

Multi-Grain Reconfiguration for Advanced Adaptivity in Cyber-Physical Systems

This tutorial shows how to generate a multi-grain reconfigurable system.
The used tools are:

1. the *Multi-Dataflow composer (MDC)* tool (<http://sites.unica.it/rpct/>);
2. the *ARTICo³* framework (more info [here](#))

Adopted Use-Case

Edge detection application involving two different algorithms: Sobel and Roberts. Both algorithms consider the convolution of two kernels with a grayscale image (see Fig. 1), in order to highlight the high-frequency variations due to the horizontal and vertical edges.

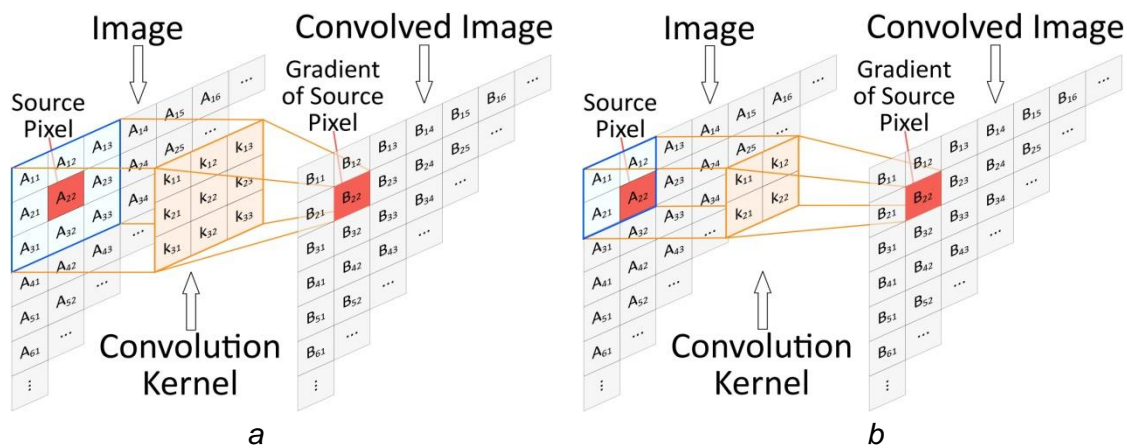
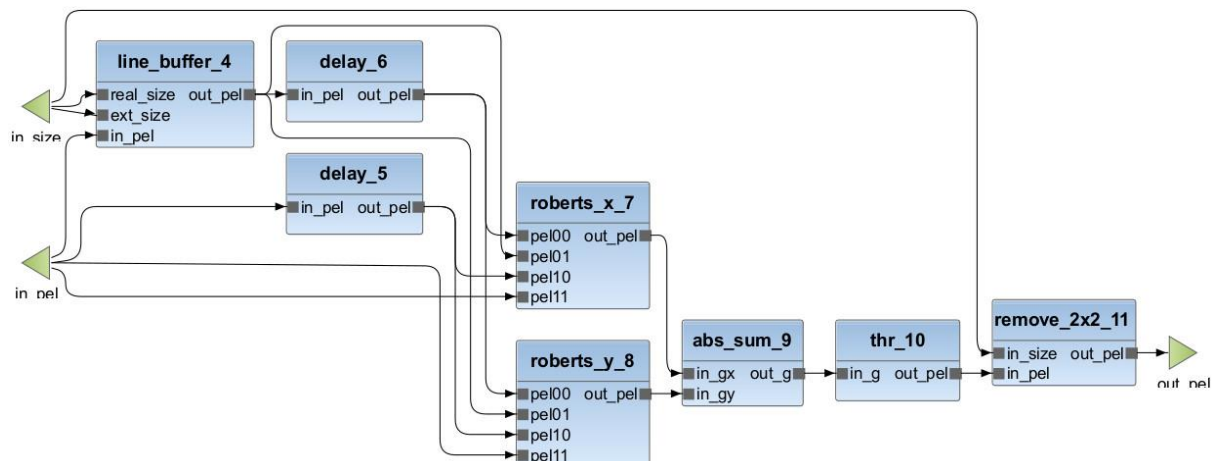
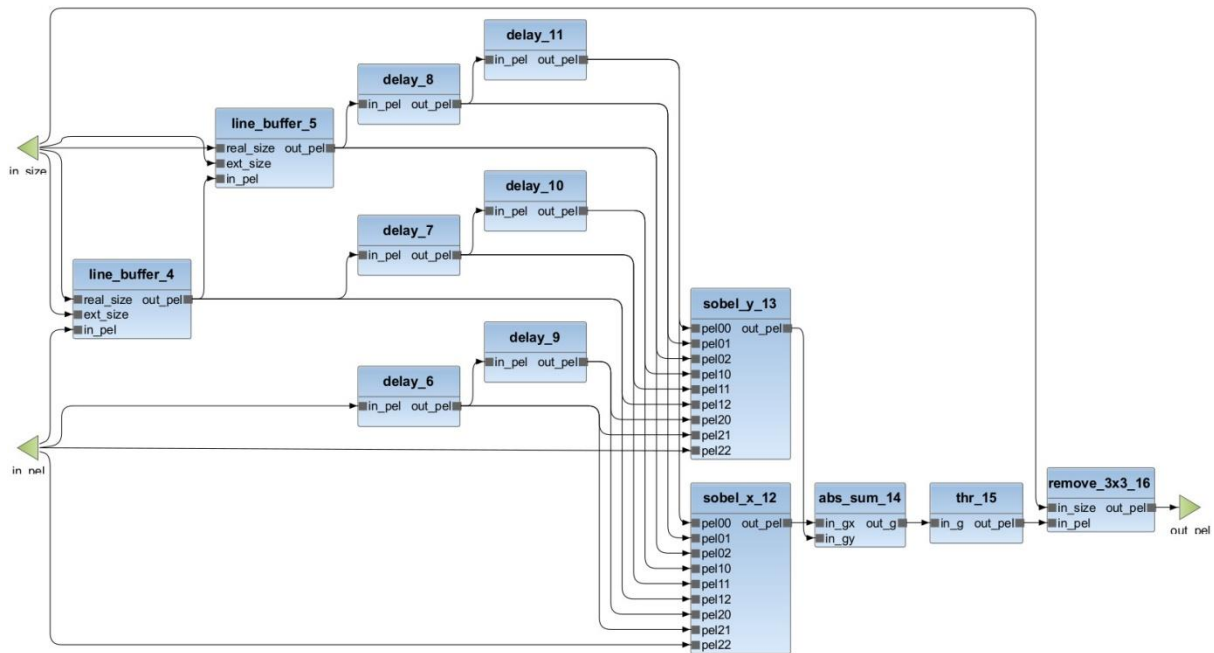


