Hands On: Design Flow for Heterogeneous Embedded Computing Infrastructures

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

1 Hardware Accelerators Creation

As in the tutorial of the last year[1](in which is based this one), the use case is an edge detection application involving two different algorithms: Sobel and Roberts. The single networks and HDL component libraries have been created using CAPH[2] tool. Starting from their dataflow descriptions, MDC is capable to merge them in a multi-dataflow one (for more details, see the mentioned tutorial).

ARTICo^3 Kernel Generation

To use the coarse-grain reconfigurable computing core generated by MDC it is necessary to generate an ARTICo^3 compliant kernel.

1. Launch MDC executable, placed in folder `/home/embedded/Desktop/MDC_CPS/MDC_tool/eclipse` and confirm the workspace with OK.

2. Import Project:

`> File > Import... > General > Existing Project into Workspace`

Browse to `/home/embedded/Desktop/MDC_CPS/MDC_input/Tutorial_EdgeDetection`, then:

`> OK > Finish`

3. Create a configuration window as follows:

`> Run > Run configurations...`

then right click on Orcc compilation and then select New.

4. Fill in the following compilation settings, as shown in Figure 1.

Name: choose a name for the configuration (for instance "Tutorial_EdgeDetection")

Project: select "Tutorial_EdgeDetection"

Backend:

- Select a backend: MDC
- Output Folder: `/home/embedded/Desktop/artico3/demos/mdc_monitors`

Options:

- Tick "List of Networks to be Compiled and Merged"
- Number of Networks: 2

[1] `http://www.cpsschool.eu/wp-content/uploads/2018/09/Tutorial_Multi-Grain-Reconfiguration-1.pdf`

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

- XDF List of Files: select the two input dataflow networks: "edgeDetection.roberts" and "edgeDetection.sobel"
- Merging Algorithm: EMPIRIC
- Tick "Generate RVC-CAL multi-dataflow" with "DUMMY" as option
- Select "Generate HDL multi-dataflow"
- 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)
- Tick on "System Generation"
- Tick on "ARTICo^3 Backend"
- Tick on "Enable Monitoring" (selecting the last three monitors)

5. Select Apply and choose Run.

Output folder
Output folder contains:

- `src/`: includes all the necessary files to create the PAPIFY-monitored and ARTICo^3-compliant CGR accelerator
- `mdc-papi_info.xml`: describes the PAPI configurations of the MDC accelerator.

System Implementation
- **input**: HDL files generated by MDC framework
- **output**: bitstreams of the synthetized system

Let's 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.2, but there is one located in the output folder `/home/embedded/Desktop/artico3/demos/mdc_monitors`, for tuning the option depending on your own 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):
```
$ build_hw
```
The bitstream will be created in the folder `.../mdc_output/build.hw/bin/`. At this point, the 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[3].
Optional:

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

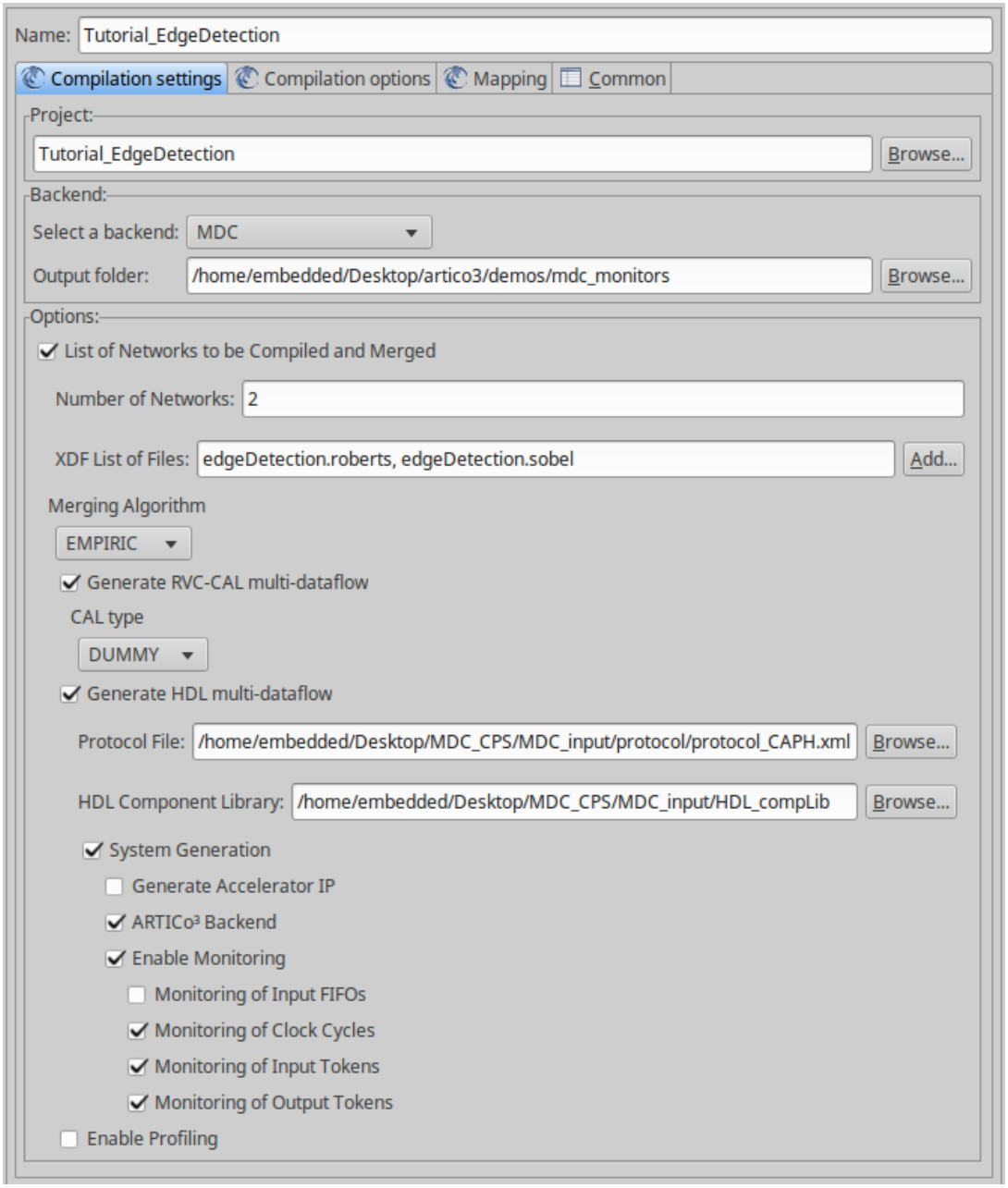

Figure 1: Compilation settings as in the MDC GUI

In order to connect the PYNQ board to your laptop, two options are available for the tutorial:

1. Using a serial connection using the `Port USB1` with a `Boud Rate of 115200`.[4]
2. Using the Ethernet port of the PYNQ board connected to your Local Area Network (LAN)[5] (or directly to your laptop with a cable). The Pynq board can be set up with a static IP:

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

[4] **Teraterm** and **Putty** are two options.

