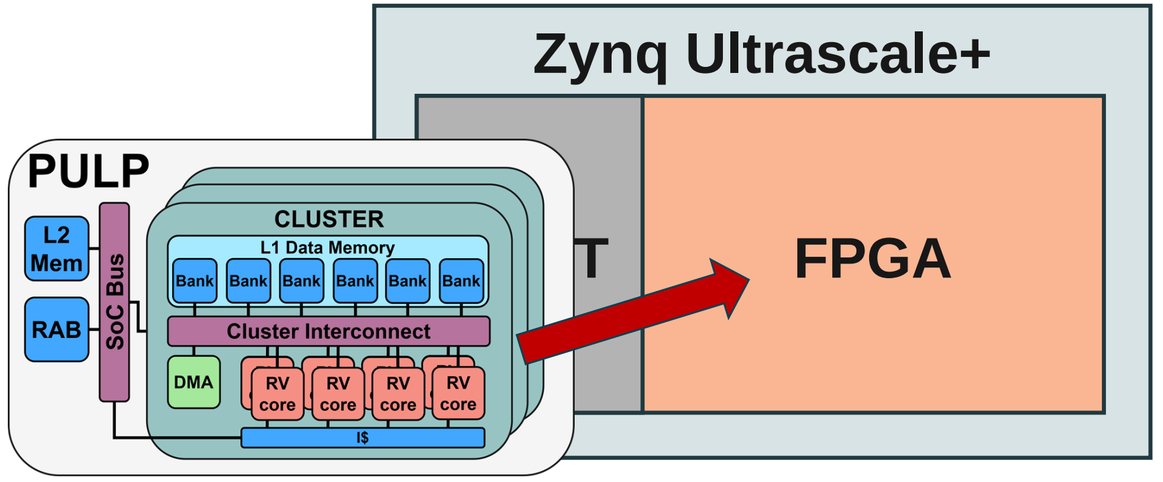
- HWCE (convolution)
- Neurostream (ML)
- HWCrypt (crypto)
- PULPO (1st order opt)

Overlay Development Kit

Starting point 2

HERO

- FPGA emulation of heterogeneous and massively parallel PULP systems
- Instantiable with COTS FPGA-based heterogeneous SoCs

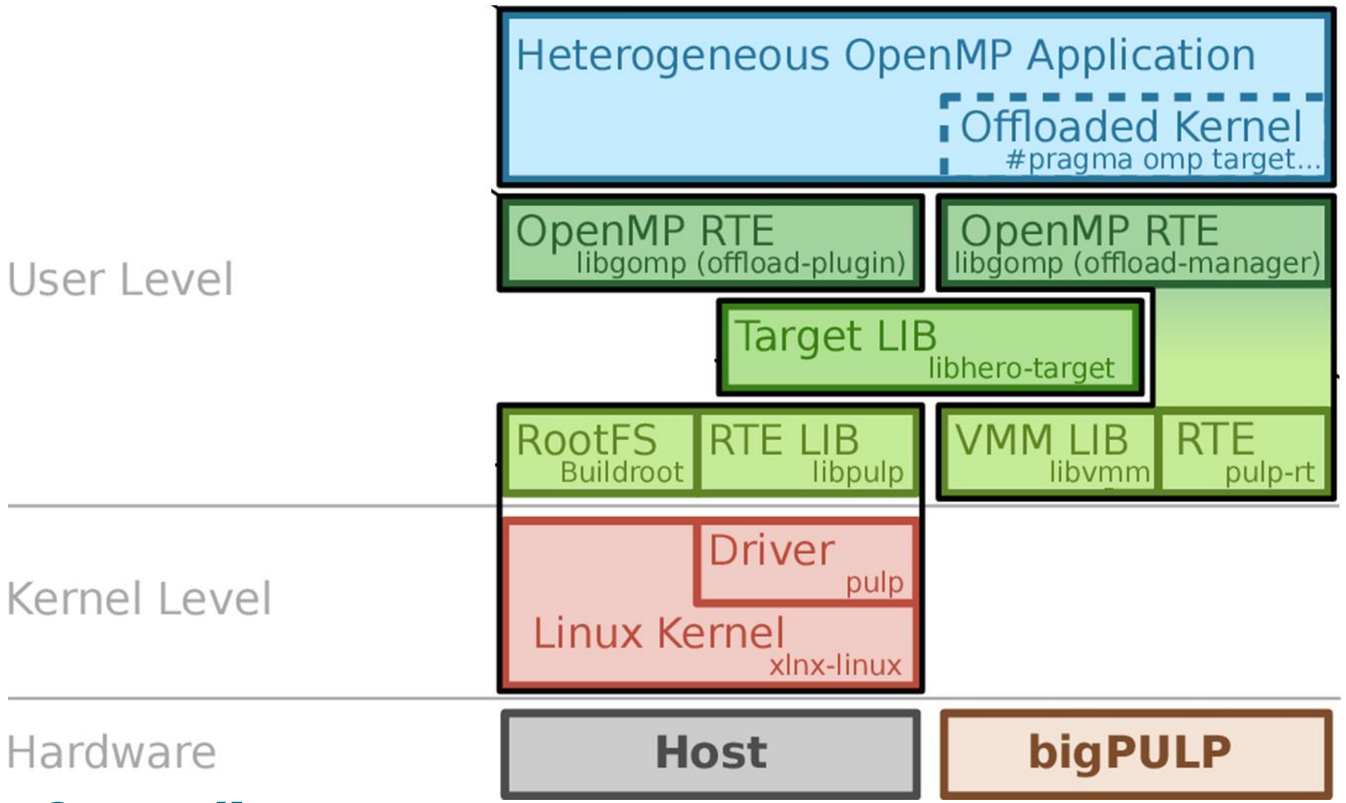


Website: ***pulp-platform.org***

Kurth, A., Capotondi, A., Vogel, P., Benini, L., & Marongiu, A. (2018)
HERO: An open-source research platform for HW/SW exploration of heterogeneous manycore systems.

HERO is not only HW

- **Includes complete SW support**
  - Linux-based OS Distribution
  - Easy to port legacy code!
- **Heterogenous programming model**
  - Based on OpenMP 5.x

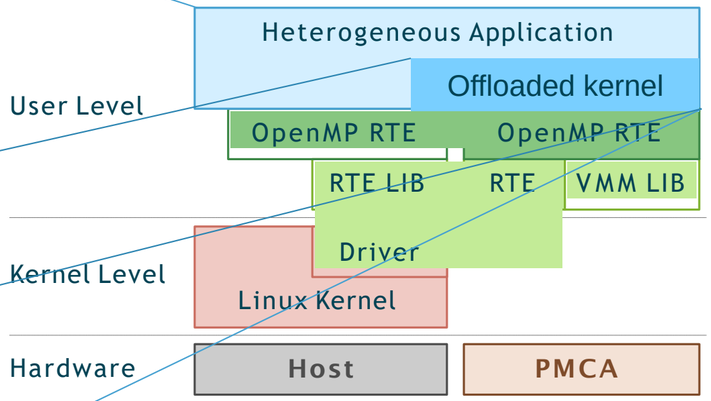


https://github.com/pulp-platform/hero

Programming Model

Allows to write programs that start on the host but seamlessly integrate the PMCAs.

```
int main()
{
  vertex vertices[N];
  load(&vertices, N);
  #pragma omp target map(tofrom:vertices)
  {
    #pragma omp parallel for
    for (i = 0; i < N; ++i)
      process(vertices[i]);
  }
}
```



- Offloads with OpenMP 4.5 `target` semantics, zero-copy (pointer passing) or copy-based

OODK

FPGA overlay

We need tools! (BIG TOOLS)

What is it?

- Hardware abstraction layer
- Overlays the original FPGA fabric → Hides hardware details
- Enable easy customization and integration of new HW Accelerator

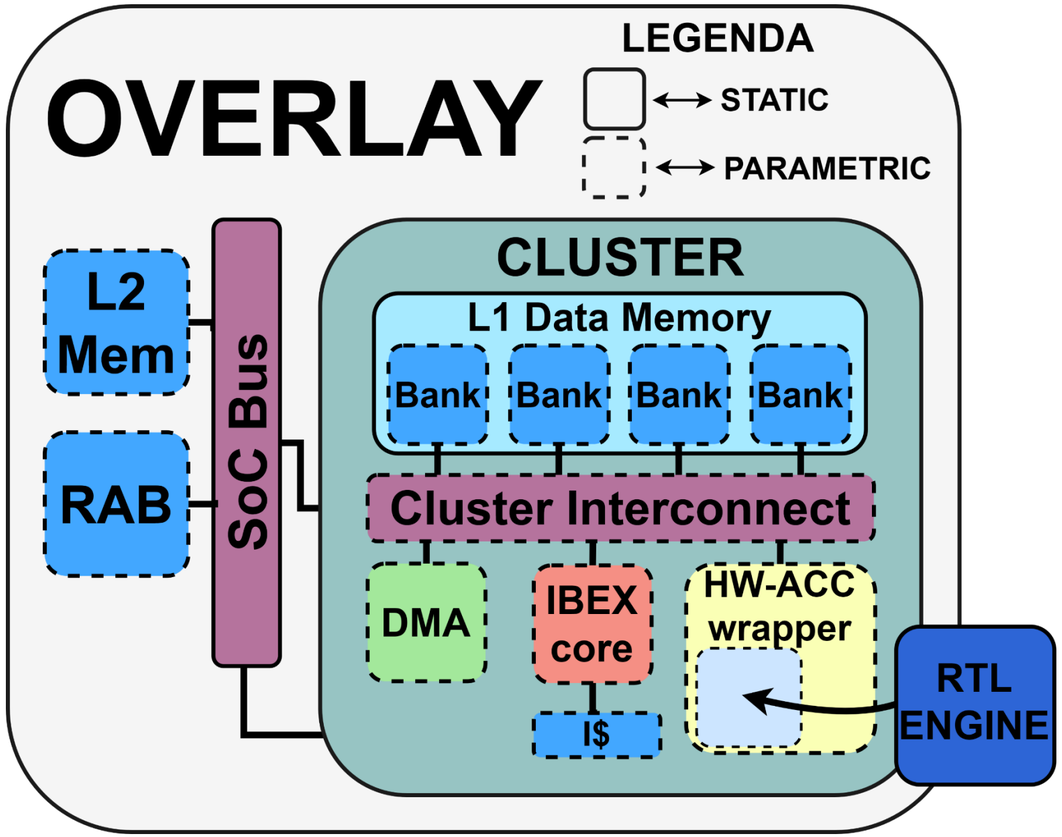
Features:

- Parametrized HW → Flexible design of custom architectures
- Abstracted design flow → Improved design productivity
- Programmable via standard APIs for heterogeneous compute platforms (e.g. OpenMP)

HDL-based IP are not easy to be customized using standard HDL language features! (can rely mainly on parametrization….)

Bellocchi, Gianluca, et al. "A risc-v-based fpga overlay to simplify embedded accelerator deployment." 2021 24th Euromicro Conference on Digital System Design (DSD). IEEE, 2021.