Figure 1: Computation of the convolution of kernels (x and y) of the Sobel (a) and Roberts (b) operators with an input image (A).

Figure 2 illustrates the single networks for Sobel (a) and Roberts (b) algorithms.



a



b

Figure 2: Dataflow description of the single networks for the Sobel (a) and Roberts (b) operators and for the multi-dataflow reconfigurable system (c).

The single networks and HDL component libraries have been created using CAPH tool (<http://caph.univ-bpclermont.fr>). CAPH is a framework for the specification, simulation and implementation of stream processing applications based on a dynamic Dataflow MoC ([ref](#)).

Dataflow-to-Hardware

Launch **MDC executable**, placed in folder MDC_CPS.

Import Project: File → Import... → General → Existing Project into Workspace

Browse to: MDC_CPS/MDC_input/Tutorial_EdgeDetection (OK → Finish)

Merging Process

To Run the MDC Merging Process

Create a new run configuration: Run > Run configurations..., right click on Orcc compilation then select New.

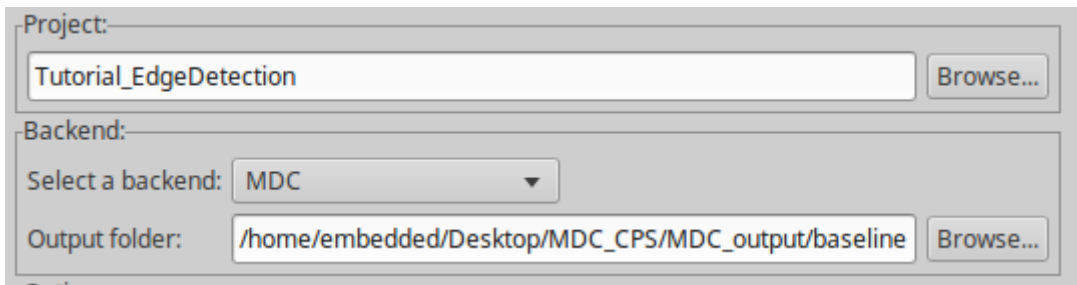
Name: chose a name for the configuration

Project: select *Tutorial_EdgeDetection* project

Backend

Select a backend: MDC

Output Folder: "MDC_CPS/MDC_output/baseline".



