Hands On: The CERBERO Design Flow for Adaptive Heterogeneous Embedded Systems

This tutorial aims at teaching how to use the CERBERO toolchain for porting an application on a heterogeneous embedded architecture, embedding hard-cores and an FPGA substrate. The leveraged example hardware is a Xilinx Zynq board and the chosen educational application is an image edge detection filter. The CERBERO toolchain demonstrated in this tutorial is composed of:

- **ARTICo^3** (`https://github.com/des-cei/artico3`): an open source tool for automating hardware acceleration over reconfigurable logic regions on an FPGA,
- **MDC** (`https://github.com/mdc-suite`): an open source tool for managing coarse grain reconfiguration of dataflow hardware-ported applications,
- **PAPIFY** (`https://github.com/Papify`): an open source tool for live performance monitoring on parallel and heterogeneous systems,
- **PREESM** (`https://github.com/Preesm`): an open source tool for dataflow application design and parallel systems programming.

The following tools are also used as back-end code production tools in this tutorial:

- The **CAPH Compiler** (`http://caph.univ-bpclermont.fr`): an open source tool for generating RTL from the higher-level CAPH language,
- The **Open RVC-CAL Compiler** (https://github.com/orcc): an open source tool for generating RTL from the higher-level CAL language,
- The **Xilinx Vivado** design suite for generating FPGA bitstream and programming the target board.

These different tools are integrated for providing seamless porting of dataflow applications to heterogeneous hardware. One may note that the tools developed in this tutorial are only a part of the CERBERO project tooling.

The procedure to follow for building an adaptive system with the demonstrated CERBERO design flow is:

- Design static versions of your dataflow applications (i.e. with fixed parameter values and fixed topology) with either PREESM for coarse-grain dataflow actors or CAPH for small-grain, hardware optimized dataflow actors.
- Exploit PAPIFY performance monitoring to observe the resulting system performance.
- Exploit ARTICo^3 and MDC reconfigurable hardware management capacity to enhance system performance and safety.

The tutorial is divided into three parts. In the first part, a dataflow-based application, developed using PREESM, is shown and explained. PREESM is in charge of automatically dispatching jobs among available hardware resources (CPUs and/or FPGA slots). A second part shows how to generate and automatically instrument code in order to monitor the whole hardware infrastructure by using PAPIFY. In a third part, we create a hardware accelerator with the MDC tool which is compliant with the ARTICo^3 processing architecture. The system bitstreams are created using Xilinx Vivado, invoked by the ARTICo^3 toolchain.

1 Hardware/Software Model-Based Design for Design Space Exploration and Code Generation

In this part of the tutorial, we observe how to design an application and model a hardware architecture for design space exploration and code generation purposes. The application is described with the dataflow model of computation named Parameterized and Interfaced Synchronous DataFlow (PiSDF), and within the PREESM tool using a graphical user interface. The architecture is also modeled graphically with the System-Level Architecture Model (S-LAM). The main idea of these models is to make them coarse grain and minimal for system-level decisions. A *scenario* ensures PiSDF and S-LAM models independance, gathering information that relates to both.

1.1 PiSDF Application Design

The educational considered application is an edge detection application combining two different algorithms: Sobel image filtering and Roberts image filtering.

The algorithm description is based on a dataflow MoC optimized for expressing predictable concurrency. The PiSDF MoC is a graph that connects *Actors* and *Parameters* through *FIFO* and *Parameter dependency* links. Processing is triggered by data arrival and data rates and actor firings can be impacted by parameter value modifications.

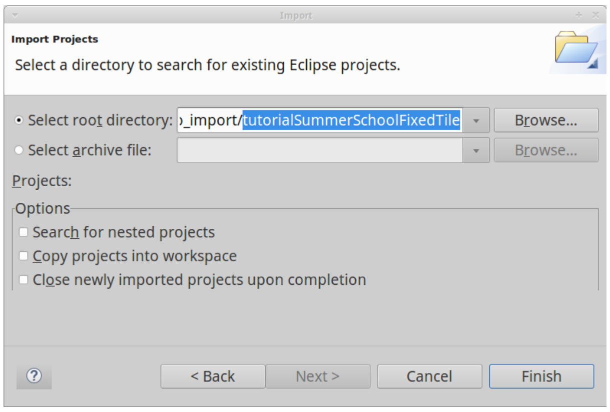
1. **Open PREESM**:
   Within the folder:
   `> /home/embedded/Desktop/preesm-3.17.0.201909161224-linux.gtk.x86_64/`
   open PREESM by double-clicking on
   `> eclipse`

2. **Import the template project**:
   The project created for this tutorial is located within the folder
   `> /home/embedded/Desktop/preesm_project/tutorial`
   In the `> Project Explorer` panel, click on the `> Import projects`.

   Then, in the appearing wizard window, select:
   `> General > Existing Project into Workspace > Next`

   Select root directory:
   `> /home/embedded/Desktop/preesm_project/tutorial/tutorialSummerSchoolFixedTile`
   and press `OK` and `Finish`:

   Within the `Algo` folder, the algorithm is described with a PiSDF compliant graph. Within the `Archi` folder, the hardware architecture is described with an S-LAM compliant graph. Information



Import

Import Projects

Select a directory to search for existing Eclipse projects.

- [✓ Select root directory:] ›_import/tutorialSummerSchoolFixedTile — Browse...
- Select archive file: — Browse...

Projects:

Options

- ☐ Search for nested projects
- ☐ Copy projects into workspace
- ☐ Close newly imported projects upon completion

< Back | Next > | Cancel | Finish

Figure 1: Select Project

on models semantics can be found on the PREESM web page[1].

3. **Open the PiSDF application description**: Within the folder `> Algo`, double click on the `.diagram` file: the PiSDF of the image processing algorithm will be displayed (Fig. 2). The application is composed of a pipeline of actors with data parallelism introduced by applying image filtering per tile of the input image. Note that much larger applications are available on `https://github.com/preesm/preesm-apps`.