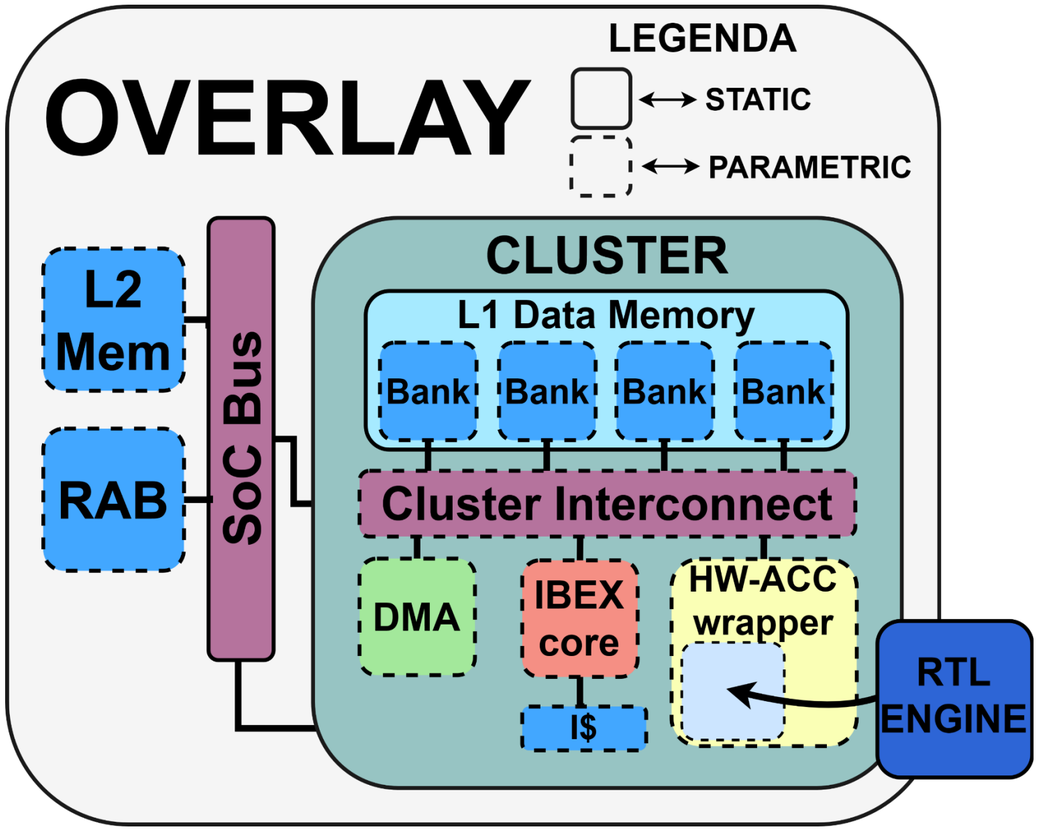
OODK
Architecture



OODK
Architecture

System Domain

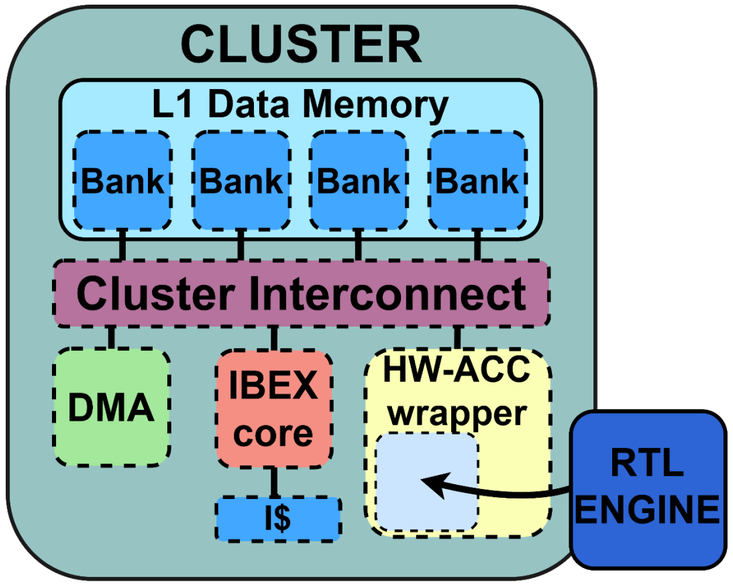
- Cluster
  - Multi and single-cluster architectures
  - Agile integration of different accelerators
- L2 memory
  - Data and instruction memory
- Remapping address block (RAB)
  - An IO-MMU for translation of virtual addresses
- SoC bus
  - Highly-scalable interconnect



OODK
Architecture

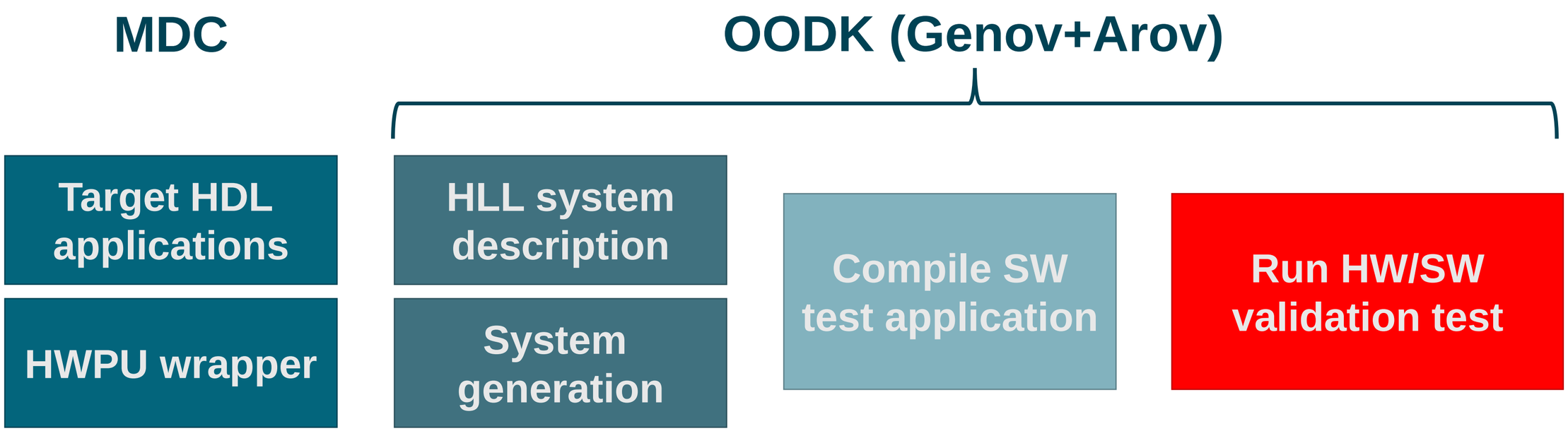
Cluster Domain

- HW accelerators
  - MDC-based HWPU
- RISC-V core
  - Tightly-coupled SW control - Accelerator routines, data management policies, etc.
  - L1 Instruction cache
- DMA
  - Specialized core for efficient L2 ↔ L1 data transfers
  - Support for 2D and 1D data transfers
- L1 data memory
  - Multi-banked scratchpad data memory (not a cache!)
- Cluster interconnect
  - Highly-scalable logarithmic interconnect + Peripheral bus



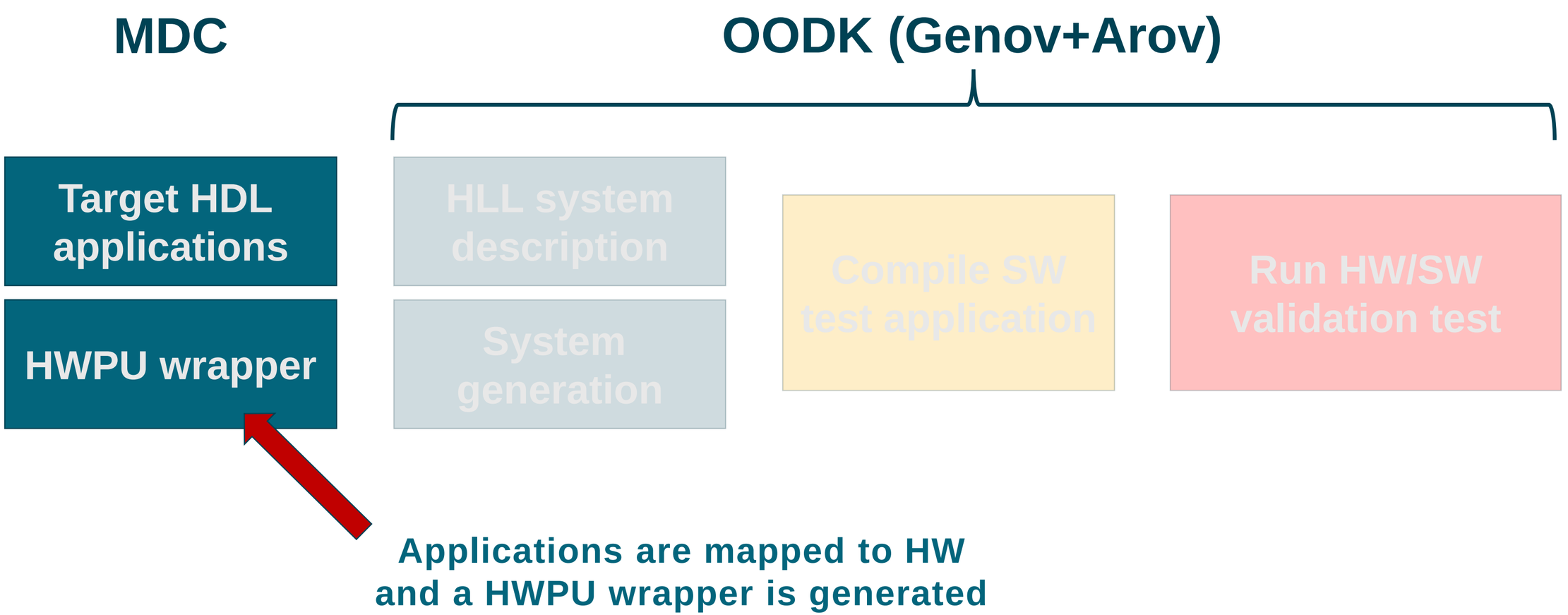
OODK

HW/SW co-design and verification tool



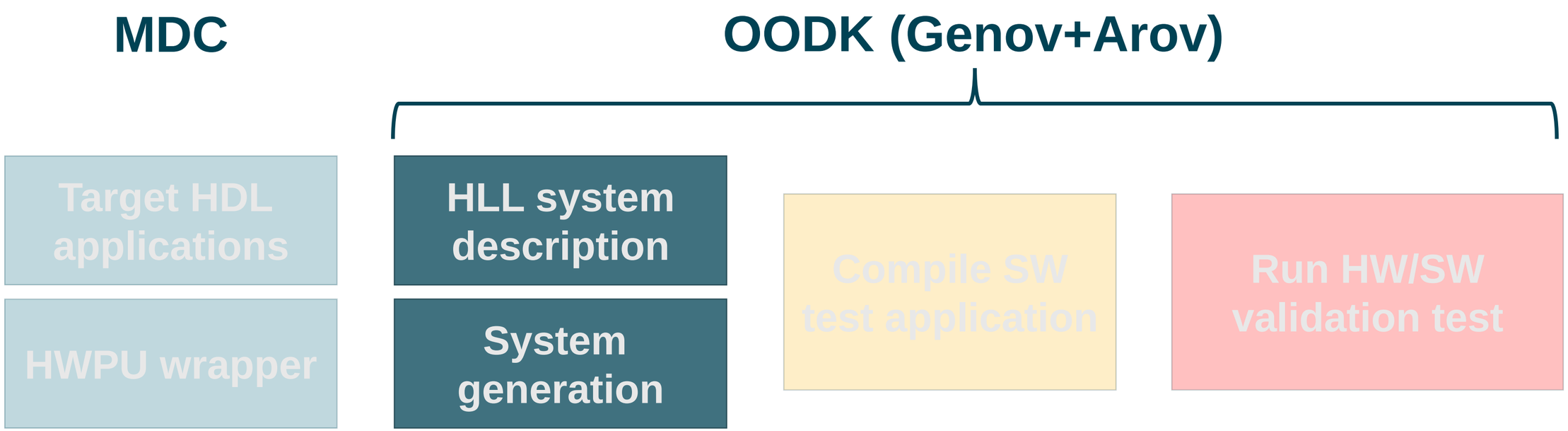
OODK

HW accelerator generation and integration



OODK

System generation

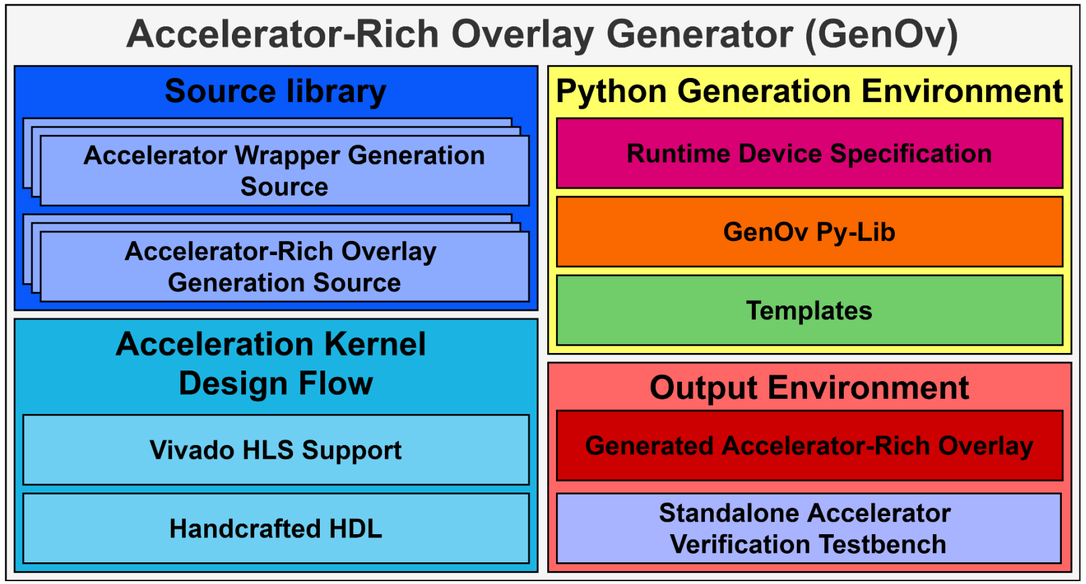


AROv (Accelerator-Rich Overlay)
GenOv (Generator Overlay)

COMP4DRONES

Download on Github:

https://github.com/gbellocchi/arov



OODK

System generation

Choose how to interconnecting accelerators is a primary requirement

- Which type of interconnect topology better fits our needs?
- What about the clustering level?
- How do accelerators mutually work?
  - Accelerators can either work in parallel or sequentially