Options:

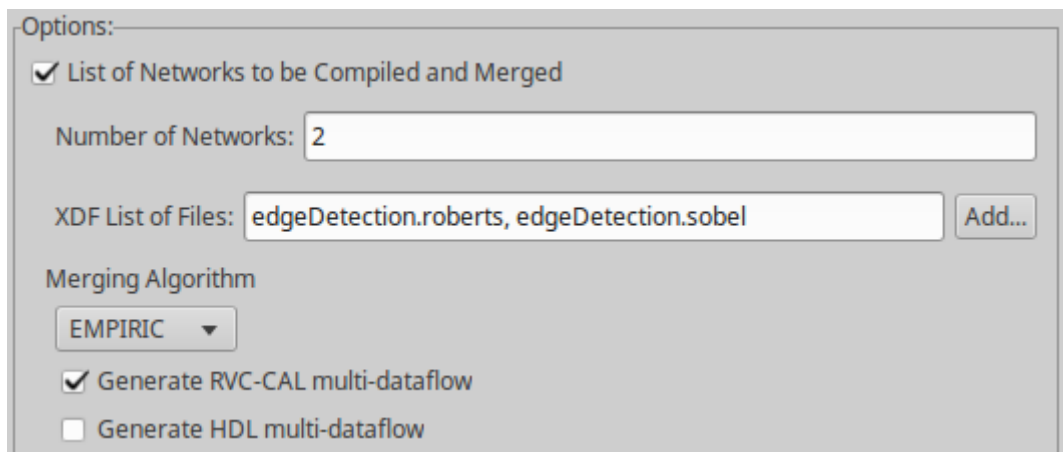
Tick “List of Networks to be Compiled and Merged”

Number of Networks: 2

XDF List of Files: select the two input dataflow networks *edgeDetection.roberts* and *edgeDetection.sobel*

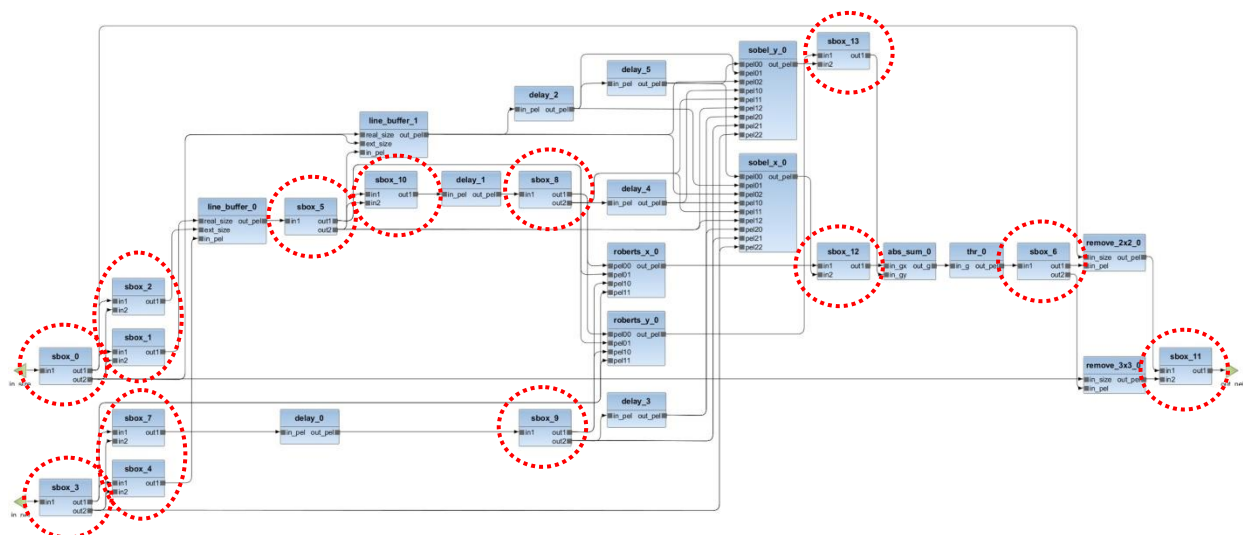
Merging Algorithm: *EMPIRIC*

Tick “Generate RVC-CAL multi-dataflow”. And Run.