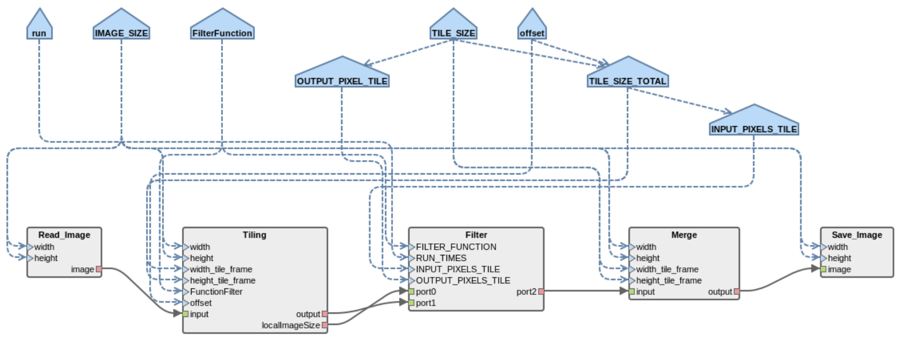


Figure 2: PiSDF representation of the image processing algorithm.

1.2 S-LAM Architecture Modeling

The specific platform to be used to test the application, in our case the Xilinx Zynq FPGA of the Xilinx Pynq board together with ARTICo^3 slots, is modeled using S-LAM abstract architecture modeling. This model serves as an input for the mapping and scheduling of the application onto the architecture.

1. **Open the S-LAM**: Within the `> Archi` folder, double click on the file `ARTICo3_4.slam` file: the

[1]`https://preesm.github.io/`

S-LAM model of the computing platform is displayed (Fig. 3). It is composed of a hardware core and four ARTICo^3 slots supporting dynamic partial hardware reconfiguration.

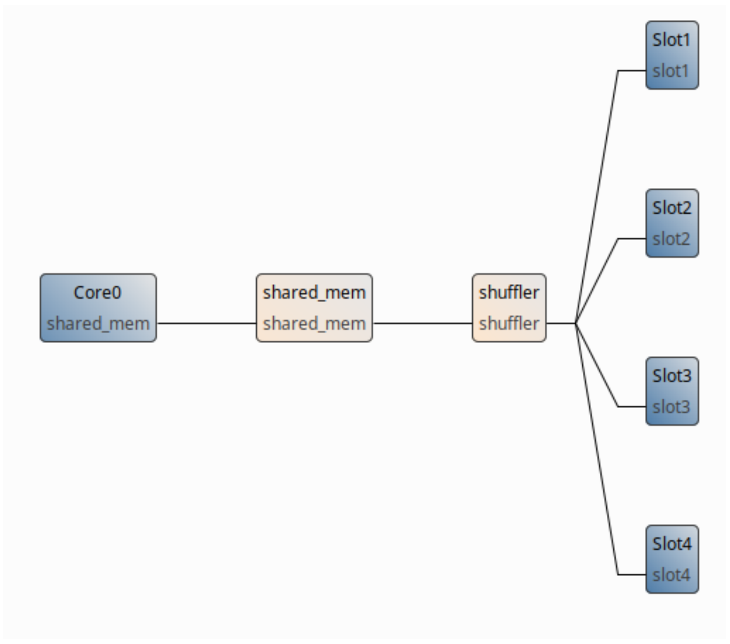


Figure 3: Architecture: one CPU and four ARTICo^3 slots.

In Fig. 3, the blue boxes are Processing Elements (PEs) and the pink boxes are communication facilities of our architecture. The board used for the tutorial is a Pynq equipped with a Zynq device. The device is composed by two ARM Cortex-A9 and a Xilinx FPGA. In the tutorial S-LAM model, one CPU core is modelled (Core0), as well as four ARTICo^3 slots (from Slot1 to Slot4). These numbers can be easily extended to improve system performance.

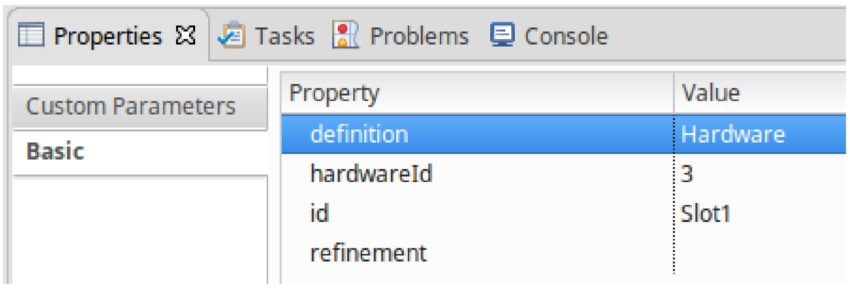
2. The S-LAM model can be graphically modified to report architecture modifications. Each PE is related to a code generation back-end within PREESM and libraries must be integrated that support message passing inter-PE communication.

PREESM gives the possibility to specify the nature of the PEs within the S-LAM model in order to describe heterogeneous systems. In this case, by choosing a PE and selecting the tab `> Properties > Basic` on the bottom of the screen (Fig. 4):

Within the `> definition`, it is possible to set the PE to:

- **ARM**: it generates software code ready to be compiled and executed upon an ARM CPU over a Linux support.

- **Hardware**: it generates code ready to be compiled to communicate with the corresponding ARTICo^3 hardware slot. The resulting system will offload processing into the FPGA logic side and make use of ARTICo^3 hardware acceleration.