Generation principles

- User knobs:
  - ***System optimization***
    - ✓ Memory hierarchy, control cores, DMA, etc.
    - ✓ Accelerator interconnections (generic vs. application-specific interconnects)
    - ✓ Accelerator scheduling (concurrent, serial or mixed scheduling)

OODK

System generation (spec.py)

1. System information
2. Cluster information
3. HW accelerators interconnection
   - ❖ Logarithmic interconnect
   - ❖ Heterogeneous interconnect

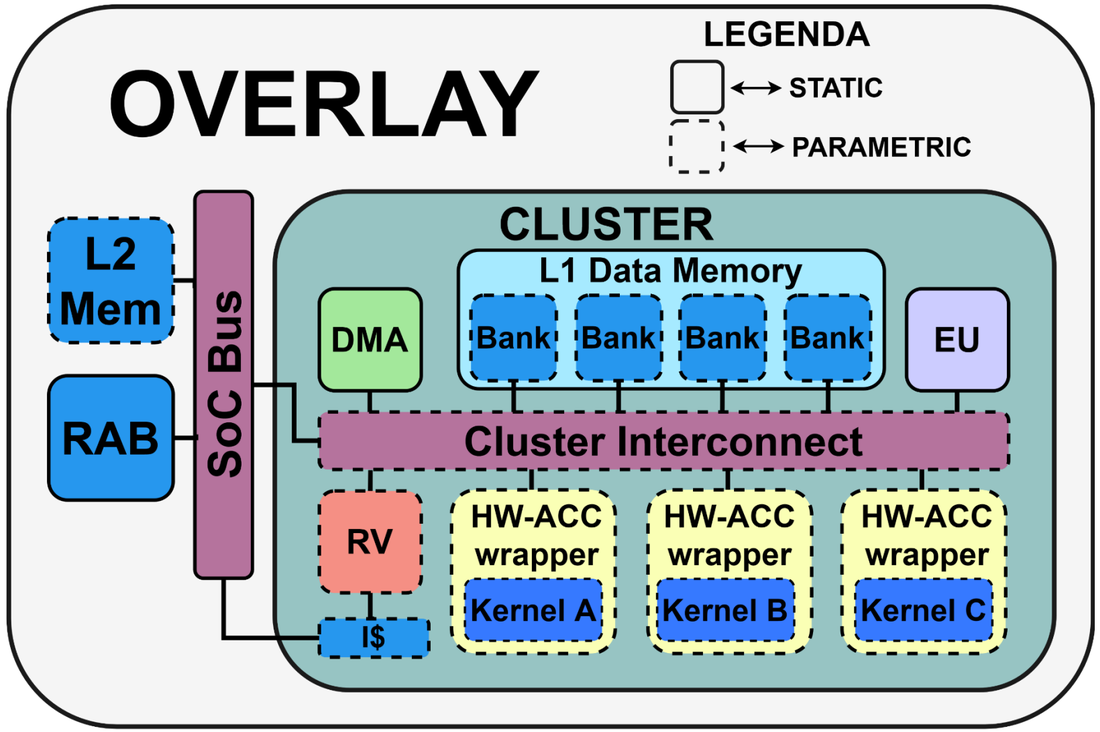
```
class oodk_specs:

    def system(self):
        self.oodk_config                    = 'ex_1_sys_gen'
        return self

    def cluster_0(self):
        self.cl_offset                      = 0
        self.core                           = [ 'ibex', 1 ]
        self.tcdm                           = [ 32 , 128]
        self.lic                            = [ [ 'kernel_A' , 'hwpu'],
                                                [ 'kernel_B' , 'hwpu'],
                                                [ 'kernel_C' , 'hwpu']]
        self.hci                            = [ ]
        return self
```

1
2
3

OODK

Example #1 – Connection to Cluster Interconnect

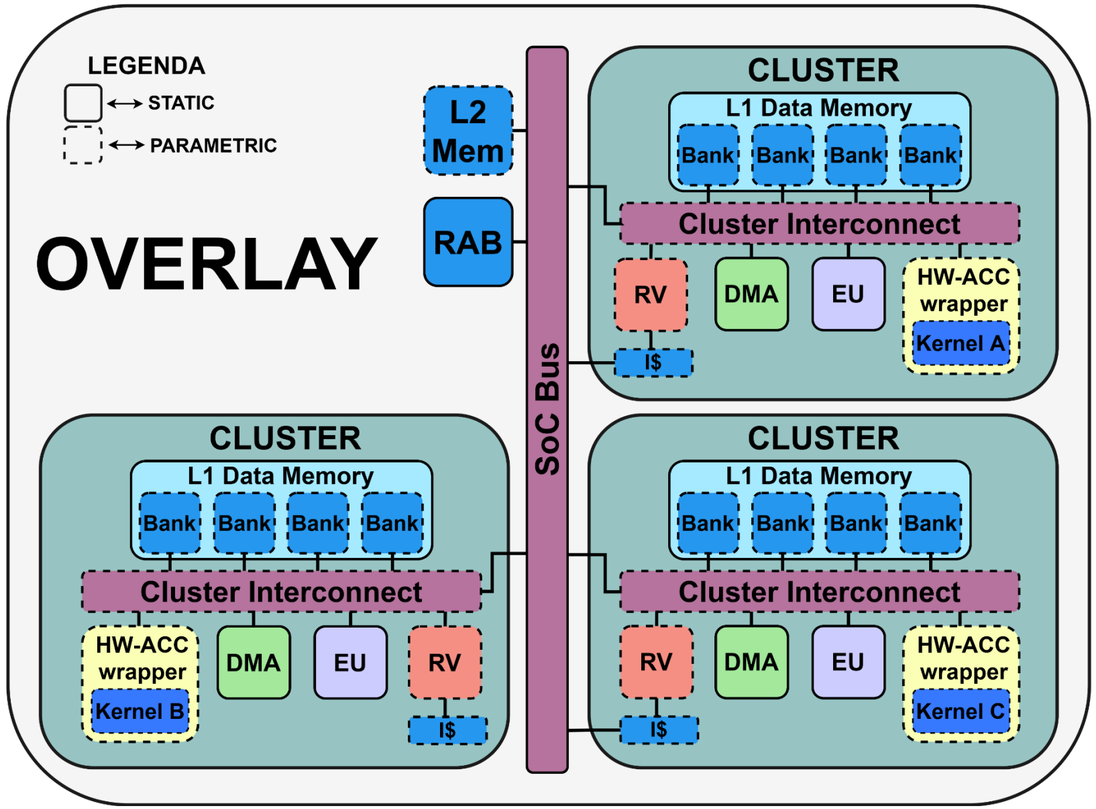
```
class oodk_specs:

    def system(self):
        self.oodk_config                    = 'ex_1_sys_gen'
        return self

    def cluster_0(self):
        self.cl_offset                      = 0
        self.core                           = [ 'ibex', 1 ]
        self.tcdm                           = [ 32 , 128]
        self.lic                            = [ [ 'kernel_A' , 'hwpu'],
                                                [ 'kernel_B' , 'hwpu'],
                                                [ 'kernel_C' , 'hwpu']]
        self.hci                            = [ ]
        return self
```



OODK

Example #2 – Multi-Cluster Interconnection

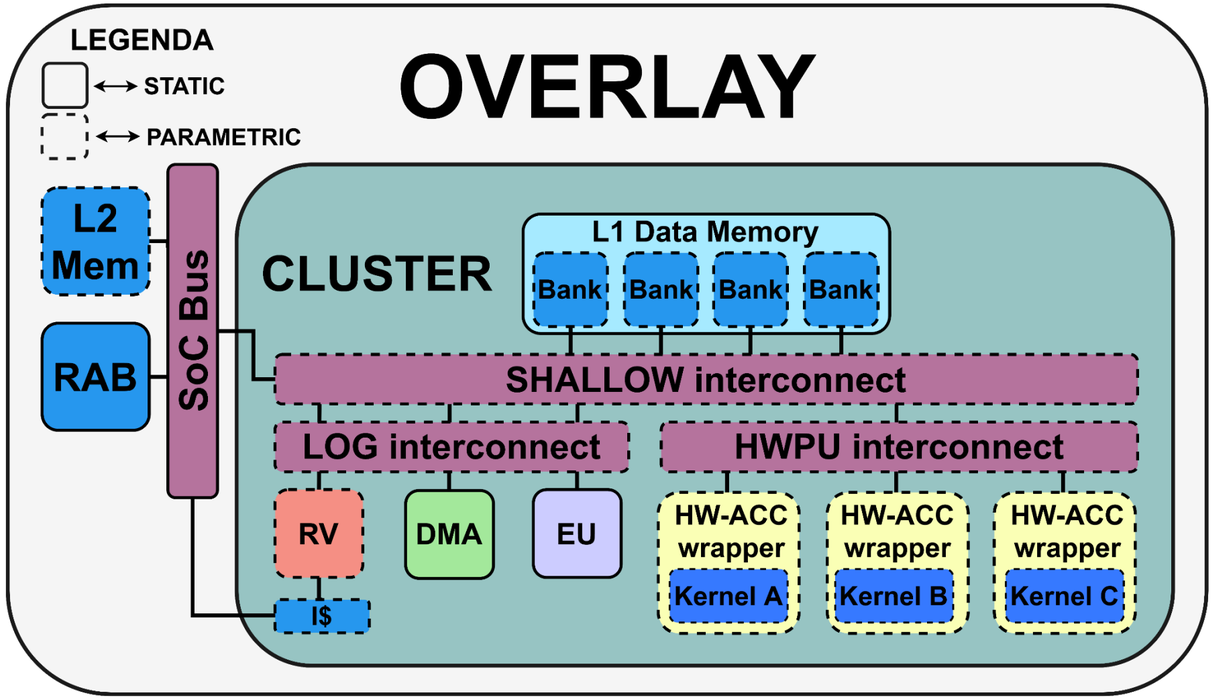
```
class oodk_specs:

    def system(self):
        self.oodk_config                = 'ex_2_sys_gen'
        return self

    def cluster_0(self):
        self.cl_offset                  = 0
        self.core                       = [ 'ibex', 1 ]
        self.tcdm                       = [ 32 , 128]
        self.lic                        = [ [ 'kernel_A' , 'hwpu']]
        self.hci                        = [ ]
        return self

    def cluster_1(self):
        self.cl_offset                  = 0
        self.core                       = [ 'ibex', 1 ]
        self.tcdm                       = [ 32 , 128]
        self.lic                        = [ [ 'kernel_B' , 'hwpu']]
        self.hci                        = [ ]
        return self

    def cluster_2(self):
        self.cl_offset                  = 0
        self.core                       = [ 'ibex', 1 ]
        self.tcdm                       = [ 32 , 128]
        self.lic                        = [ [ 'kernel_C' , 'hwpu']]
        self.hci                        = [ ]
        return self
```



OODK

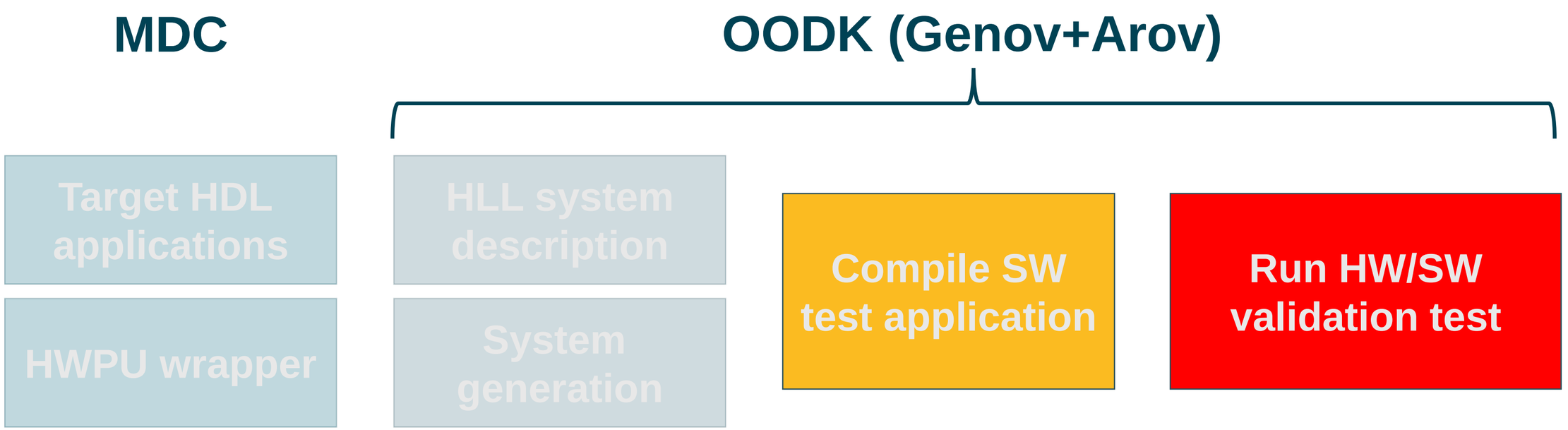
Example #3 – Heterogeneous Interconnection

```
class oodk_specs:

    def system(self):
        self.oodk_config                    = 'ex_3_sys_gen'
        return self

    def cluster_0(self):
        self.cl_offset                      = 0
        self.core                           = [ 'ibex', 1 ]
        self.tcdm                           = [ 32 , 128]
        self.lic                            = [ ]
        self.hci                            = [ [ 'kernel_A' , 'hwpu'],
                                                [ 'kernel_B' , 'hwpu'],
                                                [ 'kernel_C' , 'hwpu']]
        return self
```