This step merges the two input dataflow networks, through the selected datapath merging algorithm. Click on Run.

Refresh the project folder to visualize the output folder with the generated multi-dataflow. In the generated networks you can notice as actors are shared, and functionalities of the two input networks are guaranteed by the insertion of the switching boxes.



HDL Generation

Open again the configuration window: Run > Run configurations...

Choose previous configuration.

Options:

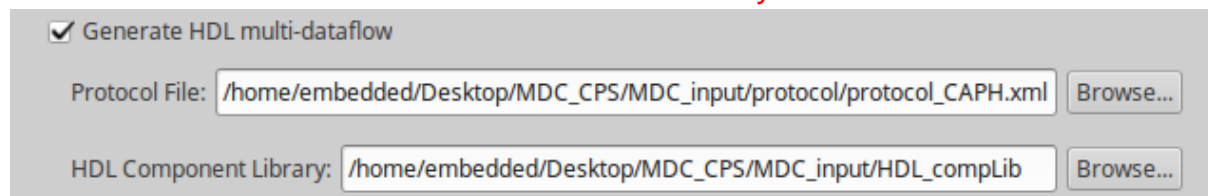
Deselect “Generate RVC-CAL multi-dataflow”

Select “Generate HDL multi-dataflow”.

Protocol file: “MDC_CPS/MDC_input/protocol/protocol_CAPH.xml”

HDL component library: “MDC_CPS/MDC_input/HDL_compLib”

The HDL component library must contain all the necessary HDL files. If a specific library is needed, files should be put in: Component_Library_Folder/lib/libName folder. *Please pay attention that libName folder name has to match the library name!*



Generate HDL multi-dataflow

Protocol File:

HDL Component Library:

Output folder

Output folder contains one folder and two files:

1. HDL: includes all the necessary files to create simulate and synthesize our CGR accelerator.

It contains also two files:

1. configNetID.txt - reports the ID value associated to each input dataflow.
2. report.txt - reports:
 - a. the number of actors of each input network
 - b. number of merged networks,
 - c. number of actors
 - d. number of original actors
 - e. number sbox actors
 - f. number of shared actors

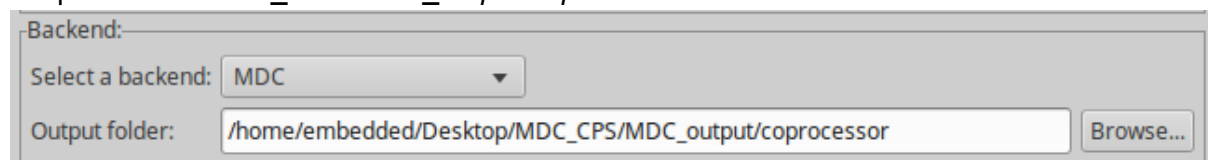
MDC CGR IP Generation

Open again the configuration window: Run > Run configurations...

Duplicate *baseline* configuration.

Backend:

Output folder: “MDC_CPS/MDC_output/coprocessor”.



Backend:

Select a backend:

Output folder:

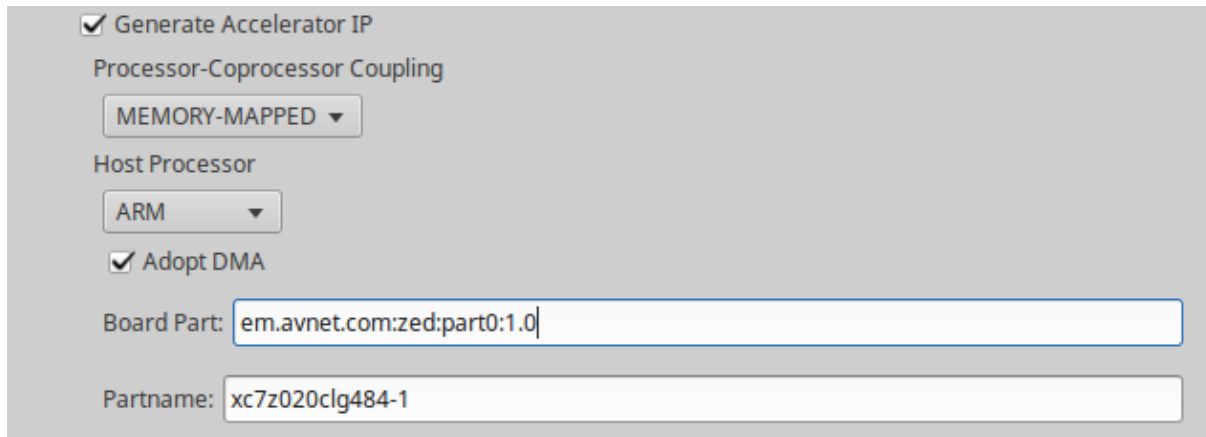
Options:

The previous settings for *Protocol file* and *HDL component library* are still valid.

Tick Generate Accelerator IP

Processor-Coprocessor Coupling: *MEMORY-MAPPED*

Host Processor: *ARM*
Tick on: Adopt DMA
Board Part: *em.avnet.com:zed:part0:1.0*
Partname: *xc7z020clg484-1*



Generate Accelerator IP

Processor-Coprocessor Coupling

MEMORY-MAPPED ▾

Host Processor

ARM ▾

Adopt DMA

Board Part:

Partname:

Select Apply and choose Run.

Output folder

Output folder contains two folders:

2. Mm_accelerator:
 - a. hdl: includes all the necessary files to create the Custom IP in Vivado.
 - b. bd: include the necessary file to properly import the Custom IP in a top project.
 - c. drivers: include the .c and .h files necessary to easily communicate with accelerator from the host processor.
3. Scripts: contains two scripts:
 - a. generate_ip.tcl - uses inputs in mm_accelerator to create a project and package a Custom IP.
 - b. generate_top.tcl - create a top project, where the IP is instantiated and connected to Host Processor, using the necessary logic according to user requirements.

By default, these scripts consider as root the folder where mm_accelerator and scripts folder are saved.

If Vivado is launched in that folder, they don't need any modification. If Vivado is launched in a different folder (e.g. Windows users), the users should open the scripts and replace root path "." (set root ".") with were necessary folders are saved.

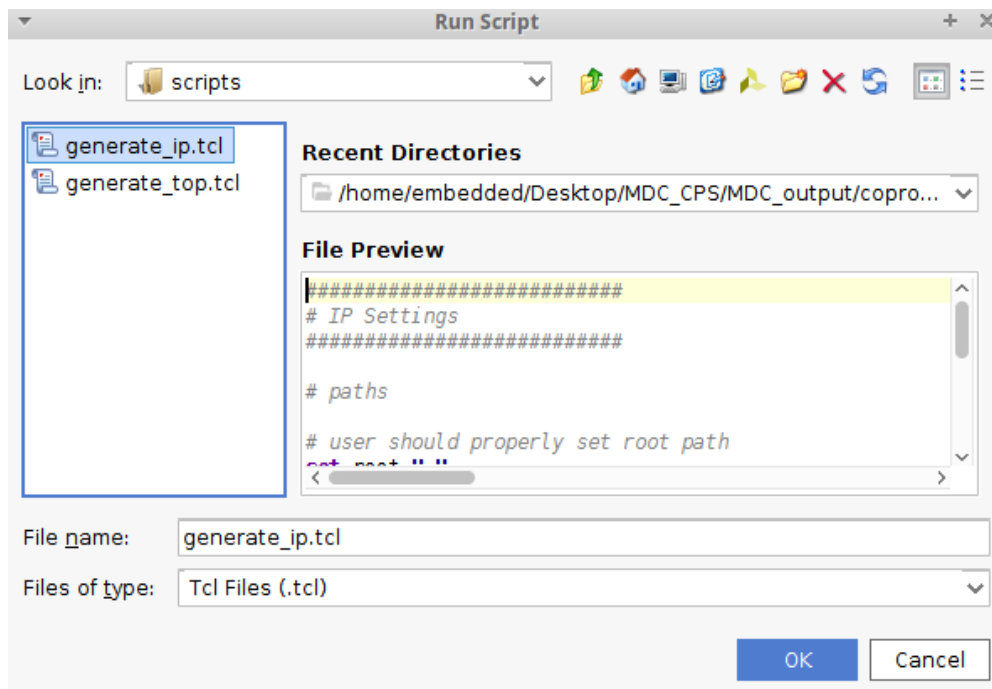
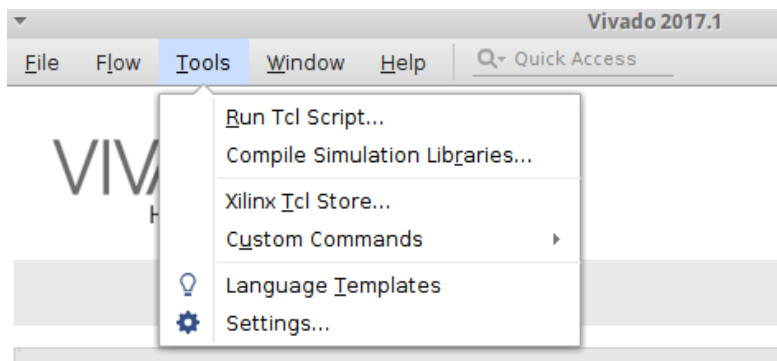
MDC Processor - Coprocessor System Generation