[5] `https://www.wikihow.com/Create-a-Local-Area-Network-(LAN)`

build.cfg

```
[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 2: Configuration File Options

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, the best option is to use the Ubuntu's File Manager and the `sftp` protocol as shown in Figure 3.

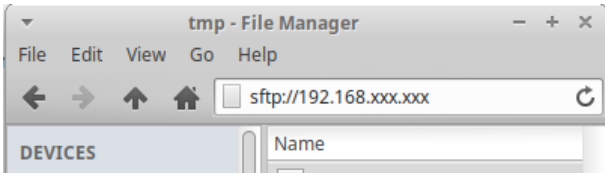

Figure 3: Ubuntu's File Manager to navigate the Pynq Filesystem

2 Hardware/Software Code-Generation Setup for Design Space Exploration

- **inputs**: bitstreams
- **outputs**: code ready to be compiled and executed

2.1 Parameterized and Interfaced Synchronous DataFlow (PiSDF)

The algorithm of the application is one of the main inputs of the method and needs to be specified. Besides, being the algorithm description compliant with a Dataflow Model of Computation (MoC), the method exploits its intrinsic expressiveness of parallelism. For this purpose a PiSDF MoC is utilized: a graph that connects *Actors* and *Parameters* through *FIFO* and *Parameter dependency link*.

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`:

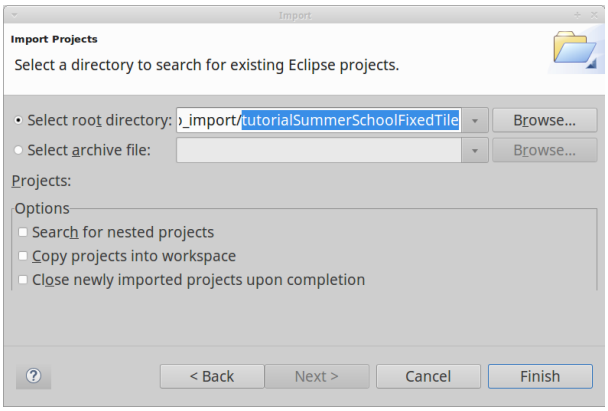

Figure 4: Select Project

   Within the folder `Algo`, the algorithm is described by making use of the PiSDF. Within the folder `Archi`, the hardware architecture is described by making use of the S-LAM. Other information can be found in the PREESM web page[6].

3. **Open the PiSDF**: Within the folder `> Algo`, double click on the file `.diagram` file: the PiSDF of the image processing algorithm will be dispayed (Fig. 5).

2.2 S-LAM