OODK

System generation



OODK

System generation

Test application

- Baremetal software test
- Compiled for the OODK system
- A template version is generated together with the system itself

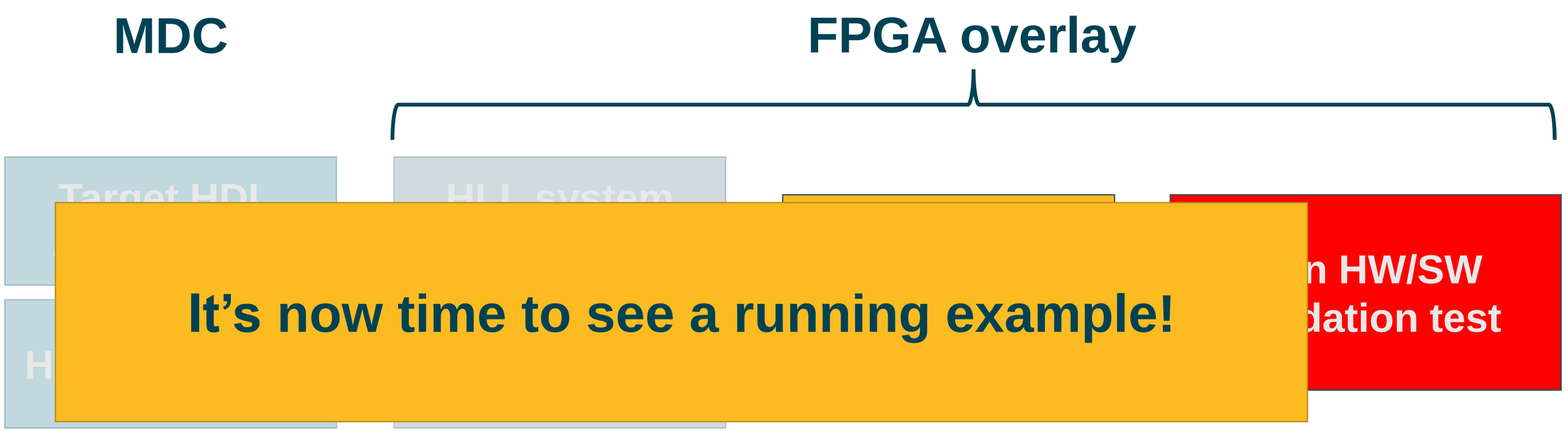
Accelerator Driver Generation

HW/SW validation test

- RTL simulation
  - Before to head up to the FPGA set-up, the generated designs are tested in QuestaSim testbench
  - The real behavior of the baremetal application is tested
    - The RISC-V core executes the test application
    - The accelerators functionality is validated with synthetic stimuli

OODK

System generation



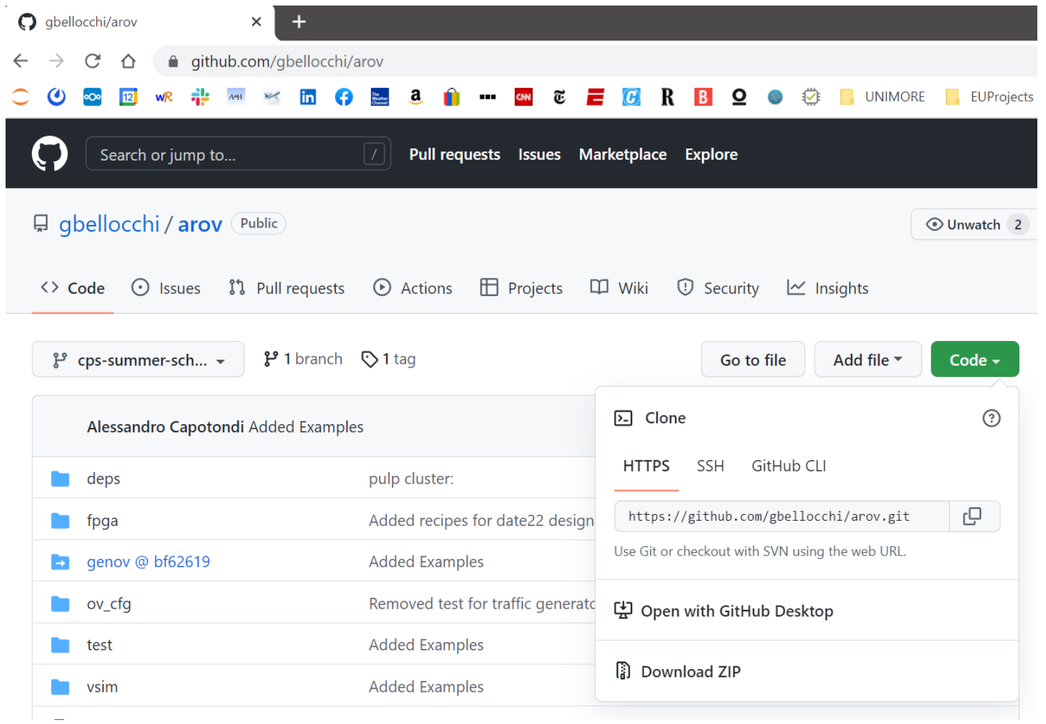
OODK Tutorial

Download Sources and Installation

Download Arov+Genov Repository

- Github:
  - https://github.com/gbellocchi/arov

- Open terminal
- mkdir oodk; cd oodk
- git clone https://github.com/gbellocchi/arov
- cd arov; source setup.sh
- git submodule update --init –recursive



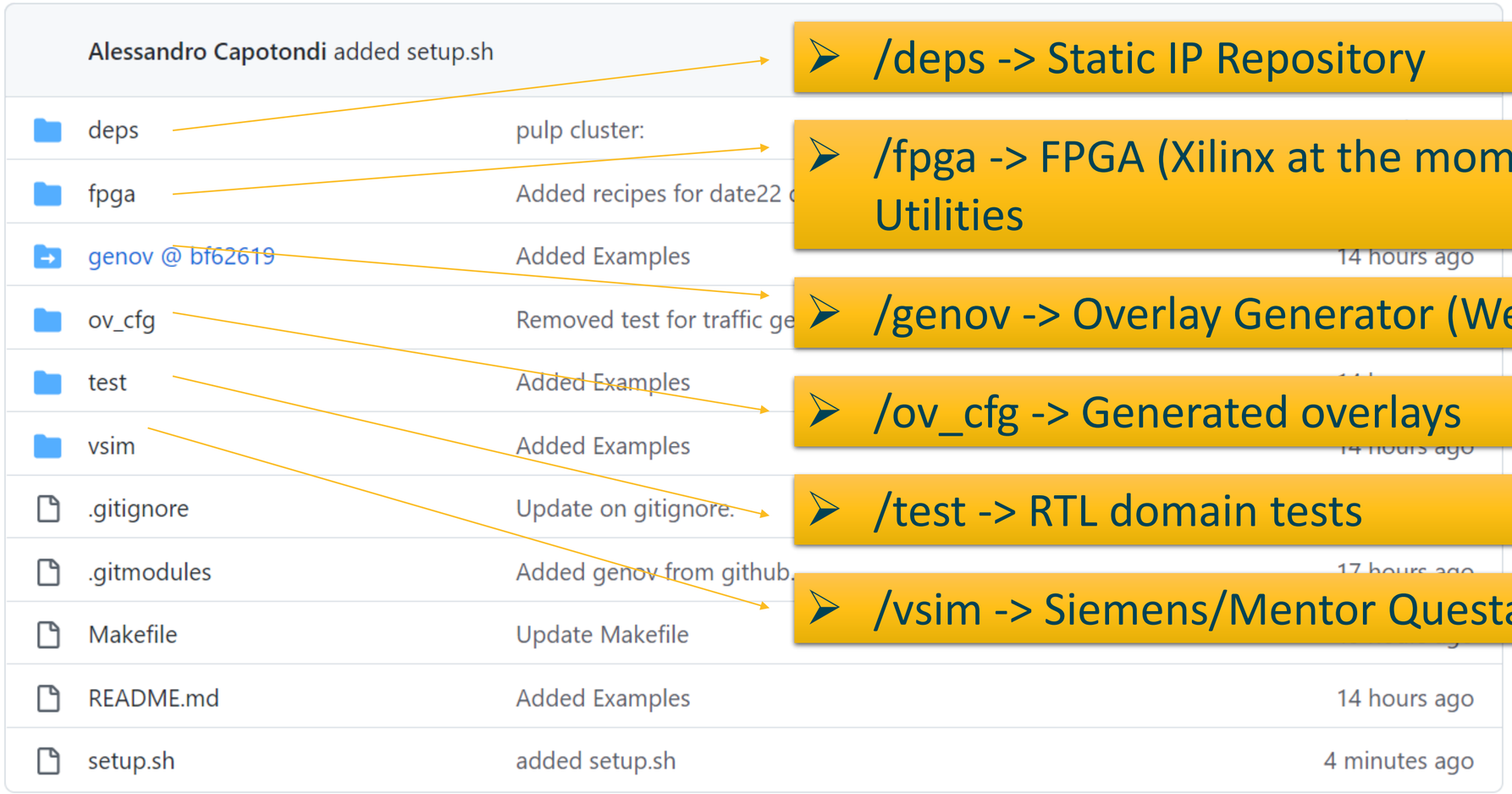
(optional) SW Development kit HERO Repository

- Github:
  - https://github.com/pulp-platform/hero

- Open terminal
- git clone https://github.com/pulp-platform/hero
- git checkout cps-school

OODK Tutorial

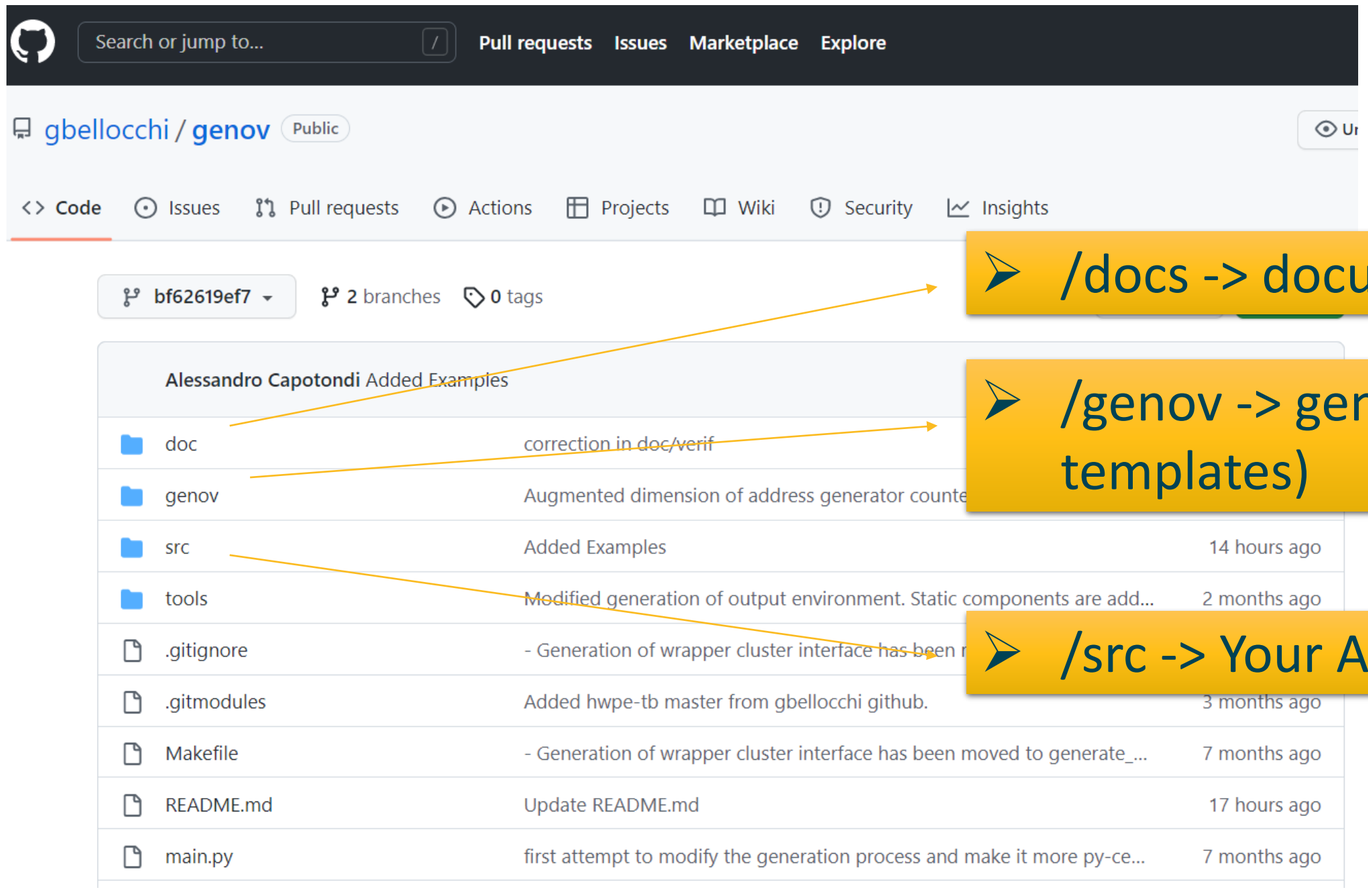
Arov (Accelerator Rich Overlay)