Open the terminal in the "MDC_CPS/MDC_output/coprocessor" folder.

Set Vivado Environment Variables: "source /opt/Xilinx/Vivado/2017.1/settings64.sh"

Launch Vivado typing "vivado"

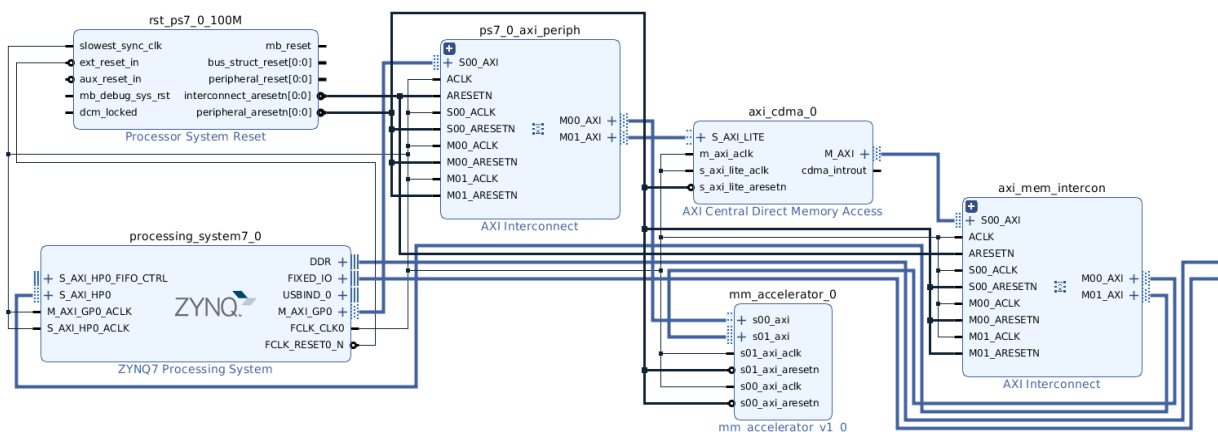
Tools → Run Tcl Script... and choose `./scripts/generate_ip.tcl`.



Then, launch the `generate_top.tcl` script file at the same manner:

Tools → Run Tcl Script... and choose `./scripts/generate_top.tcl`.

At this point, the Processor-Coprocessor System is ready for the synthesis or simulation.



ARTICo³ Kernel Generation

To use the coarse-grain reconfigurable computing core generated by MDC it is necessary to generate an ARTICo³ compliant kernel. An automatic script generates the required kernel (CGR_accelerator.v) starting from the MDC generated mm_accelerator.v

Open the terminal in *MDC_CPS* folder

Type "*source generateArticoKernel*"

Enter input file path:

/home/embedded/Desktop/MDC_CPS/MDC_output/coprocessor/mm_accelerator/hdl

Enter output file path:

/home/embedded/Desktop/MDC_CPS/MDC_output/coprocessor/mm_accelerator/hdl

Enter number of inputs: 2

Enter number of outputs: 1

In the output folder is now present the ARTICo³ compliant *cgr_accelerator.v*

System Implementation

NOTE: additional information about ARTICo³ project requirements and kernel specification can be found in /home/embedded/artico3/doc.

The ARTICo³ toolchain is required to build both FPGA configuration files (bitstreams) and application executable for the ARM microprocessors (ELF). For this, a specific folder structure is required.

