Tutorial: Multi-Dataflow Composer Tool

CGR Coprocessor Generation from Dataflow Networks

This tutorial shows how to generate a processor-coprocessor AXI-based system for Xilinx FPGA using the MDC tool (http://sites.unica.it/rpct/), and how to simulate it.

The multi-dataflow network performs two functionalities for edge detection: Sobel and Roberts operators. Both algorithms consider the convolution of two kernels with an grayscale image (as shown in Fig. 1), in order to highlight the high-frequency variations due to the horizontal and vertical edges.

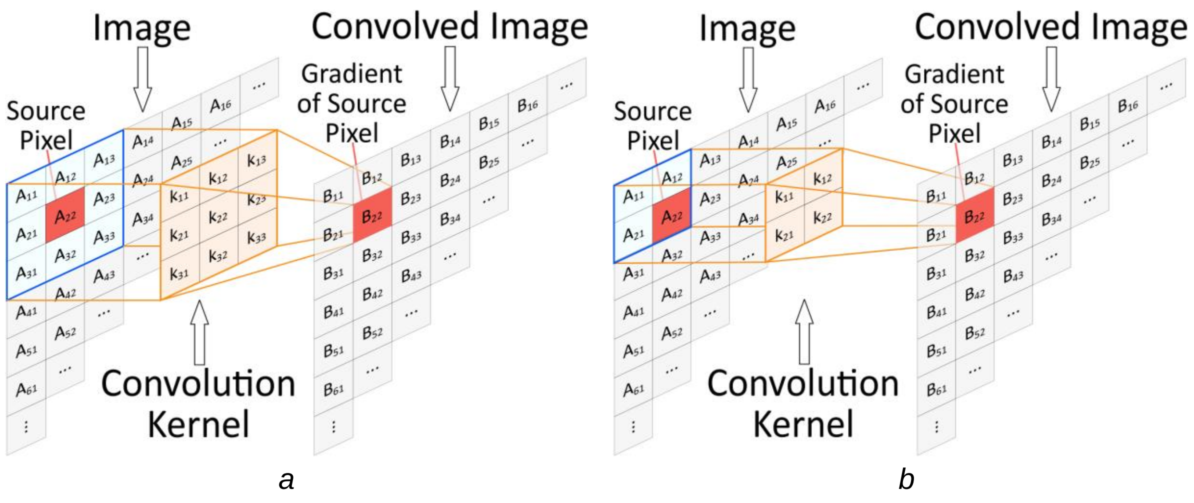
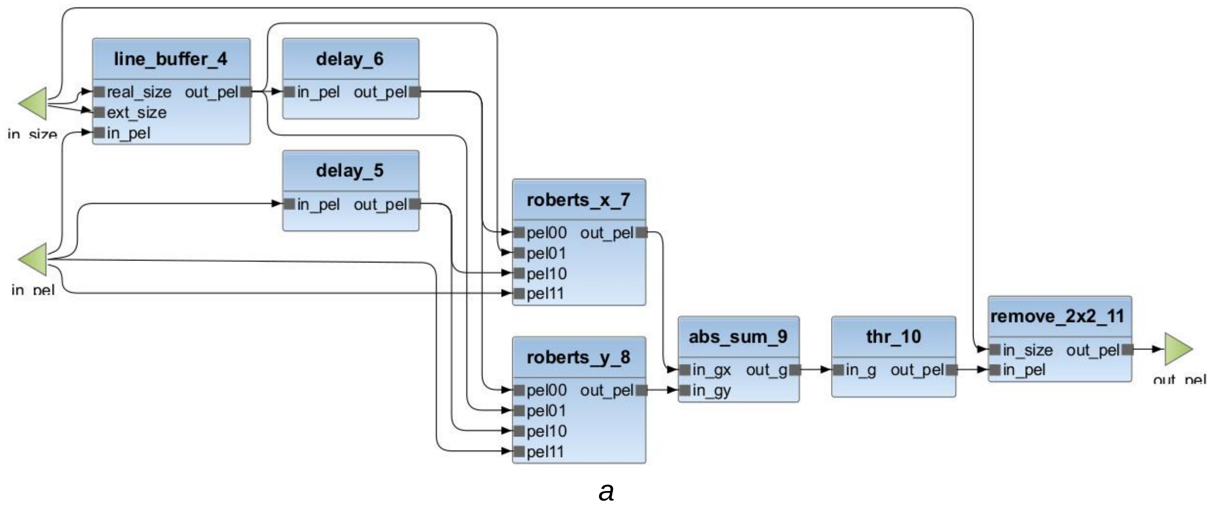
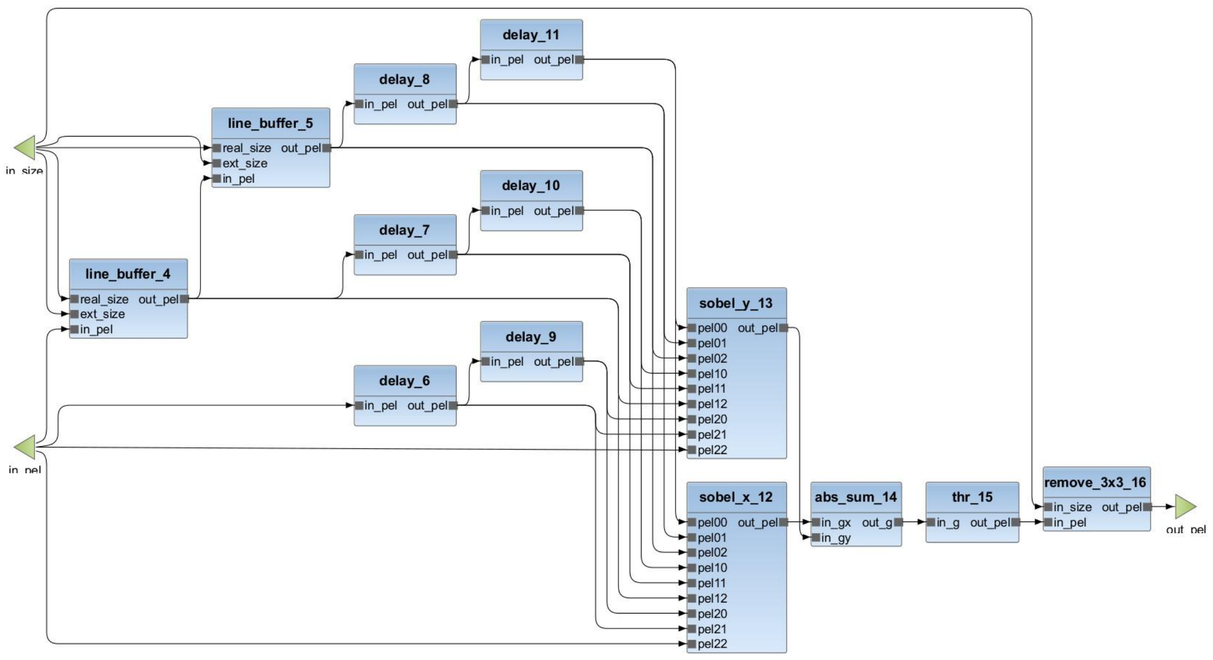