- /deps -> Static IP Repository
- /fpga -> FPGA (Xilinx at the moment) Build Scripts and Utilities
- /genov -> Overlay Generator (We will see later)
- /ov_cfg -> Generated overlays
- /test -> RTL domain tests
- /vsim -> Siemens/Mentor QuestaSim Utilities

OODK Tutorial

Genov (Generator of Overlay)



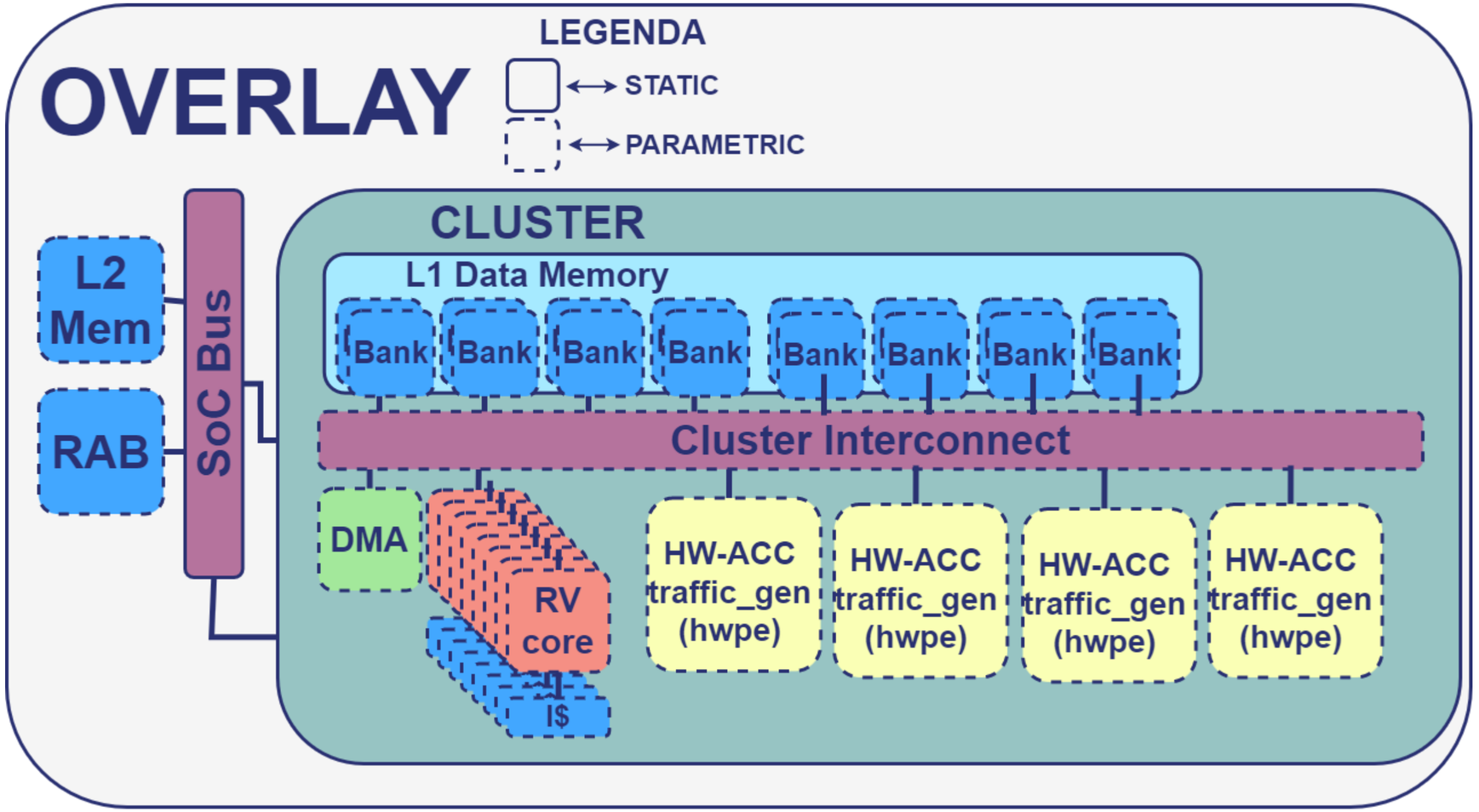
- /docs -> documentation
- /genov -> generators (python) + backend (IP templates)
- /src -> Your Accelerators Specs, Your Systems spcs

Exercise 1

Instantiate your first Overlay

1 Cluster

- **16xMem Banks L1**
- **128KB L1**
- **8x RISC-V ‘risky’ cores**
- **4x traffic_gen, hwpe Accelerators**



Exercise 1

Instantiate your first Overlay

Create:

exercise1/specs/ov_specs.py

```
    def cluster_0(self):
        self.cl_offset                          = 0
        self.core                               = [ 'riscy', 8 ]
        self.tcdm                               = [ 16 , 128]
        self.lic                                = [ [ 'traffic_gen' , 'hwpe'],
                                                    [ 'traffic_gen' , 'hwpe'],
                                                    [ 'traffic_gen' , 'hwpe'],
                                                    [ 'traffic_gen' , 'hwpe']
                                                  ]
        self.hci                                = [ ]
        return self
```

Exercise 1

Instantiate your first Overlay

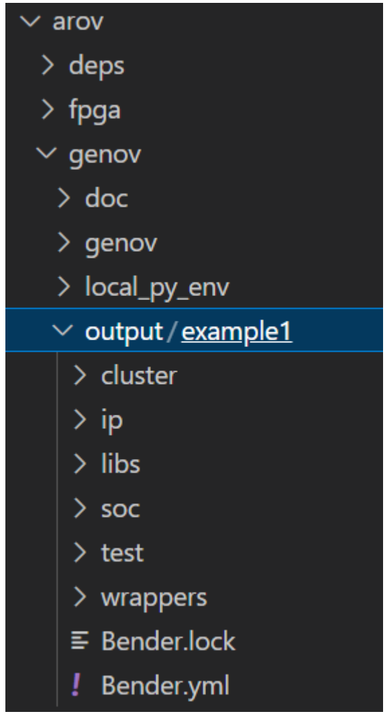
(First time only)

cd genov
make py_env
source local_py_env/bin/activate

Load Python Environment

cd genov
source local_py_env/bin/activate

Generate the Overlay



cd genov

make TARGET_OV=<OVERLAY FOLDER NAME> ov_gen

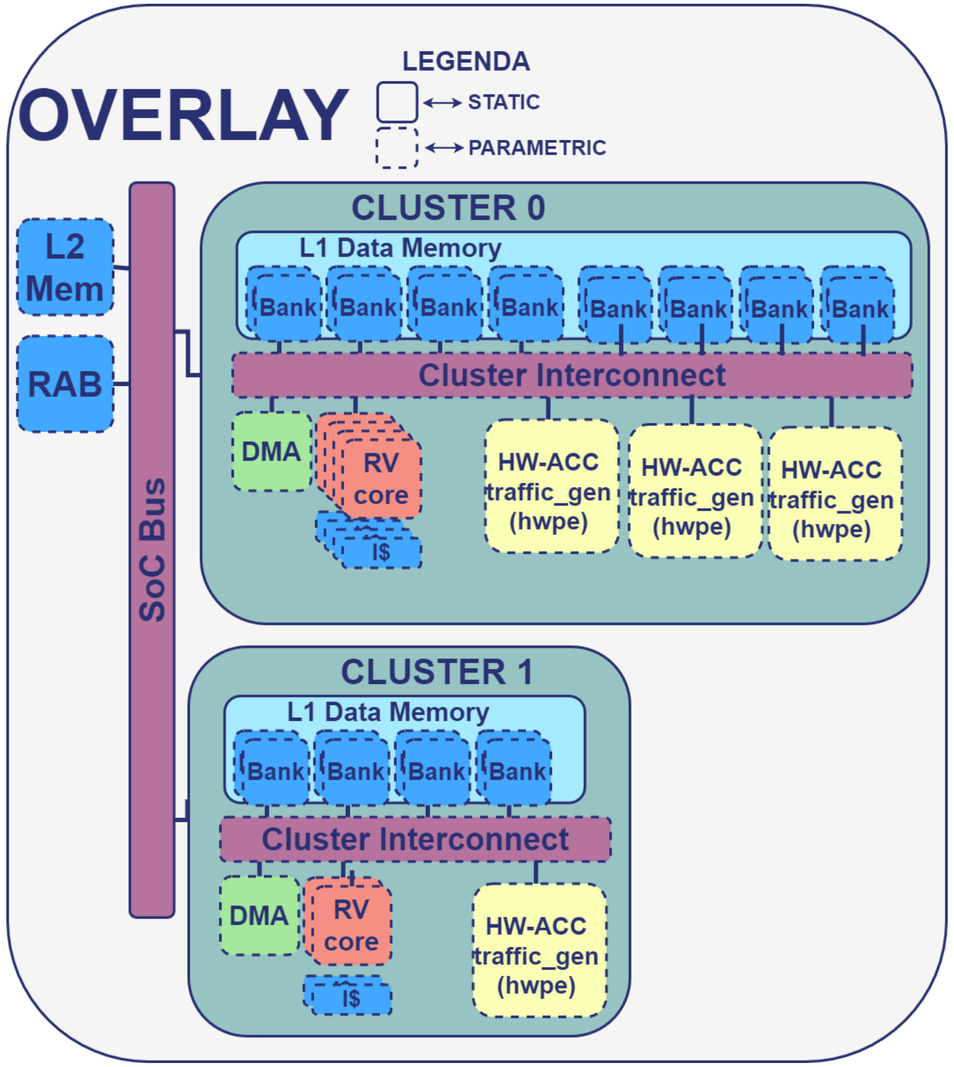
make TARGET_OV=example1 ov_gen #### in our specific case ####

Exercise 2

Instantiate your second Overlay!

2 Clusters

- **Cluster 0**
  - **16xMem Banks L1**
  - **128KB L1**
  - **4x RISC-V ‘risky’ cores**
  - **3x traffic_gen, hwpe Accelerators**
- **Cluster 1**
  - **8xMem Banks L1**
  - **128KB L1**
  - **2x RISC-V ‘risky’ cores**
  - **1x traffic_gen, hwpe Accelerators**



Exercise 2

exercise2/specs/ov_specs.py

```
    def cluster_0(self):
        self.cl_offset                          = 0
        self.core                               = [ 'riscy', 4 ]
        self.tcdm                               = [ 16 , 128]
        self.lic                                = [ [ 'traffic_gen' , 'hwpe'],
                                                    [ 'traffic_gen' , 'hwpe'],
                                                    [ 'traffic_gen' , 'hwpe']]
        self.hci                                = [ ]
        return self

    def cluster_1(self):
        self.cl_offset                          = 1
        self.core                               = [ 'riscy', 2 ]
        self.tcdm                               = [ 8 , 128]
        self.lic                                = [ [ 'traffic_gen' , 'hwpe'] ]
        self.hci                                = [ ]
        return self
```

cd genov

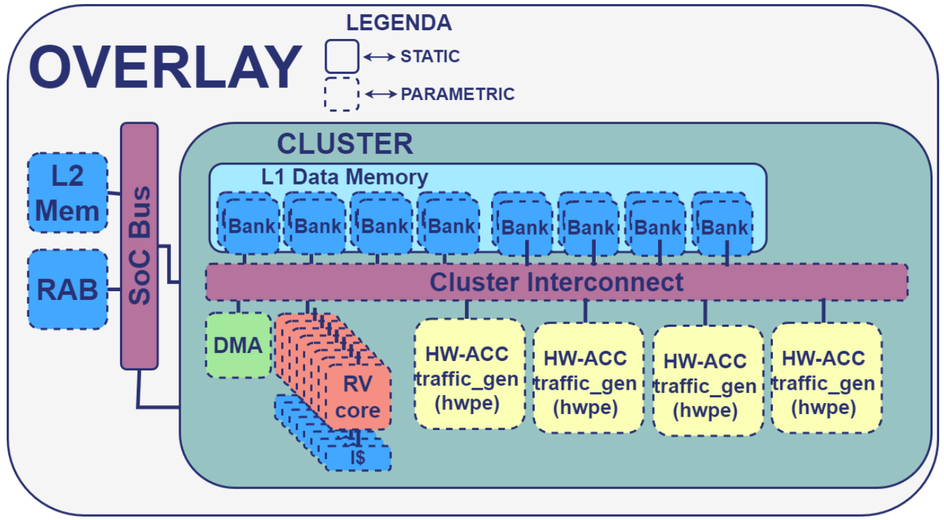
make TARGET_OV=example2 ov_gen

Ok, ok, but how to code this thing!