Open a terminal (*terminal #1*) in */home/embedded/Desktop*

Run **mkdir -p tutorial/src/a3_cgr_accelerator/verilog** (hardware components)

Run **mkdir -p tutorial/src/application** (software components)

Run **touch tutorial/build.cfg** (create ARTICo³ project configuration file)

Project Configuration

Open */home/embedded/Desktop/tutorial/build.cfg* with a text editor (e.g., Geany).

Write general project settings:

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

Write kernel-specific settings:

```
[A3Kernel@CGR_accelerator]
HwSource = verilog
MemBytes = 49152
MemBanks = 3
```

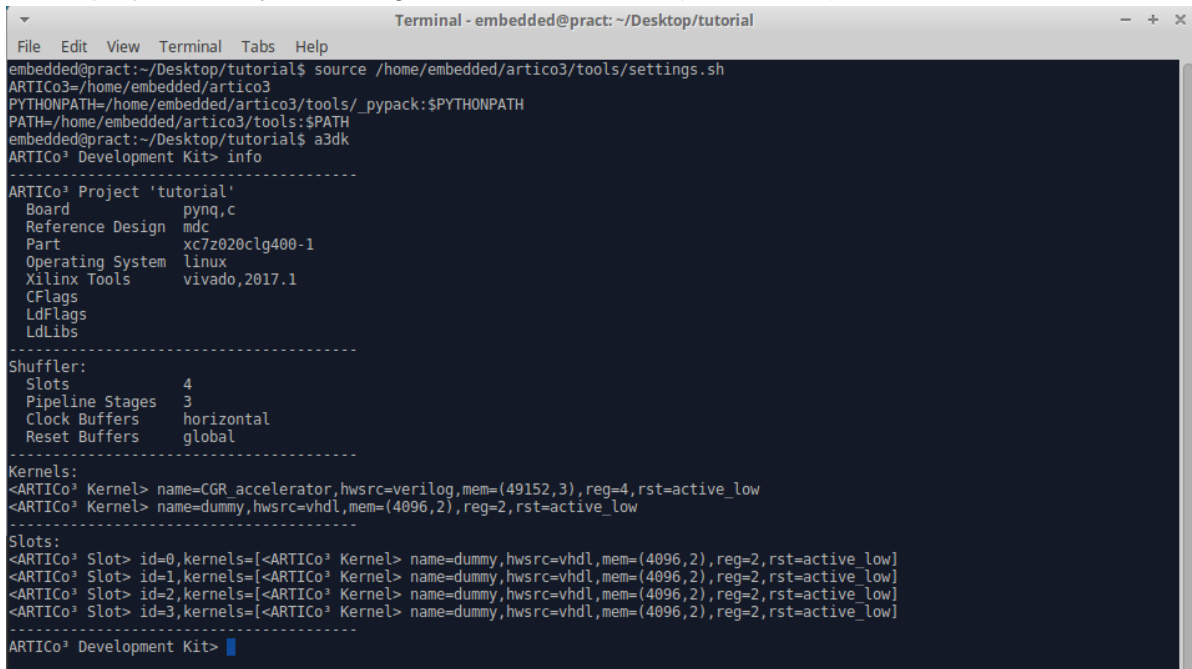
```
Regs = 4
RstPol = low
```

Save file and close.

Launch the ARTICo³ toolchain from a terminal (*terminal #2*):

```
cd /home/embedded/Desktop/tutorial
source /home/embedded/artico3/tools/settings.sh
a3dk
```