Figure 1: Computation of the convolution of kernels (x and y) of the Sobel (a) and Roberts (b) operators with an input image (A).

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





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).

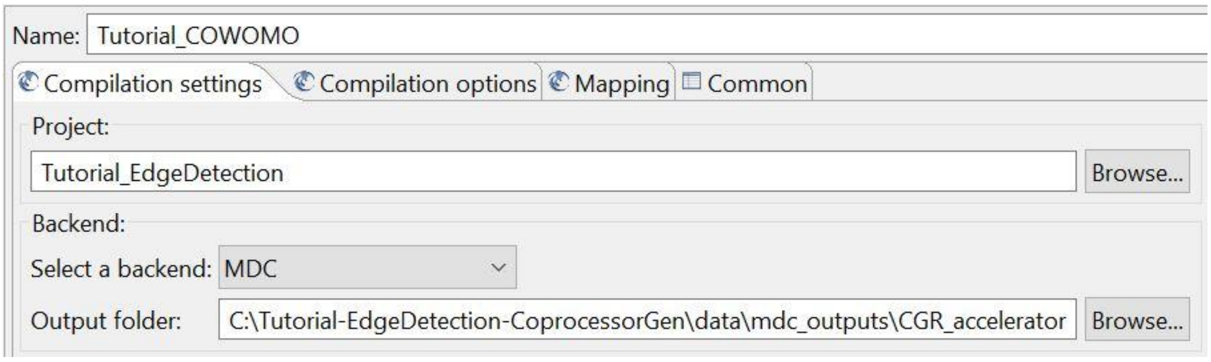
Necessary tools:

- MDC tool (to install this tool without a provided executable file, please, follow installation guide);
- Java Runtime Environment (v8 or higher);
- Xilinx Vivado tool. (Tutorial has been tested with Vivado 2017.1, 2017.3, 2017.4 and 2018.1);
- A Xilinx FPGA board.

1. Merging Process

- In folder COWOMOtutorial/MDC_tool/eclipse launch *MDC* executable;
- Select the folder where is your executable file of MDC as a workspace and then choose OK.

Create a new run configuration: Run > Run configurations… , right click on Orcc compilation then select New. Now, chose a name for the configuration, and select *Tutorial_EdgeDetection* project, MDC backend and the output folder path.


Name: Tutorial_COWOMO

Compilation settings | Compilation options | Mapping | Common

Project:

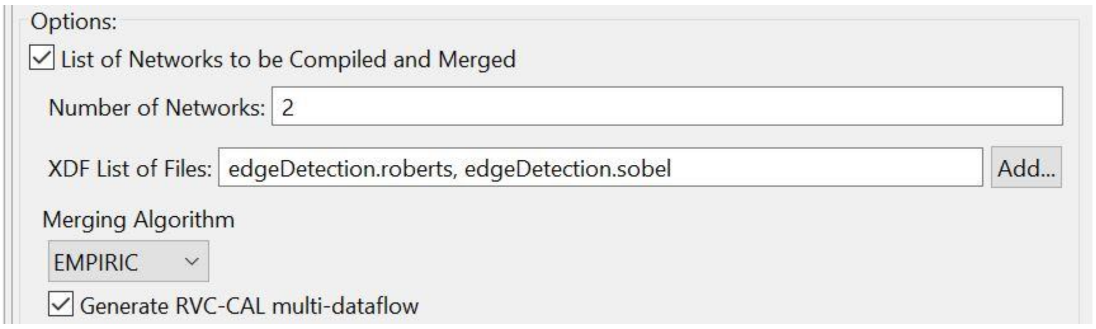
Tutorial_EdgeDetection [Browse...]

Backend:

Select a backend: MDC

Output folder: C:\Tutorial-EdgeDetection-CoprocessorGen\data\mdc_outputs\CGR_accelerator [Browse...]

Then select the two input dataflow networks: edgeDetection.roberts and edgeDetection.sobel and choose the merging algorithm (EMPIRIC or MOREANO, ref). In this tutorial we adopt the EMPIRIC one.


Options:

☑ List of Networks to be Compiled and Merged

Number of Networks: 2

XDF List of Files: edgeDetection.roberts, edgeDetection.sobel [Add...]