Properties | Tasks | Problems | Console

Custom Parameters
Basic

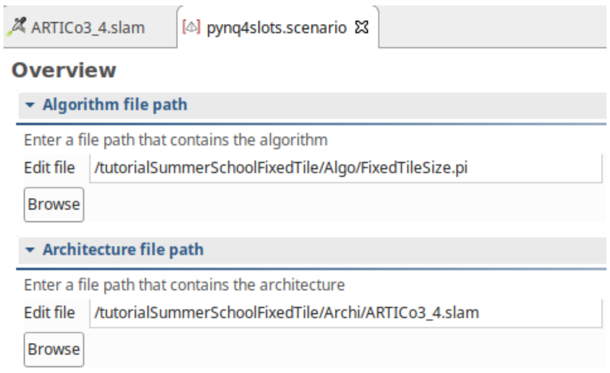
Property	Value
definition	Hardware
hardwareId	3
id	Slot1
refinement	

Figure 4: Properties tab on the bottom of the screen.

Other types of PEs are natively supported, such as x86 cores. New PE types can be created if related code generation and communication libraries are introduced. In the tutorial, an S-LAM model is proposed with one CPU and four ARTICo^3 slots (Fig. 3).

1.3 Scenario

The **Scenario** is the third input to the mapping and scheduling within PREESM. It aims at separating algorithm concerns from architecture concerns and contains information such as: optional affinity (constraints) for actors, forcing their execution on specific PEs, data size for the communication First-In-First-Out (FIFO) queues, timing of actor firings, etc. A detailed explanation of all the features available in the **Scenario** can be found online[2].
Let us open the scenario: in the Project Explorer tab, double click on:
`Scenario > pynq4slot.scenario`.
as shown in Figure 5.
Let us now set up the input files for the PiSDF and the S-LAM by clicking on `Browse` and by choosing:
- PiSDF : `FixedTileSize.pi`
- S-LAM : `ARTICo3_4.slam`



ARTICo3_4.slam | pynq4slots.scenario

Overview

Algorithm file path
Enter a file path that contains the algorithm
Edit file /tutorialSummerSchoolFixedTile/Algo/FixedTileSize.pi
Browse

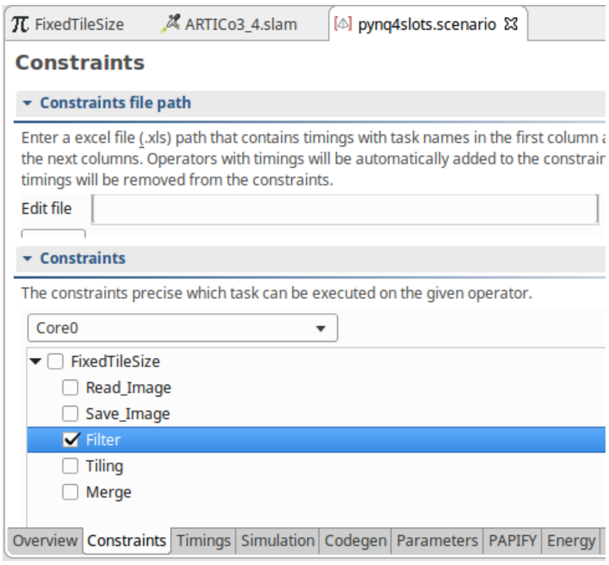
Architecture file path
Enter a file path that contains the architecture
Edit file /tutorialSummerSchoolFixedTile/Archi/ARTICo3_4.slam
Browse

Figure 5: Scenario overview.

Details of the tab `> PAPIFY` are going to be analyzed in a subsequent section of the tutorial. Let us

[2]`https://preesm.github.io/tutos/`

focus the attention on the tab `> Constraints` as highlighted in Fig. 6:



FixedTileSize | ARTICo3_4.slam | pynq4slots.scenario

Constraints

Constraints file path

Enter a excel file (.xls) path that contains timings with task names in the first column a
the next columns. Operators with timings will be automatically added to the constrain
timings will be removed from the constraints.

Edit file

Constraints

The constraints precise which task can be executed on the given operator.

Core0

- [ ] FixedTileSize
  - [ ] Read_Image
  - [ ] Save_Image
  - [x] Filter
  - [ ] Tiling
  - [ ] Merge

Figure 6: Scenario in PREESM.

In this tab, we can assign a specific actor (or a set of actors) firings to a specific PE (or a set of specific PEs). Keep in mind that you can execute on the FPGA **only** actors which behaviour has been previously synthesized and implemented using Vivado. Indeed, this part is external from PREESM. Conversely, actors can be moved to software, provided that C code for internal actor behavior is provided.
Having designed only the hardware accelerator for the *Filter* actor, let us set the *Constraints* as follow:
- `Core0`: select all actors executions
- `Slot1`: select just *Filter*
- `Slot2`: select just *Filter*
- `Slot3`: select just *Filter*
- `Slot4`: select just *Filter*

Using such configuration, some instances of the filter can also be executed in software, based on automated resource mapping decisions.

Computing the Gantt chart of execution
- Within PREESM, right click on *Codegen.workflow* available in *Workflows* folder
- Click on `Preesm > Run Workflow`
- Select *pynq4slots.scenario* from *tutorialSummerSchoolFixedTile/Scenarios* folder

The Gantt chart displays, as a prediction of future execution times and synchronization.

1.4 Early Design Space Exploration

It is possible to change the parameter values on the PiSDF, the S-LAM and/or the Scenario and execute the workflow many times and explore design. As a rule of thumb, PREESM workflow executions remain under a few minutes for a number of actors firings limited to a thousand. After the execution of the generated code on the target device, the consequence of the changing can be observed and collected, thus providing Design Space Exploration (DSE) possibilities.

As an example, in the scenario, constraints can be put to manually invalidate hardware accelerators.

These design space exploration steps are labelled as 0 to 4 in Figure 7. The next presented steps will build a more advanced profiling-based DSE, exploiting hardware system generation and performance profiling.

2 System Performance Monitoring Setup

As an overlay of the PAPI (Performance Application Programming Interface) library[3], PAPIFY makes it possible to observe different events through different Performance Monitoring Counters (PMCs). The PAPIFY configuration steps consist in choosing the events of interest, among the ones offered by the platform. The ARTICo^3 and MDC hardware support offer different PMCs and the software stack to observe their related events.