Helloworld on exercise1 Overlay

SW Requirements

- Installation of HERO SDK
  - Download prebuild in the release:
    - https://github.com/gbellocchi/arov/releases/tag/cps-summer-school-22-v0.2
  - Build from sources (takes time…)
    - https://github.com/pulp-platform/hero
    - git checkout cps-summer-school-22
    - Follow README.md



Where do we exploit OpenMP?

- Offload from the host of SoC computation to the overlay
- Parallel OpenMP pragma to PARALLELIZE execution on the RISC-V cores

OpenMP Helloworld

hero/openmp-example/helloworld/helloworld.c

```c
#include <hero-target.h>  // BIGPULP_MEMCPY
#include <stdio.h>        // printf()

#pragma omp declare target
void helloworld(void) {
#pragma omp parallel
  printf("Hello World, I am thread %d of %d\n", omp_get_thread_num(), omp_get_num_threads());
}
#pragma omp end declare target

int main(int argc, char *argv[]) {
#pragma omp target device(BIGPULP_MEMCPY)
  helloworld();

  return 0;
```

cd hero

source setup.sh

cd openmp-examples/helloworld/

make clean all *## build heterogenous application for board*

make clean all only=pulp ### build for RTL simulation (questasim) ← Use this today

Execute QuestaSim simulation*

COMP4DRONES

cd arov/genov

make TARGET_OV=example1 ov_deploy *## make design ready for deployment (simulation, build)*

cd ../

make TARGET_OV=example1 APP_PATH=/path/to/hero/openmp-examples/helloworld GUI=0 vsim

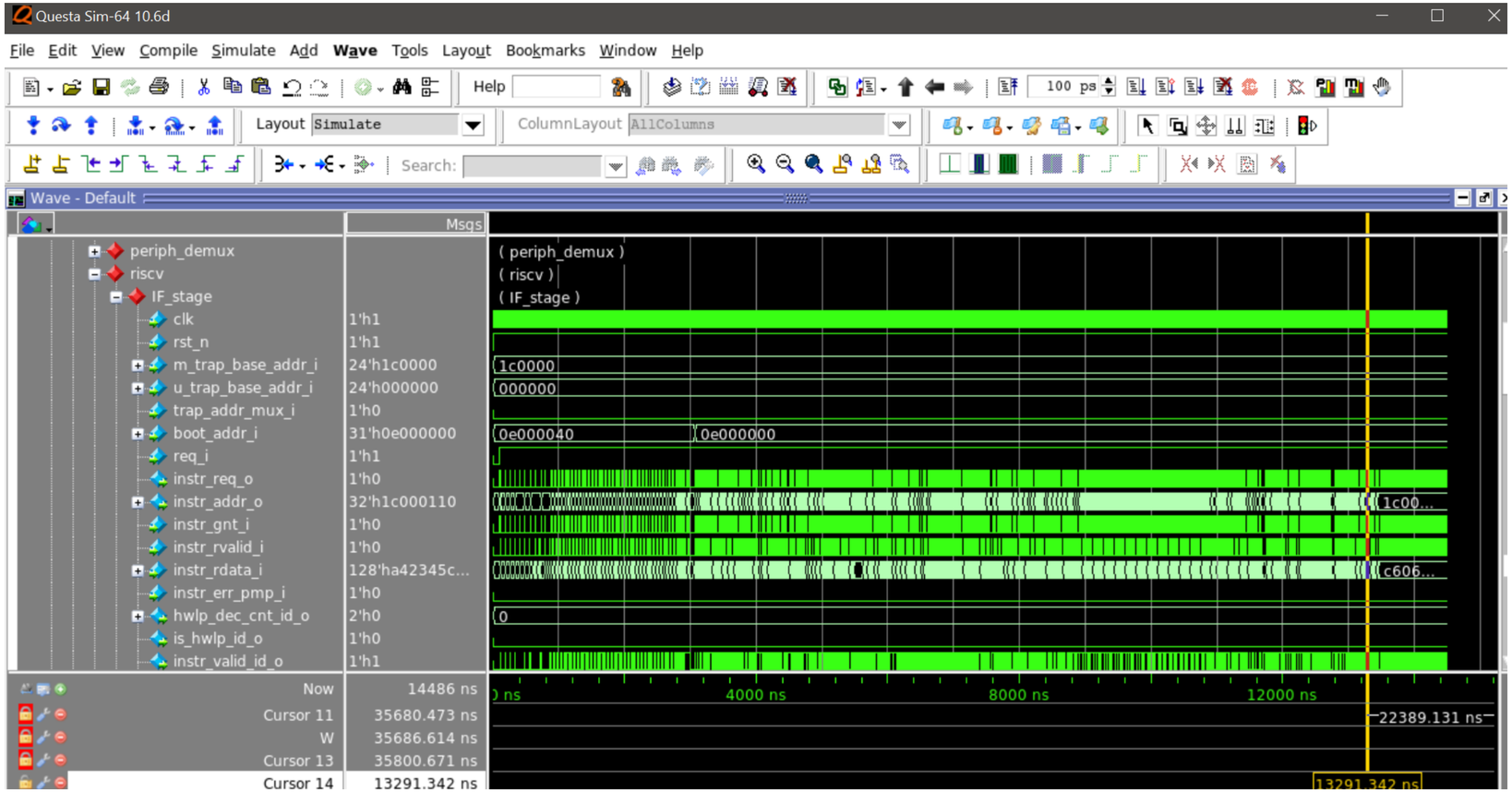
```
# [0,0] Hello World, I am thread 0 of 8
# [0,1] Hello World, I am thread 1 of 8
# [0,2] Hello World, I am thread 2 of 8
# [0,3] Hello World, I am thread 3 of 8
# [0,4] Hello World, I am thread 4 of 8
# [0,5] Hello World, I am thread 5 of 8
# [0,6] Hello World, I am thread 6 of 8
# [0,7] Hello World, I am thread 7 of 8
```

* Questasim installation is required. If you do not access to any modelsim simulator you can also use the IntelQuartus Edition https://www.intel.it/content/www/it/it/software/programmable/quartus-prime/questa-edition.html

Execute QuestaSim simulation*

cd ../

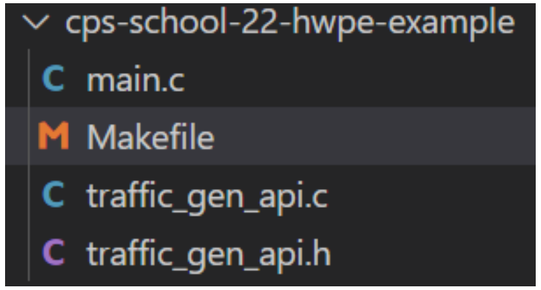
make TARGET_OV=example1 APP_PATH=/path/to/hero/openmp-examples/helloworld GUI=1 vsim



* Questasim installation is required. If you do not access to any modelsim simulator you can also use the IntelQuartus Edition https://www.intel.it/content/www/it/it/software/programmable/quartus-prime/questa-edition.html

Traffic Gen Accelerator example

/hero/openmp-examples/cps-school-22-hwpe-example




```c
#pragma omp parallel
{
  #pragma omp master
  {
    printf("I am the master, and I am going to program the accelerator\n");
    arov_struct arov;
    int offload_id;
    int cluster_id = 0;
    int acc_id = 0;

    __device uint32_t * a_local = (__device uint32_t *)hero_l1malloc(1024*sizeof(uint32_t));

    printf("Initialized the Traffic Gen %d\n", acc_id);
    arov_init(&arov, cluster_id, acc_id);

    printf("Prepare Traffic Gen %d Job descriptor\n", acc_id);
    arov_map_params_traffic_gen(&arov, cluster_id, acc_id,
      a_local, /* buffer_l1_base_pointer */
      1024,    /* i/o size in word (I/O) */
      512,     /* input size in word */
      1, 1, 512, /* total tx generated */
      1, 1, 512, 1,/* n_reps */
      16);     /* n_banks touched */

    printf("Program Traffic Gen %d\n", acc_id);
    offload_id = arov_activate(&arov, cluster_id, acc_id);
    arov_program(&arov, cluster_id, acc_id);
```

cd hero

source setup.sh

cd openmp-examples/cps-school-22-hwpe-example

make clean all *## build heterogenous application for board*

make clean all only=pulp ### build for RTL simulation (questasim) ← Use this today

Execute QuestaSim simulation*

cd ../

make TARGET_OV=example1 APP_PATH=/path/to/hero/openmp-examples/cps-school-22-hwpe-example GUI=0 vsim

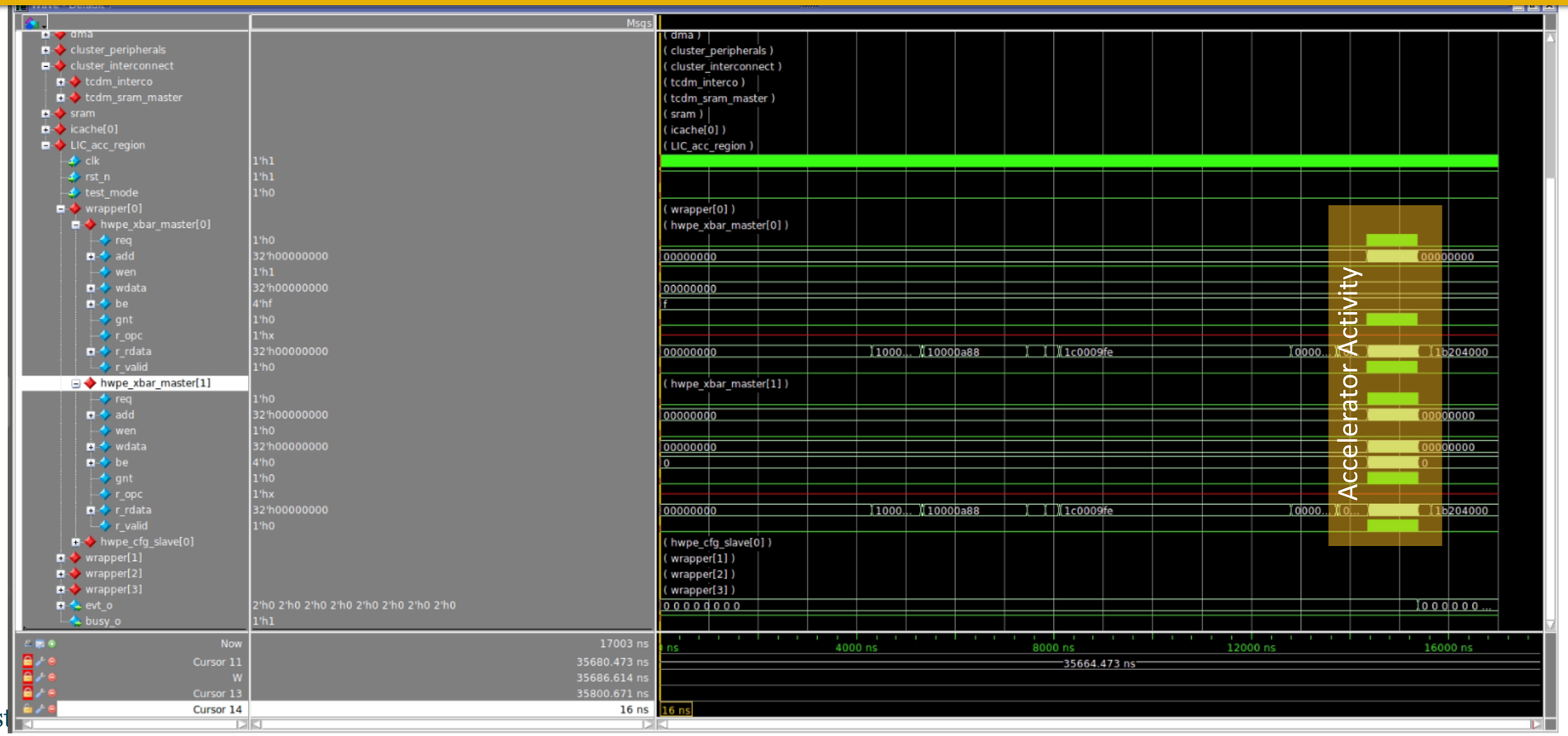
```
# [0,1] I am waiting for the termination of all the threads
# [0,3] I am waiting for the termination of all the threads
# [0,2] I am waiting for the termination of all the threads
# [0,4] I am waiting for the termination of all the threads
# [0,5] I am waiting for the termination of all the threads
# [0,7] I am waiting for the termination of all the threads
# [0,6] I am waiting for the termination of all the threads
# [0,0] I am the master, and I am going to program the accelerator
# [0,0] Initialized the Traffic Gen 0
# [0,0] Prepare Traffic Gen 0 Job descriptor
# [0,0] Program Traffic Gen 0
# [0,0] Start Traffic Gen 0
# [0,0] Wait for termination Traffic Gen 0
# [0,0] Traffic Gen 0 execution is complete!
# [0,0] I am waiting for the termination of all the threads
# [0,0] That s all folks!
# [0,1] That s all folks!
# [0,3] That s all folks!
# [0,2] That s all folks!
# [0,4] That s all folks!
# [0,5] That s all folks!
# [0,6] That s all folks!
# [0,7] That s all folks!
```