The specific device to be used to test the application needs to be described and serves as an input for the mapping and scheduling of the application onto the architecture. Because of this, the SLAM was used as an abstract platform model.

1. **Open the S-LAM**: Within the folder `> Archi`, double click on the file `ARTICo3_4.slam` file: the PiSDF of the image processing algorithm will be displayed (Fig. 6).

   In the case reported in the above figure, the blue boxes are the Processing Elements (PEs) and the pink boxes are the memories 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.

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

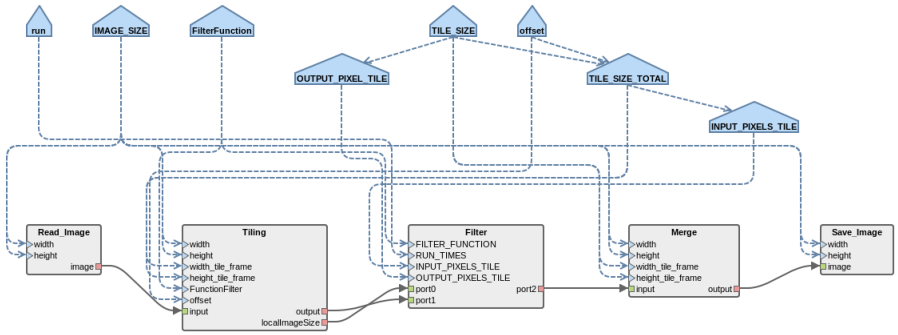

Figure 5: PiSDF of the image processing algorithm.

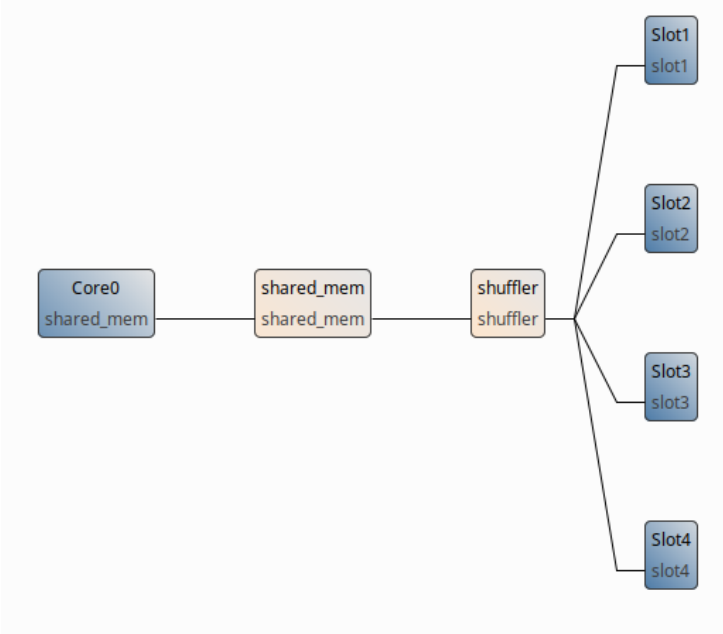

Figure 6: Architecture: one CPU and four ARTICo^3 slots.

In the SLAM, one CPU core is modelled (the Core0) and four ARTICo^3 slots (from Slot1 to Slot4).

2. The SLAM can be modified by using the palette on the right side of the screen.

PREESM gives the possibility to specify the nature of the PE within the SLAM in order to describe heterogeneous system. In this case, by choosing a PE and selecting the tab `> Properties > Basic` on the bottom of the screen (Fig. 7):

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

- **ARM**: it generates code ready to be compiled and executed upon a CPU.

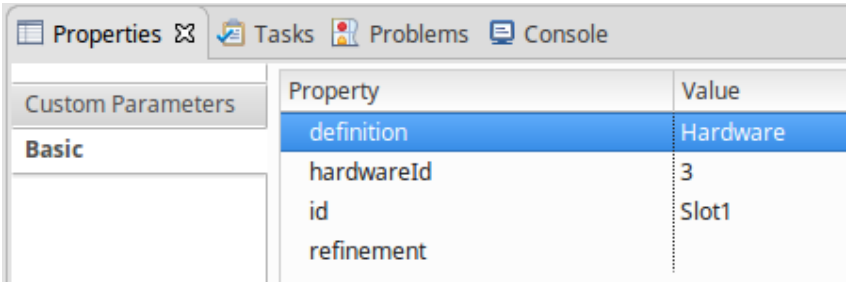

Figure 7: Properties tab on the bottom of the screen.

- **Hardware**: it generates code ready to compiled. It offloads some "processing" into the FPGA side (by making use of Hardware Accelerators).

In the tutorial, a SLAM is proposed with one CPU and four ARTICo^3 slots (Fig. 6).

2.3 Scenario