Monitoring Configuration

The configuration of the system monitoring is done in the PAPIFY tab of PREESM, in the scenario file. The resulting configuration is shown in Figure 8.

1. Import monitoring info if it is not already done
   - Click on *Browse* button
   - Select *PAPI_info.xml* available in *tutorialSummerSchoolFixedTile/Code*
2. In PAPIFY PE configuration, associate PAPI components with PE types
   - *perf_event*↔*x86*
   - *artico3*↔*Hardware*
3. In PAPIFY actor configuration, associate PAPI events with actors
   - Select *timing* event for every actor
   - Select *PAPI_L1_DCM* event for every actor
   - Select *artico3:::MDC_CLOCK_CYCLE* event for every actor

The monitoring configuration to be used will be selected automatically during the application execution.

3 Creating Hardware Accelerators with DPR and CGR

The PiSDF graph designed in PREESM is a dataflow representation of the algorithm at a coarse granularity: each actor firing thus comprises a large set of operations, processing large data. Conversely, hardware offloading requires a fine-grain dataflow representation of the algorithm to be offloaded to

[3]https://icl.utk.edu/papi/

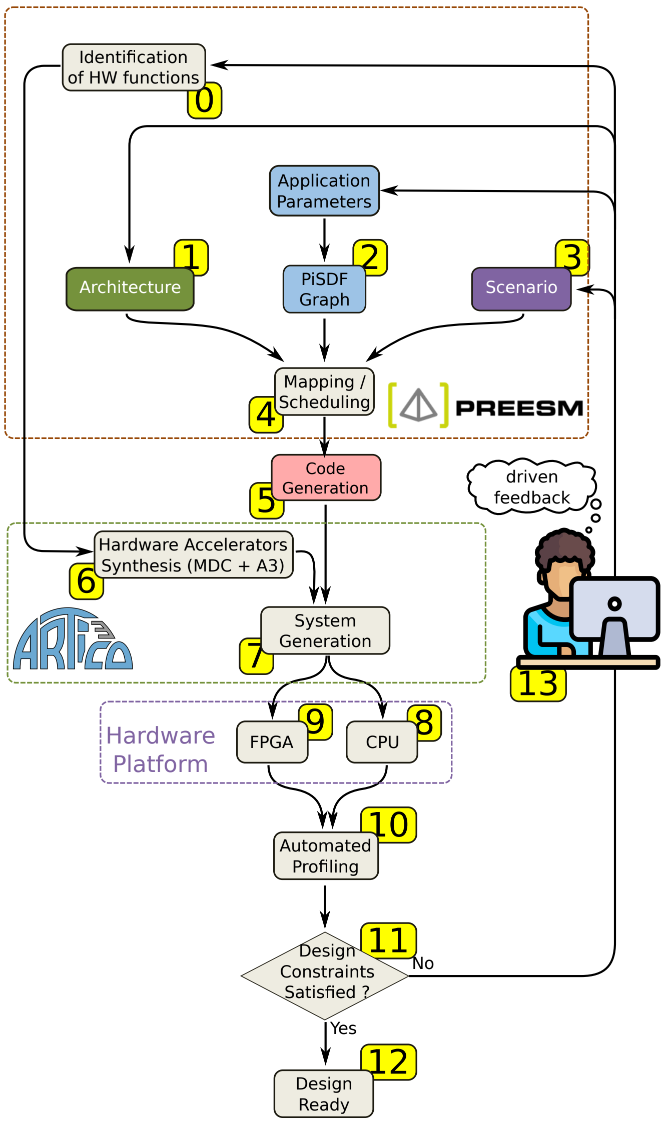
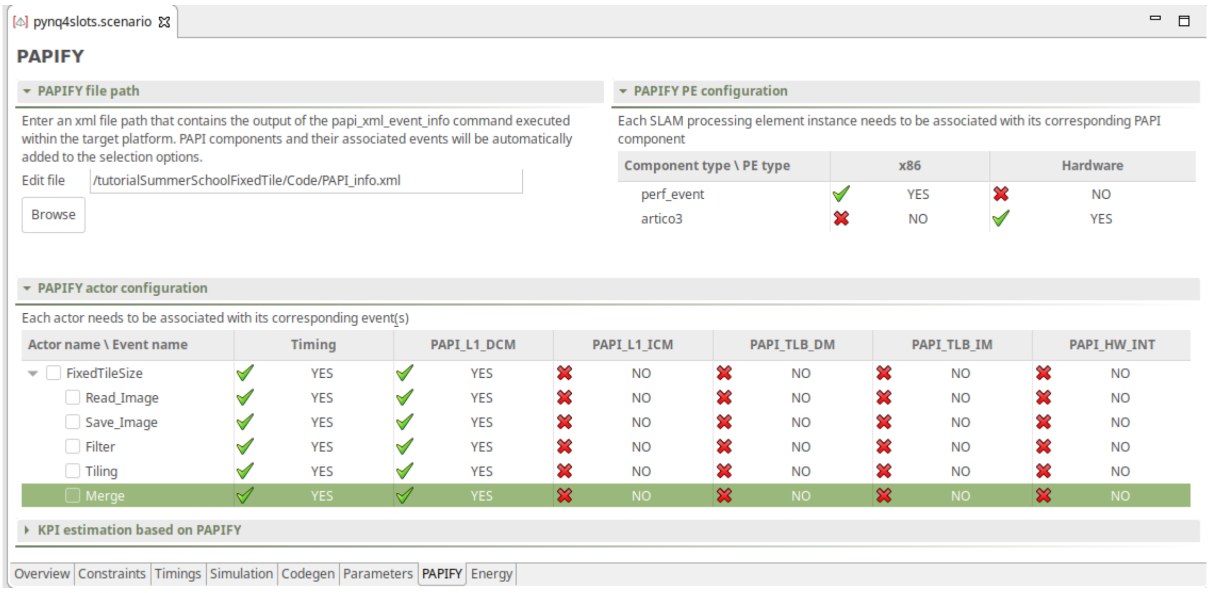


Figure 7: Flowchart of Profiling-based DSE: three different inputs are provided to the Mapping/Scheduling algorithm of PREESM. The architecture description of the targeted platform (S-LAM), the PiSDF description of the application and the scenario containing the constraints linking the two. After the Mapping/Scheduling part, PREESM generates compilable code within few seconds. Then, the ARTICo^3 toolchain performs system generation taking as inputs the MDC verilog code of the accelerators. An instrumented run of the application is performed with automated profiling and tests. If the design constraints are satisfied then the DSE is finished, otherwise the provided feedbacks are used to modify parameters in either of the inputs.The figure describes a DSE as presented here.