* Questasim installation is required. If you do not access to any modelsim simulator you can also use the IntelQuartus Edition https://www.intel.it/content/www/it/it/software/programmable/quartus-prime/questa-edition.html

Execute QuestaSim simulation*

cd ../

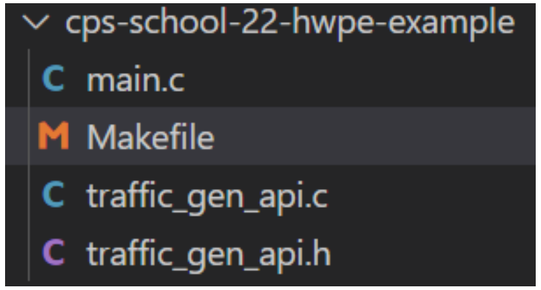
make TARGET_OV=example1 APP_PATH=/path/to/hero/openmp-examples/cps-school-22-hwpe-example GUI=1 vsim



* Questasim inst
https://www.intel.it/content/www/it/it/software/programmable/quartus-prime/questa-edition.html

Traffic Gen Accelerator example

/hero/openmp-examples/cps-school-22-hwpe-example




```c
#pragma omp parallel
{
  #pragma omp master
  {
    printf("I am the master, and I am going to program the accelerator\n");
    arov_struct arov;
    int offload_id;
    int cluster_id = 0;
    int acc_id = 0;

    __device uint32_t * a_local = (__device uint32_t *)hero_l1malloc(1024*sizeof(uint32_t));

    printf("Initialized the Traffic Gen %d\n", acc_id);
    arov_init(&arov, cluster_id, acc_id);

    printf("Prepare Traffic Gen %d Job descriptor\n", acc_id);
    arov_map_params_traffic_gen(&arov, cluster_id, acc_id,
      a_local, /* buffer_l1_base_pointer */
      1024,    /* i/o size in word (I/O) */
      512,     /* input size in word */
      1, 1, 512, /* total tx generated */
      1, 1, 512, 1,/* n_reps */
      16);     /* n_banks touched */

    printf("Program Traffic Gen %d\n", acc_id);
    offload_id = arov_activate(&arov, cluster_id, acc_id);
    arov_program(&arov, cluster_id, acc_id);
```

cd hero

source setup.sh

cd openmp-examples/cps-school-22-hwpe-example

make clean all *## build heterogenous application for board*

make clean all only=pulp ### build for RTL simulation (questasim) ← Use this today

But, where are the FPGA????

OODK provides support also for synthesis and implementation on FPGA*

Avnet Ultra96

Xilinx Kria
KV260

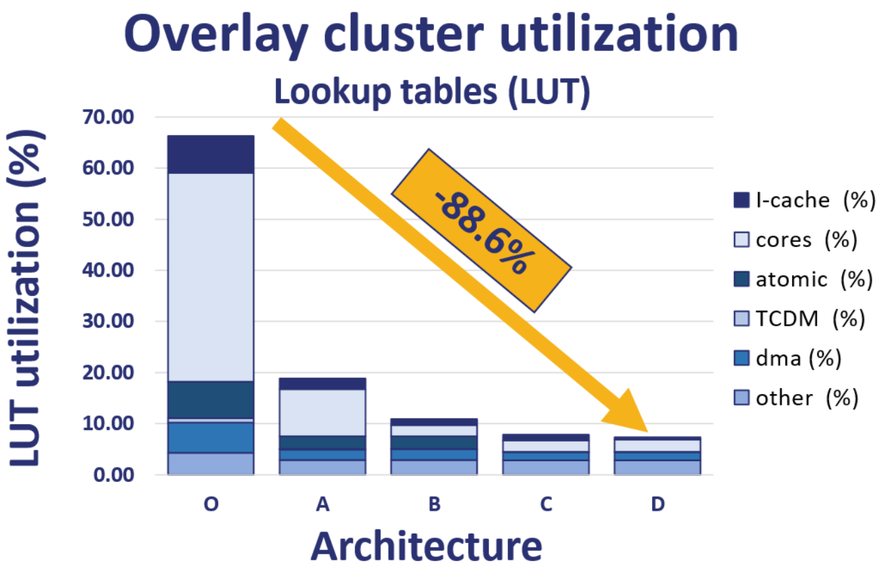
Xilinx
ZCU102
ZCU104
ZCU106

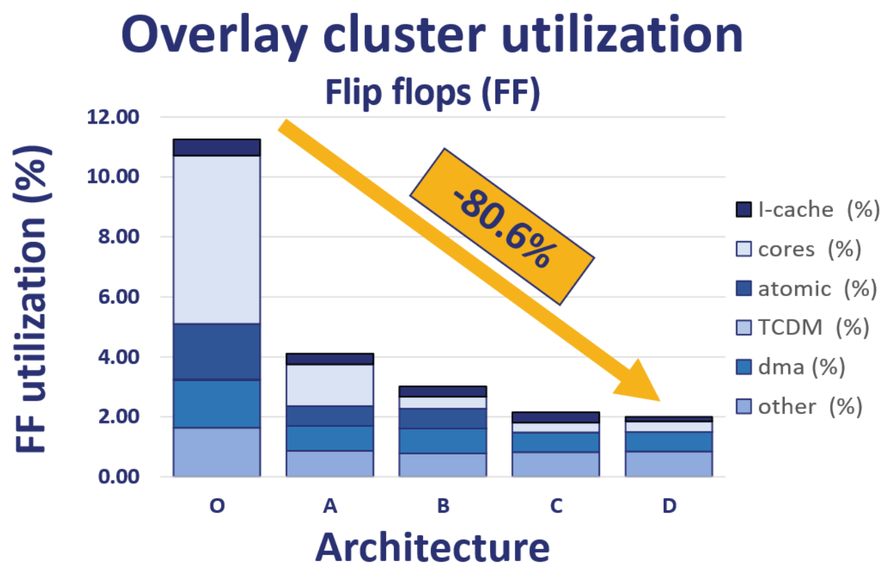
cd arov
make TARGET_OV=example1 fpga

* Implementation and synthesis requires Xilinx Vivado Installation and Valid License for the target board.

What can you do with this tool???

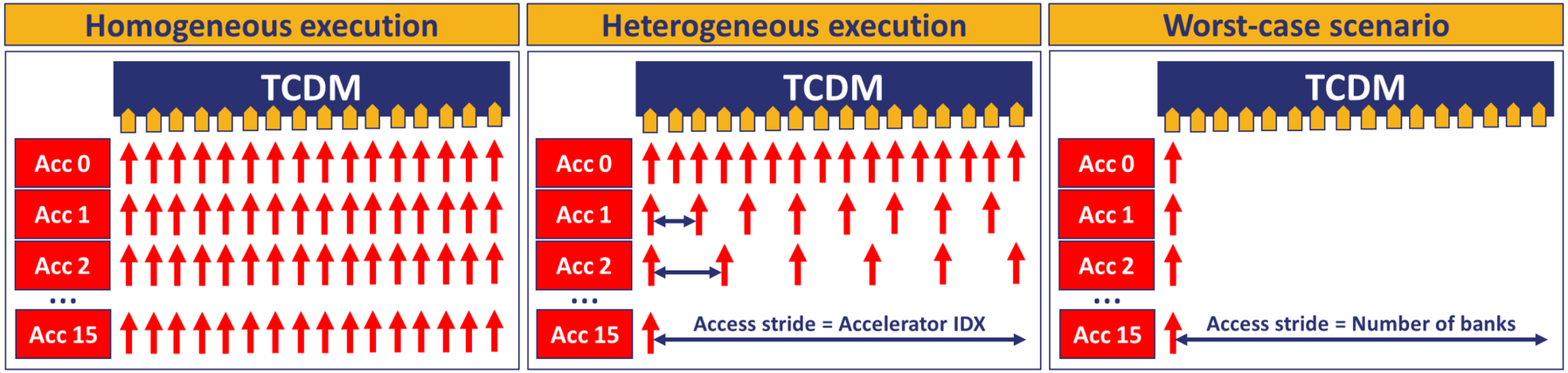
Automated resource Space Exploration

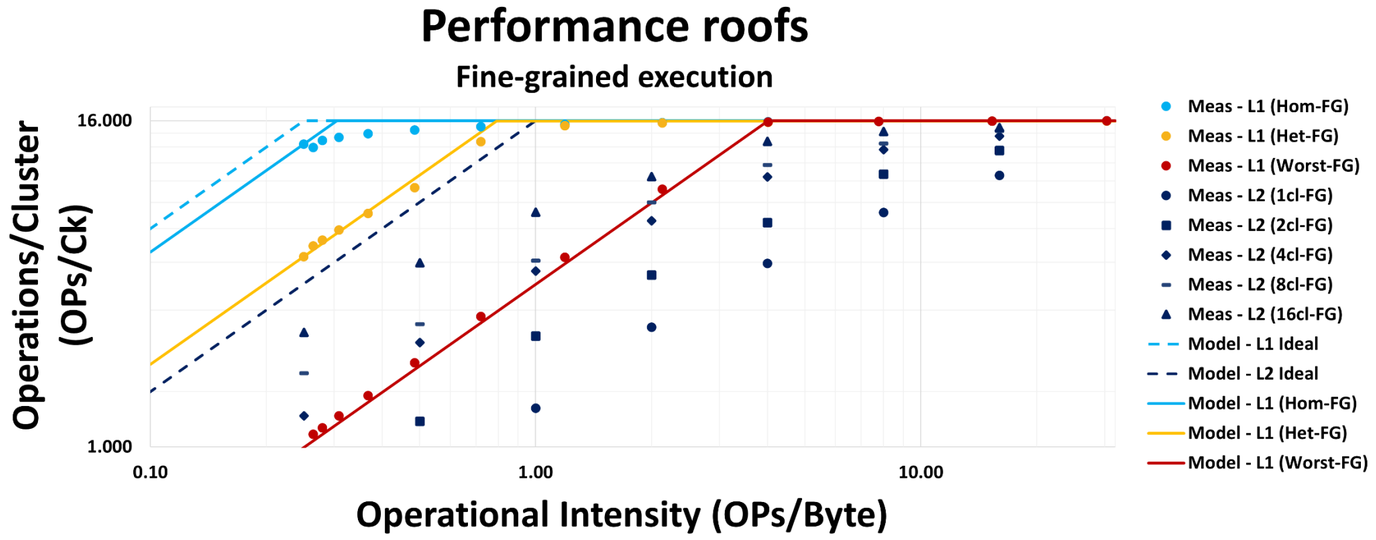




What can you do with this tool???

Automated Performance Evaluation





AGENDA

Current application: C4D

Current application: C4D

Development and assessment of Smart and Precision Agriculture Technologies to enable:

1. **Improve non-real time actions**, i.e. forecast on production volume and optimized water management.
2. **Real-time field monitoring and inspection**, i.e. automatic disease detection and cross-correlation of plants indexes;
3. **Prompt on-field intervention**, i.e. customized spot spraying;

Current application: C4D

Development and assessment of Smart and Precision Agriculture Technologies to enable:

1. **Improve non-real time actions**, i.e. forecast on production volume and optimized water management.
2. **Real-time field monitoring and inspection**, i.e. automatic disease detection and cross-correlation of plants indexes;
3. **Prompt on-field intervention**, i.e. customized spot spraying;

TECHNICAL SET-UP

Tandem of cooperative autonomous vehicles composed of a field rover, responsible of gathering and processing field data, and a spraying drone

Current application: C4D motivation

Current application: C4D motivation

USER NEEDS

1. **Use as little pesticides**: Proper assessment of health status & on spot interventions
2. **Waste as little water as possible**: Precise growth assessment

Current application: C4D motivation

USER NEEDS

1. **Use as little pesticides**: Proper assessment of health status & on spot interventions
2. **Waste as little water as possible**: Precise growth assessment