Check project info by executing the **info** command (*terminal #2*):



```
Terminal - embedded@pract: ~/Desktop/tutorial
File Edit View Terminal Tabs Help
embedded@pract:~/Desktop/tutorial$ source /home/embedded/artico3/tools/settings.sh
ARTICo3=/home/embedded/artico3
PYTHONPATH=/home/embedded/artico3/tools/_pypack:$PYTHONPATH
PATH=/home/embedded/artico3/tools:$PATH
embedded@pract:~/Desktop/tutorial$ a3dk
ARTICo3 Development Kit> info
-----
ARTICo3 Project 'tutorial'
Board          pynq,c
Reference Design mdc
Part           xc7z020clg400-1
Operating System linux
Xilinx Tools   vivado,2017.1
CFlags
LdFlags
LdLibs
-----
Shuffler:
Slots          4
Pipeline Stages 3
Clock Buffers  horizontal
Reset Buffers  global
-----
Kernels:
<ARTICo3 Kernel> name=CGR_accelerator,hwsrc=verilog,mem=(49152,3),reg=4,rst=active_low
<ARTICo3 Kernel> name=dummy,hwsrc=vhdl,mem=(4096,2),reg=2,rst=active_low
-----
Slots:
<ARTICo3 Slot> id=0,kernels=[<ARTICo3 Kernel> name=dummy,hwsrc=vhdl,mem=(4096,2),reg=2,rst=active_low]
<ARTICo3 Slot> id=1,kernels=[<ARTICo3 Kernel> name=dummy,hwsrc=vhdl,mem=(4096,2),reg=2,rst=active_low]
<ARTICo3 Slot> id=2,kernels=[<ARTICo3 Kernel> name=dummy,hwsrc=vhdl,mem=(4096,2),reg=2,rst=active_low]
<ARTICo3 Slot> id=3,kernels=[<ARTICo3 Kernel> name=dummy,hwsrc=vhdl,mem=(4096,2),reg=2,rst=active_low]
-----
ARTICo3 Development Kit> █
```

Hardware Components

Copy all files generated by MDC to the ARTICo³ project folders (*terminal #1*):

```
cp -rf
/home/embedded/Desktop/MDC_CPS/MDC_output/coprocessor/mm_accelerator/hdl/*
/home/embedded/Desktop/tutorial/src/a3_cgr_accelerator/verilog
```

Remove AXI compliant interface and simulation files (*terminal #1*):

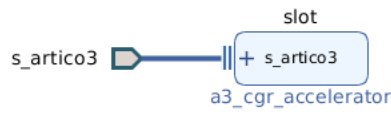
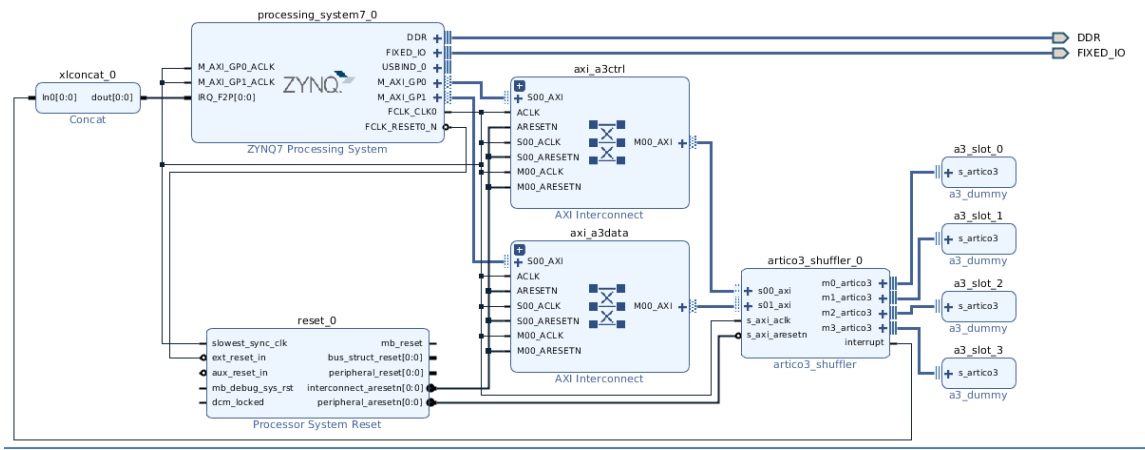
```
cd /home/embedded/Desktop/tutorial/src/a3_cgr_accelerator/verilog
rm -rf mm_accelerator.v tb_mm_accelerator.v lib tb
```

Generate Vivado project with **export_hw** (*terminal #2*).

Check generated block diagram (*terminal #1*):

```
cd /home/embedded/Desktop/tutorial/build.hw
source /opt/Xilinx/Vivado/2017.1/settings64.sh
vivado myARTICo3.xpr
```


Open both block diagrams from Vivado (In Project Manager window: IP Integrator -> Open Block Design).



[SKIP] Build Vivado project with **build_hw** (terminal #2).

Software Components

Write code in a text editor (e.g., Geany) as instructed and save it as /home/embedded/Desktop/tutorial/src/application/main.c

Generate software project with **export_sw** (terminal #2).

Build software project with **build_sw** (terminal #2).