HW. This is supported in MDC through CAPH and CAL languages. Transformations from coarse-grain dataflow to hardware has been considered in CERBERO but kept out of this tutorial that puts focus on the highest possible degree of system adaptivity.

The hardware datapaths and HDL component libraries for supporting the demonstration application have been created using the CAPH[4] dataflow high level hardware design tool. They implement the firing

[4]`http://caph.univ-bpclermont.fr`


pynq4slots.scenario

PAPIFY

PAPIFY file path

Enter an xml file path that contains the output of the papi_xml_event_info command executed within the target platform. PAPI components and their associated events will be automatically added to the selection options.

Edit file /tutorialSummerSchoolFixedTile/Code/PAPI_info.xml

Browse

PAPIFY PE configuration

Each SLAM processing element instance needs to be associated with its corresponding PAPI component

Component type \ PE type	x86	Hardware
perf_event	✓ YES	✗ NO
artico3	✗ NO	✓ YES

PAPIFY actor configuration

Each actor needs to be associated with its corresponding event(s)

Actor name \ Event name	Timing	PAPI_L1_DCM	PAPI_L1_ICM	PAPI_TLB_DM	PAPI_TLB_IM	PAPI_HW_INT
FixedTileSize	✓ YES	✓ YES	✗ NO	✗ NO	✗ NO	✗ NO
Read_Image	✓ YES	✓ YES	✗ NO	✗ NO	✗ NO	✗ NO
Save_Image	✓ YES	✓ YES	✗ NO	✗ NO	✗ NO	✗ NO
Filter	✓ YES	✓ YES	✗ NO	✗ NO	✗ NO	✗ NO
Tiling	✓ YES	✓ YES	✗ NO	✗ NO	✗ NO	✗ NO
Merge	✓ YES	✓ YES	✗ NO	✗ NO	✗ NO	✗ NO

KPI estimation based on PAPIFY

Overview | Constraints | Timings | Simulation | Codegen | Parameters | PAPIFY | Energy

Figure 8: PAPIFY configuration in the scenario tab of PREESM.

of the hardware ported actors.

Starting from the two dataflow descriptions of an image filter: a Sobel filtering variant and a Roberts filtering variant of the application, the MDC tool merges them in a multi-dataflow one with automated coarse-grain reconfiguration. The resulting application can switch dynamically between the two alternatives, according to useful information, i.e. performance requirements. Such a coarse grain reconfiguration support extends to complex datapaths with thousands or more parameters combinations.

In the advanced hardware setup underlying this tutorial, ARTICo^3 dynamically reconfigurable hardware kernels themselves embed Coarse-Grain Reconfigurable (CGR) computing logic generated by MDC. The rationale behind combining CGR with MDC and Dynamic Partial Reconfiguration (DPR) with ARTICo^3 is that CGR offers clock-cycle-time reconfiguration between two datapaths but hardware reuse is partial, while DPR offers millisecond-scale reconfiguration with total hardware reuse. In the tutorial application, DPR manages different tiles of the input image, thus exploiting data parallelism, while CGR switches between Sobel and Roberts filters.

ARTICo^3 Kernel Generation with MDC-powered CGR inside.
To use the coarse-grain reconfigurable code generated by MDC within an ARTICo^3 compliant kernel, one needs to use the following procedure. MDC is a datapath merging tool compatible with several code generations and technologies, hence requiring configuration.

1. Launch MDC executable, placed in folder `/home/embedded/Desktop/MDC_CPS/MDC_tool/eclipse`.

2. Check the project.
   If not present in the workspace, import `Tutorial_EdgeDetection` project:

`> File > Import... > General > Existing Project into Workspace`
Browse to `/home/embedded/Desktop/MDC_CPS/MDC_input/Tutorial_EdgeDetection`, then:
`> OK > Finish`
Here we can browse and check (double click) the input dataflows to be used: `> Tutorial_EdgeDetection > src > edgeDetection > roberts.xdf` and `> Tutorial_EdgeDetection > src > edgeDetection > sobel.xdf`.

3. Open and check the run configuration as follows:
   `> Run > Run configurations...`
   then Select "Tutorial_EdgeDetection" under Orcc compilation on the left menu.

4. Check the following compilation settings, as shown in Figure 10.
   Name: name for the configuration (for instance "Tutorial_EdgeDetection")
   Project: "Tutorial_EdgeDetection"
   Backend:
   - Select a backend: MDC
   - Output Folder: `/home/embedded/Desktop/artico3/demos/mdc_monitors`
   Options:
   - "List of Networks to be Compiled and Merged" ticked
   - Number of Networks: 2
   - XDF List of Files: "edgeDetection.roberts, edgeDetection.sobel"
   - Merging Algorithm: EMPIRIC
   - "Generate RVC-CAL multi-dataflow" ticked with "DUMMY" as "CAL type"
   - "Generate HDL multi-dataflow" ticked
   - Protocol file: `MDC_CPS/MDC_input/protocol/protocol_CAPH.xml`
   - HDL component library: `MDC_CPS/MDC_input/HDL_compLib` (this folder must contain all the necessary HDL files)
   - "System Generation" ticked
   - "ARTICo^3 Backend" ticked (see Fig. 9)
   - "Enable Monitoring" ticked with the last three monitors selected

5. Select Apply and choose Run.
   - Generated files within MDC workspace will appear as soon as the `> Tutorial_EdgeDetection > src` folder is refreshed. A new sub-folder with the date and hour of the run is created, containing the combined reconfigurable dataflow (`multi_dataflow.xdf`) that can checked with a double click.
   - Generated files outside MDC workspace are located in the specified output folder. In particular the `src/` sub-folder includes all the necessary files to create the PAPIFY-monitored and ARTICo^3-compliant CGR accelerator, while `mdc-papi_info.xml` describes the PAPI configurations of the MDC accelerator.