The **Scenario** is the last input for the mapping and scheduling within PREESM, where additional information is provided: optional affinity for actors forcing their execution on specific processing elements, data size of the FIFOs tokens, timings of the actors executions, etc. A detailed explanation of all the feature available in the **Scenario** can be found online[7].
Let's open the scenario: in the Project Explorer tab, double click on
`Scenario > pynq4slot.scenario`.
as shown in Figure 8.
Let's now set up the input files for the PiSDF and the SLAM by clicking on `Browse` and by choosing:
- PiSDF : `FixedTileSize.pi`
- S-LAM : `ARTICo3_4.slam`

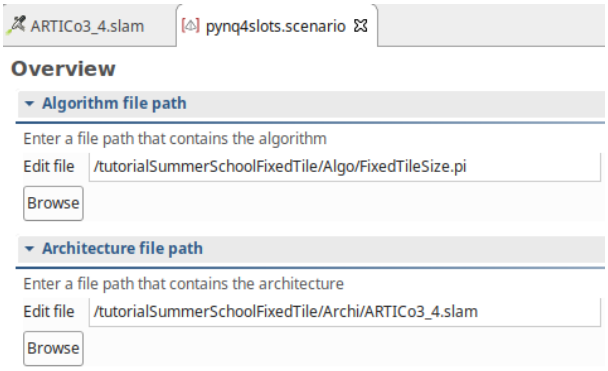

Figure 8: Scenario overview.

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

Some details of the tab > `PAPIFY` are going to be analyzed in the last part of the tutorial. Let's focus now the attention on the tab > `Constraints` as highlight in Fig. 9:

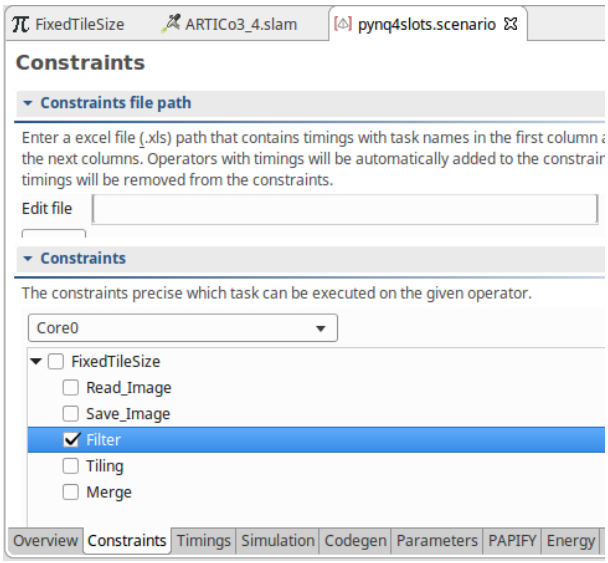

Figure 9: Scenario in PREESM.

In this tab, we can assign a specific actor (or a set of actors) execution to a specific PE (or a set of specific PEs). Keep in mind that you can execute on the FPGA **only** actor which behaviour has been previously synthesized using Vivado. If you assign any other actor to the FPGA side, the code generation will end with no error but, during the execution, the software will not find the right bitstream to be written in the Configuration Memory.
Having designed only the hardware accelerator for the *Filter* actor, let's set the *Constraints* as follow:
- `Core0`: enable all actors execution
- `Slot1`: select just *Filter*
- `Slot2`: select just *Filter*