Current application: C4D motivation

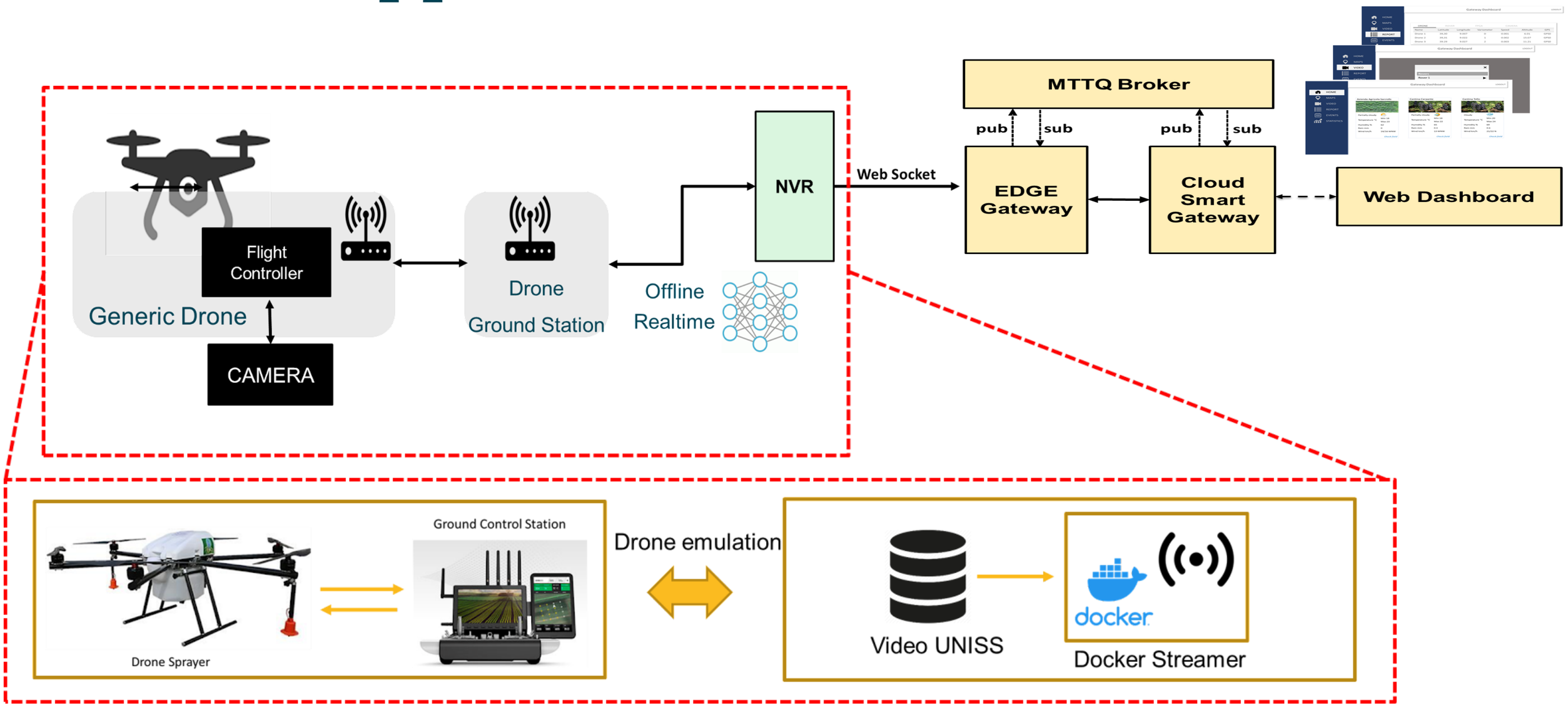
EXPECTED BENEFITS

1. Reduced impact on the environment
2. Reduced human effort
3. Improved usability of advanced technologies by non-expert operators

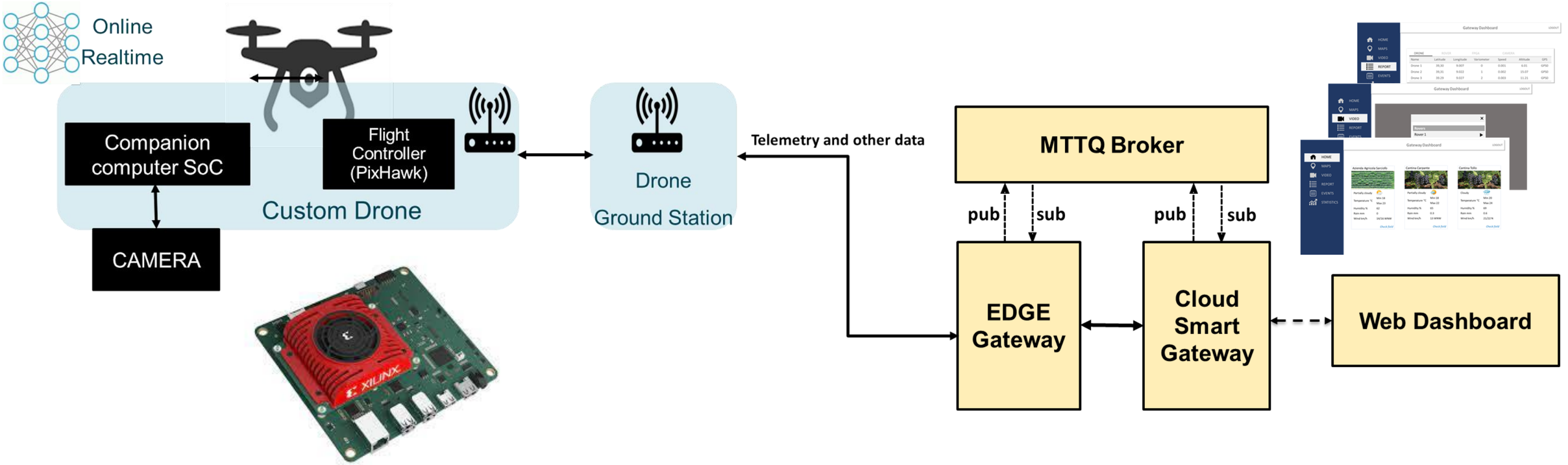
USER NEEDS

1. **Use as little pesticides**: Proper assessment of health status & on spot interventions
2. **Waste as little water as possible**: Precise growth assessment

Current application: Baseline



Current application: Scenario 2



Current application: Scenario 3

Companion computer SoC

Flight Controller (PixHawk)

Custom Drone

CAMERA

Drone Ground Station

Telemetry and other data

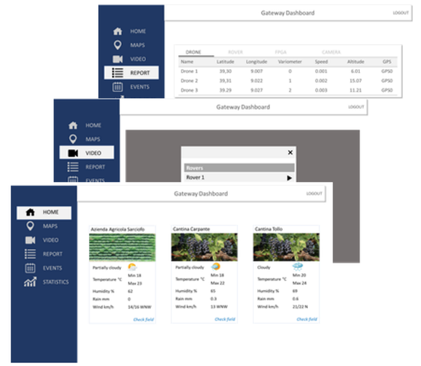
MTTQ Broker

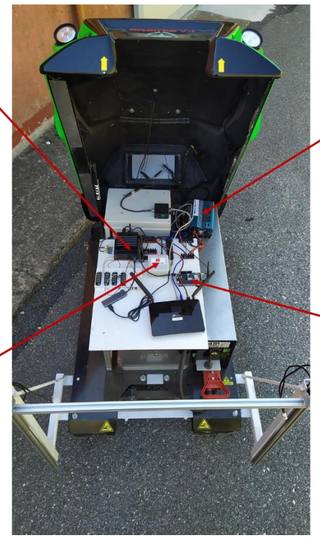
pub sub pub sub

EDGE Gateway

Cloud Smart Gateway

Web Dashboard





ON-BOARD COMPUTER

Current platform is Nvidia AGX Xavier (link for specs)

INVERTER

GPS MODULE

IMU

Companion computer SoC

Controller

Telemetry and other data

Rover

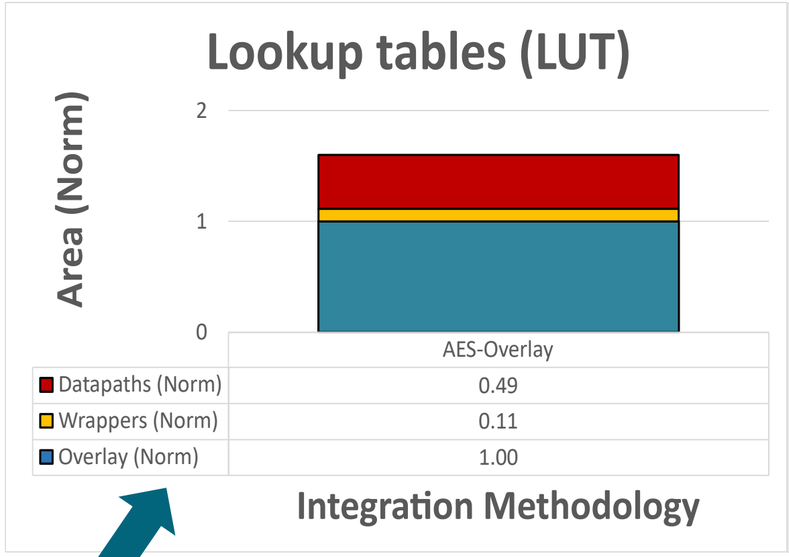
LASER

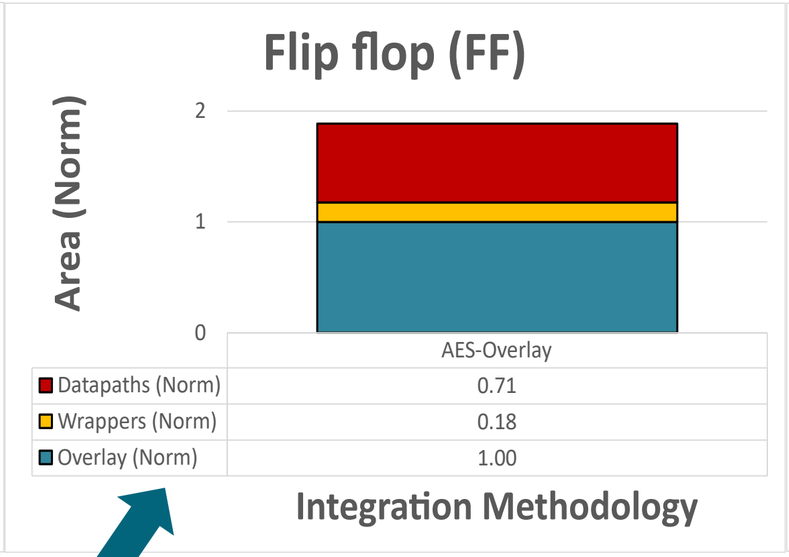
GPS ANTENNA

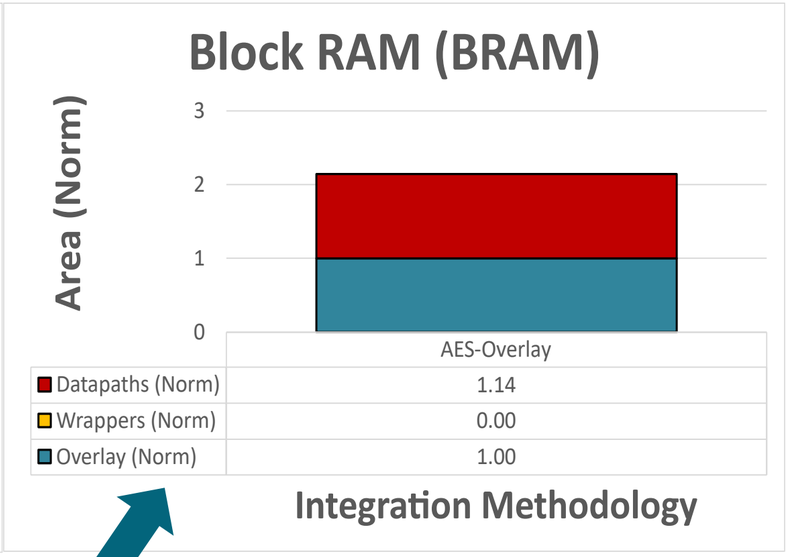
SUPPORT BAR (to hold further sensors)

C4D methodology experimental results

- ✓ **Overall x2 speedup** when comparing SW vs HW implementation of the AES algorithm
- ✓ Implementation targets a ZU9EG SoC with a resource cost of:
  - ➢ **~43.7% LUTs**
  - ➢ **~11.7% FFs**
  - ➢ **~13.2% BRAMs**







Normalized to the overlay occupation

AGENDA

Conclusions

- ✓ Simplified design of HW accelerators through MDC
- ✓ Multi-functionality, multi working-point and reconfiguration support for CGRAs
- ✓ Support for accelerators generated with different tools (e.g., CAPH, HLS)
- ✓ Agile methodology for the design and exploration of accelerator-rich systems
- ✓ Simplified validation and deployment of the generated HW/SW system
- ✓ Practical use case: COMP4DRONES

Give us a feedback!

https://www.menti.com/al4bhr2njxrh

Contacts

Prof. Alessandro Capotondi
- ✓ Alessandro.capotondi@unimore.it

Dr. Daniel Madroñal
- ✓ dmadronalquin@uniss.it

Ing. Gianluca Bellocchi
- ✓ gianluca.bellocchi@unimore.it

Prof. Andrea Marongiu
- ✓ a.marongiu@unimore.it

Prof. Francesca Palumbo
- ✓ fpalumbo@uniss.it

CPS Summer School 2022

Tutorial C4D:
A programmable and reconfigurable FPGA overlay

Alessandro Capotondi[1], Daniel Madroñal[2]

[1]Università degli Studi di Modena e Reggio Emilia
[2]Università degli Studi di Sassari

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826610. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Austria,

COMP4DRONES