System Implementation
- **input**: HDL files generated by the MDC framework

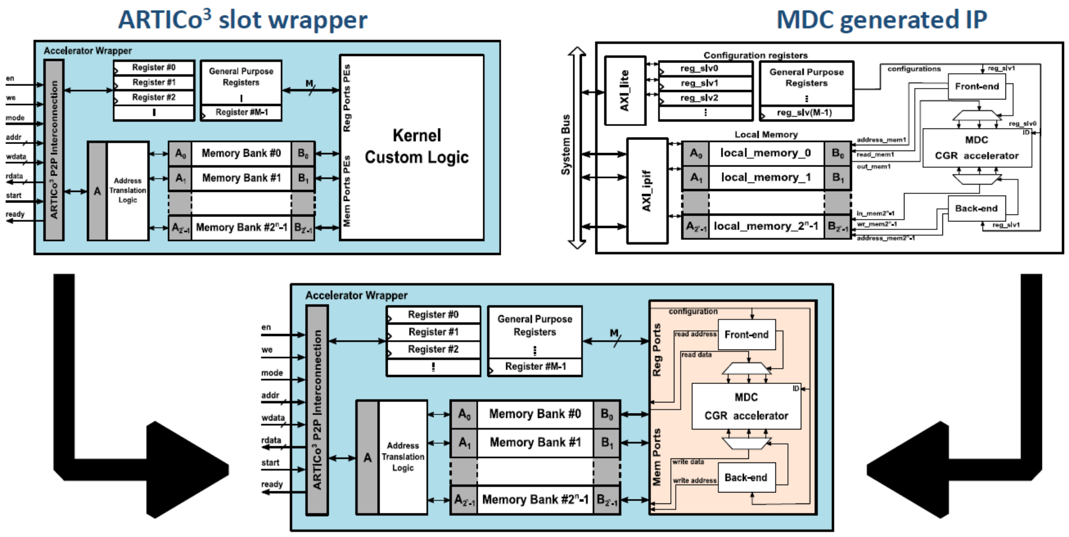


Figure 9: MDC accelerator in a ARTICo^3-slot wrapper. By choosing the option "ARTICo^3 Backend", the MDC design suite is able to generate an ARTICo^3-compliant accelerator that makes possible the use of the new CGR accelerator within the ARTICo^3 hardware structure on the FPGA.

- **output**: bitstreams of the synthetized system

Let us run the synthesis and the bitstream generation by using the ARTICo^3 toolchain and a configuration file `build.cfg`.

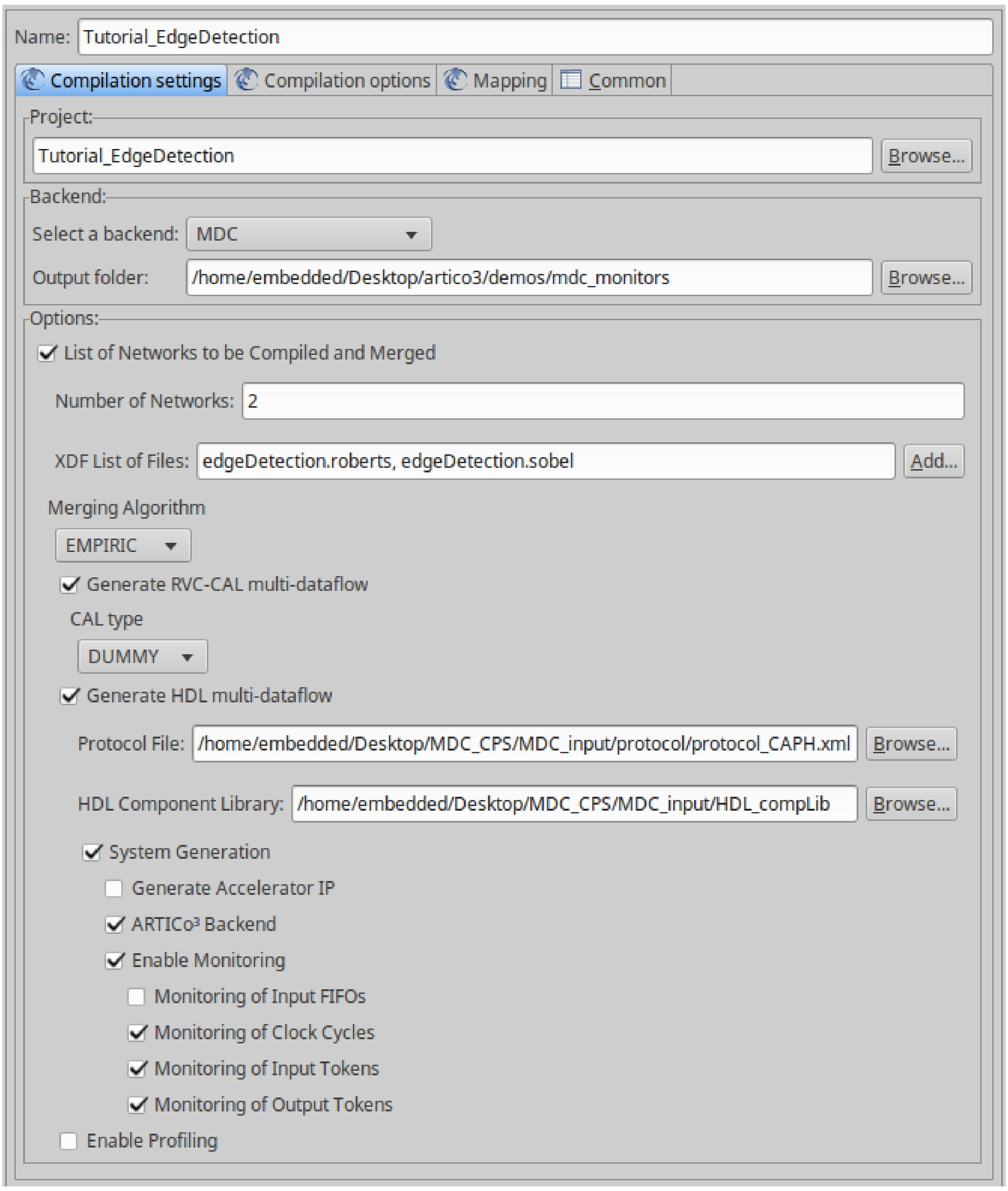
This file can be created *ex novo* with the option shown in the Fig.11, but there is one located in the output folder `/home/embedded/Desktop/artico3/demos/mdc_monitors`, for tuning the option depending on one's specific needs.