- `Slot3`: select just *Filter*
- `Slot4`: select just *Filter*

2.4 Design Space Exploration

It is possible to change the parameter values on the PiSDF, the SLAM and/or the Scenario and execute the workflow as many times you want. After the execution of the generated code on the target device, the consequence of the changing can be observed and collected, thus allowing a Design Space Exploration (DSE).

3 Monitoring, Code Generation and Profiling

Monitoring Configuration

The configuration is done in PAPIFY tab, from scenario file. The resulting configuration is shown in Figure 10

1. Import monitoring info
   - 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* and *PAPI_L1_DCM* event for every actor
   - Select *artico3:::MDC_CLOCK_CYCLE* event for actor *Filter*

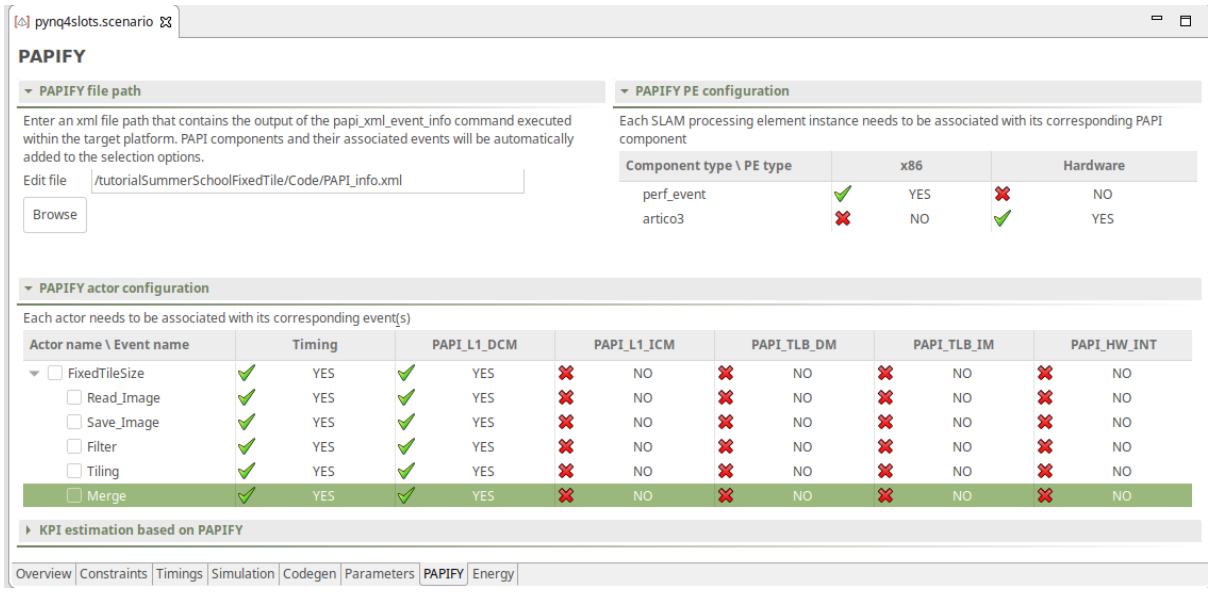

Figure 10: PAPIFY configuration.

Code Generation

- 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

- On your laptop, after installing PAPIFY-VIEWER tool[8], open its containing folder:
`cd /home/embedded/Desktop/papify/PapifyViewer`
- Launch PAPIFY-VIEWER tool: `python PapifyViewerDynamic.py`
- In *Choose Folder* option, select the *papify-output* folder (it can be found in the same folder from where the application is executed. In our case `cd /home/linaro/mdc_summer_school/bin/papify-ouput`)
- Select *Cores fixed* option to visualize the application timing execution. The result should be equivalent to the one shown in Figure 11.

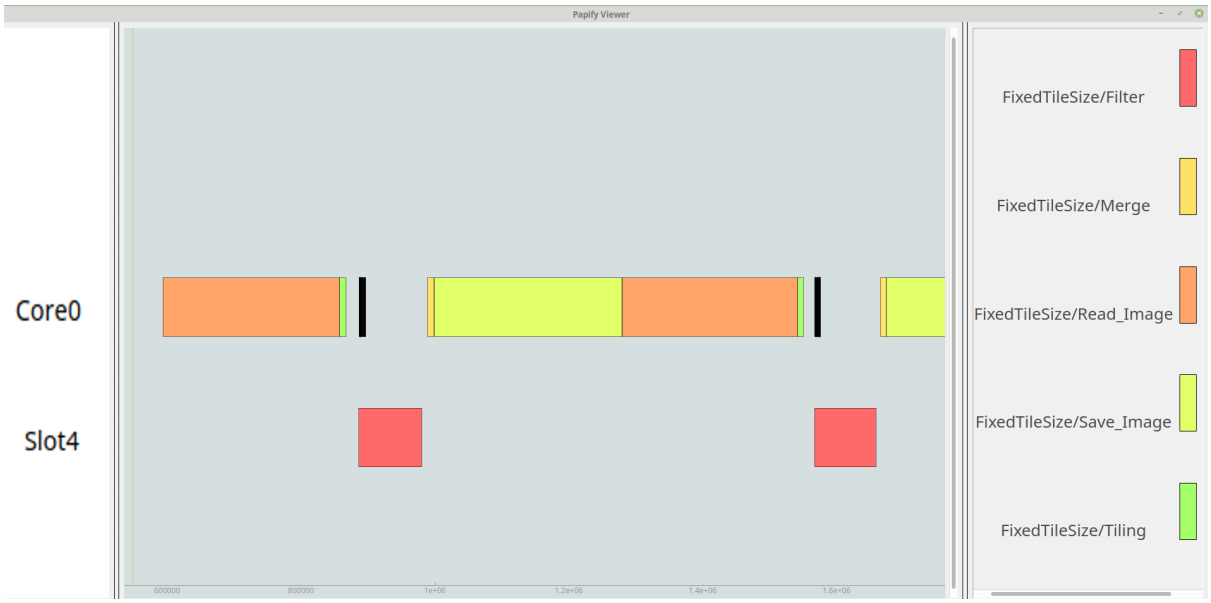

Figure 11: PAPIFY configuration.

[8]How to install PAPIFY-VIEWER: `https://gitlab.citsem.upm.es/papify/papify/tree/master/PapifyViewer`