1. Open a terminal in the output folder (`/home/embedded/Desktop/artico3/demos/mdc_monitors`) in which next commands will be launched.

2. Set up the ARTICo^3 enviroment by running:
```
$ source /home/embedded/Desktop/artico3/tools/setting.sh
```

3. Generate the RTL system:
```
$ a3dk
$ export_hw
```

4. Build the system (we are going to SKIP THIS STEP during the tutorial for timing reasons):
```
$ build_hw
```


Name: Tutorial_EdgeDetection

Project: Tutorial_EdgeDetection

Backend:
Select a backend: MDC
Output folder: /home/embedded/Desktop/artico3/demos/mdc_monitors

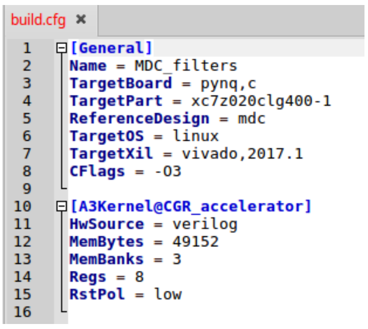
Options:
✓ List of Networks to be Compiled and Merged
Number of Networks: 2
XDF List of Files: edgeDetection.roberts, edgeDetection.sobel
Merging Algorithm: EMPIRIC
✓ Generate RVC-CAL multi-dataflow
CAL type: DUMMY
✓ Generate HDL multi-dataflow
Protocol File: /home/embedded/Desktop/MDC_CPS/MDC_input/protocol/protocol_CAPH.xml
HDL Component Library: /home/embedded/Desktop/MDC_CPS/MDC_input/HDL_compLib
✓ System Generation
☐ Generate Accelerator IP
✓ ARTICo³ Backend
✓ Enable Monitoring
☐ Monitoring of Input FIFOs
✓ Monitoring of Clock Cycles
✓ Monitoring of Input Tokens
✓ Monitoring of Output Tokens
☐ Enable Profiling

Figure 10: Compilation settings as in the MDC GUI

The bitstream is created (the blue part highlighted in Fig.12) in the folder `.../mdc_output/build.hw/bin/`. At this point, bitstreams should be moved upon the target device OS: ARTICo^3 runtime functions will be in charge of managing the FPGA reconfiguration. All the necessary steps are detailed on the ARTICo^3 website[5].

Optional:

[5]`https://des-cei.github.io/tools/artico3/tutorials/setup#execute-on-target-platform`



```
[General]
Name = MDC_filters
TargetBoard = pynq,c
TargetPart = xc7z020clg400-1
ReferenceDesign = mdc
TargetOS = linux
TargetXil = vivado,2017.1
CFlags = -O3

[A3Kernel@CGR_accelerator]
HwSource = verilog
MemBytes = 49152
MemBanks = 3
Regs = 8
RstPol = low

```

Figure 11: Configuration File Options

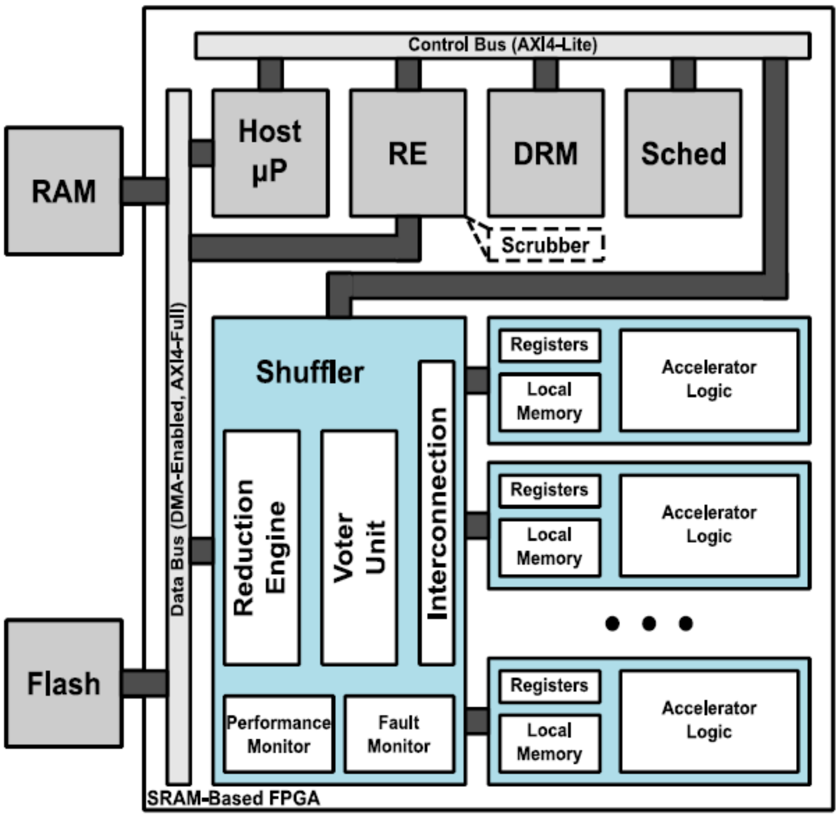


Figure 12: ARTICo^3 architecture structure on the FPGA. The ARTICo^3 framework is capable of automatically generating the whole system making the FPGA slots available and manageable through High-Level sotfware APIs. In the proposed tutorial, the Accelerator Logic is created by using the MDC design suite.

In order to connect the PYNQ board to a PC, two options are available for the tutorial:

1. Using a serial connection using the `Port USB1` with a `Baud Rate of 115200`.[6]

2. Using the Ethernet port of the PYNQ board connected to your Local Area Network (LAN)[7] (or

[6]**Teraterm** and **Putty** are two options among many for connecting to the serial port.
[7]`https://www.wikihow.com/Create-a-Local-Area-Network-(LAN)`

directly to your PC with a cable). The Pynq board can be set up with a static IP:

```
$ ifconfig eth0 192.168.0.xxx
```

To have access to the PYNQ OS command line, please use the `ssh` protocol:

```
$ ssh linaro@192.168.0.xxx
```

If you want to have full access to the PYNQ Filesystem, a convenient option is to use the Ubuntu's File Manager and the `sftp` protocol as shown in Figure 13.

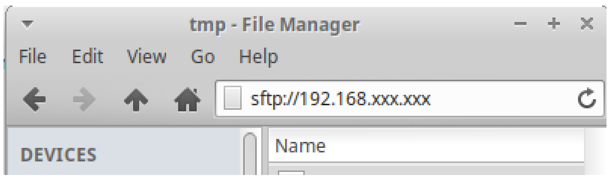


Figure 13: Ubuntu's File Manager to navigate the Pynq Filesystem

4 System Generation and Execution, and Performance Profiling

Code Generation under PREESM
- Within PREESM, right click on *Codegen.workflow* available in *Workflows* folder
- Click on `Preesm > Run Workflow`
- Select *pynq4slots.scenario* from *tutorialSummerSchoolFixedTile/Scenarios* folder

Compile and Execute on Pynq Board
- Copy on the Pynq board the complete *tutorialSummerSchoolFixedTile/generated/Code* folder
- Compilation and execution set up: `source compile_and_setup.sh`
- Go to execution directory: `cd /home/linaro/mdc_summer_school/bin`
- Execute the application: `./summerSchoolFixedTile`

Profiling analysis with Papify-Viewer
- Open the folder in which Papify-Viewer is installed:
`cd /home/Desktop/papify/PapifyViewer`
- Launch PAPIFY-VIEWER tool: `python PapifyViewerDynamic.py`
- In *Choose Folder* option, select the *papify-output* folder

- If you have a PYNQ board the folder is `/home/linaro/mdc_summer_school/bin/papify-output`
- If you do not have a PYNQ, there is a *papify-output* folder in `/home/Desktop/papify-output`

- Select *Cores fixed* option to visualize the application timing execution. The result should be equivalent to the one shown in Figure 14.

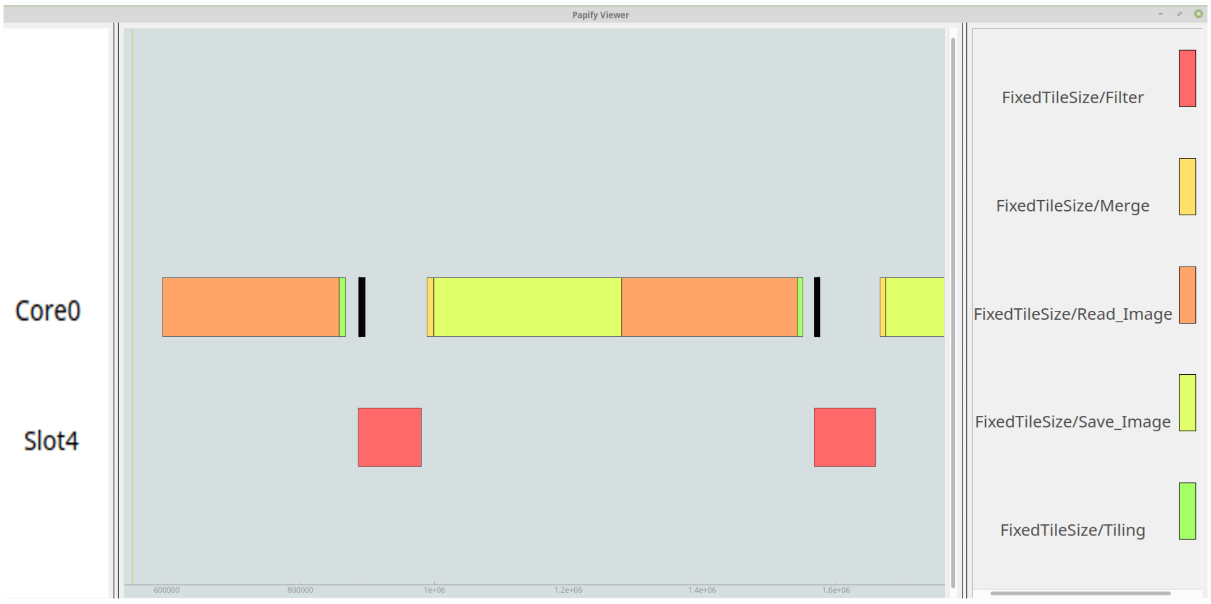


Figure 14: PAPIFY-viewer resulting Gantt charts.