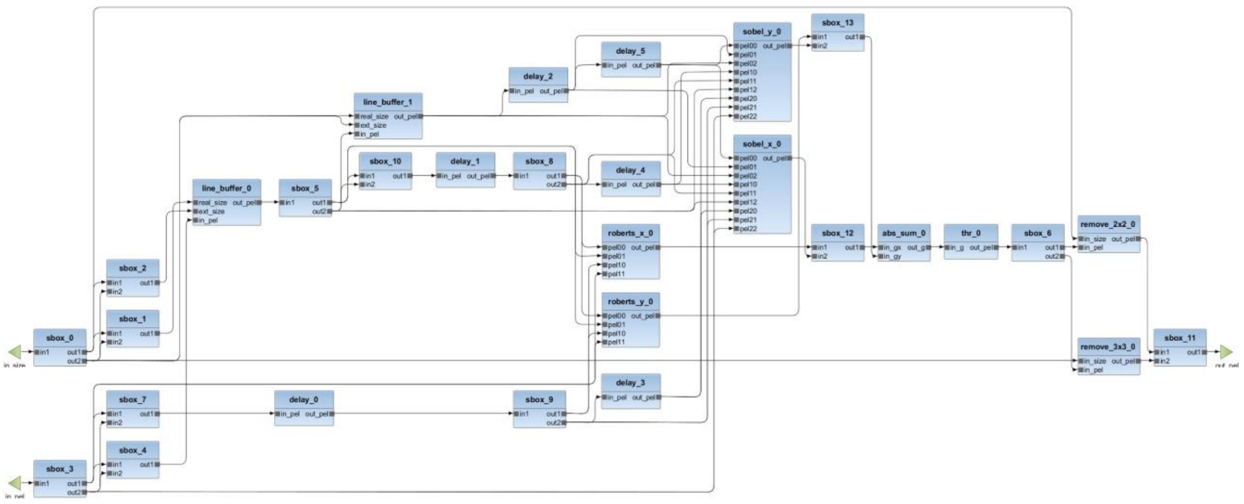
Merging Algorithm

EMPIRIC

☑ Generate RVC-CAL multi-dataflow

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.

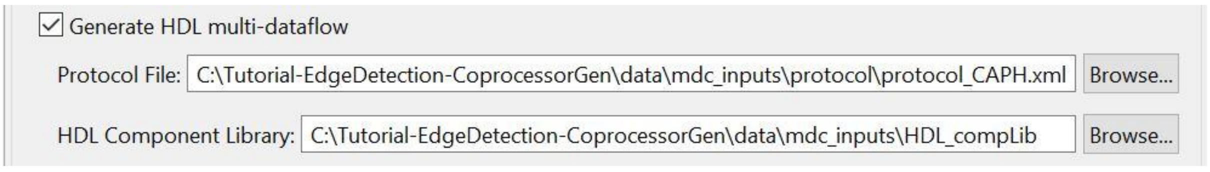


2. HDL Generation

Open again the configuration window: Run > Run configurations...
Choose previous configuration, unselect “Generate RVC-CAL multi-dataflow” and tick “Generate HDL multi-dataflow”.
Then choose the protocol file and the HDL component Library.

1. Protocol file is in *Tutorial-EdgeDetection-CoprocessorGen/data/mdc_inputs/protocol/* (link) folder.
2. HDL component library is in *Tutorial-EdgeDetection-CoprocessorGen/data/mdc_inputs/HDL_compLib* (link) folder.

The HDL component library has to contain all of the necessary hdl files. If some file, related to a specific library are needed, the 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: C:\Tutorial-EdgeDetection-CoprocessorGen\data\mdc_inputs\protocol\protocol_CAPH.xml

HDL Component Library: C:\Tutorial-EdgeDetection-CoprocessorGen\data\mdc_inputs\HDL_compLib

Output folder

Output folder contains one folder and two files:

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

I contains also two files:

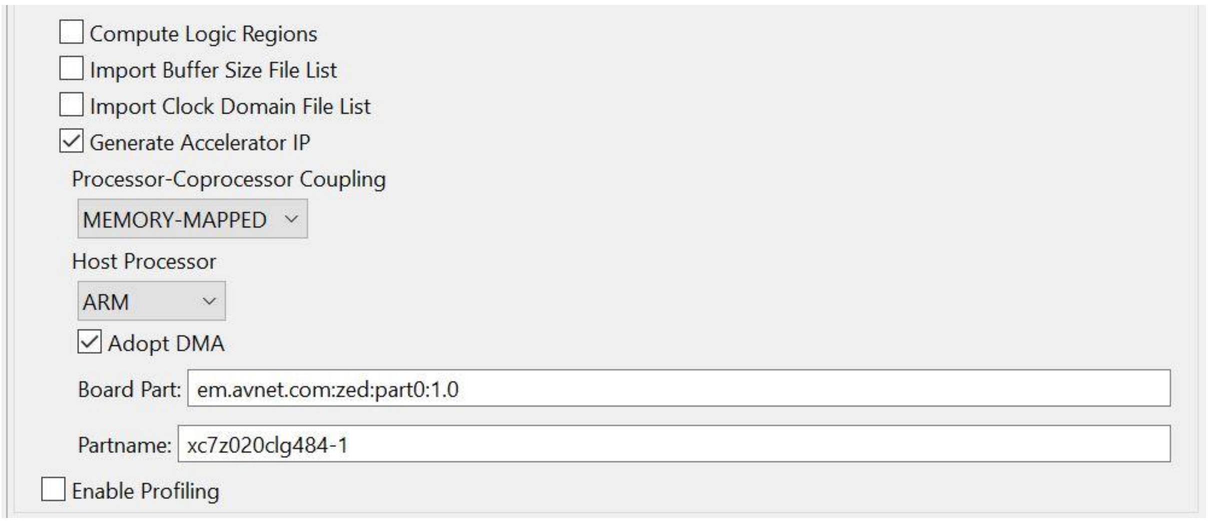
1. configNetID.txt - reports the ID value associated to each input dataflow.
2. report.txt - reports:
    a. the number of actors of each input network
    b. number of merged networks,
    c. number of actors
    d. number of original actors
    e. number sbox actors
    f. number of shared actors

3. Coprocessor Generation

To generate the Ip, you need to choose:

1. The processor-coprocessor coupling (ref), memory-mapped or stream . In this tutorial the memory-mapped is chosen.
2. The Host Processor. In this tutorial we use the ARM Processor.
3. If you want to use the DMA module (ref). We tick this box.
4. Kind of Xilinx board. In this tutorial the ZedBoard Zynq Evaluation and Development Kit is adopted:
    a. Board Part: *em.avnet.com:zed:part0:1.0*

b. Partname: *xc7z020clg484-1*



- [ ] Compute Logic Regions
- [ ] Import Buffer Size File List
- [ ] Import Clock Domain File List
- [✓] Generate Accelerator IP

Processor-Coprocessor Coupling: MEMORY-MAPPED

Host Processor: ARM

- [✓] Adopt DMA

Board Part: em.avnet.com:zed:part0:1.0

Partname: xc7z020clg484-1

- [ ] Enable Profiling

Select Apply and choose Run.

Output folder

Output folder contains two folders:

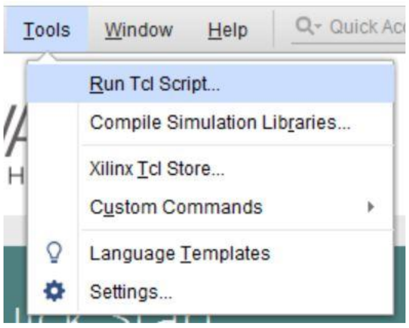
2. Mm_accelerator:
    a. hdl: includes all of the necessary files to create the Custom IP in Vivado.
    b. bd: include the necessary file to properly import the Custom IP in a top project.
    c. drivers: include the .c and .h files necessary to easily communicate with accelerator from the host processor.
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.

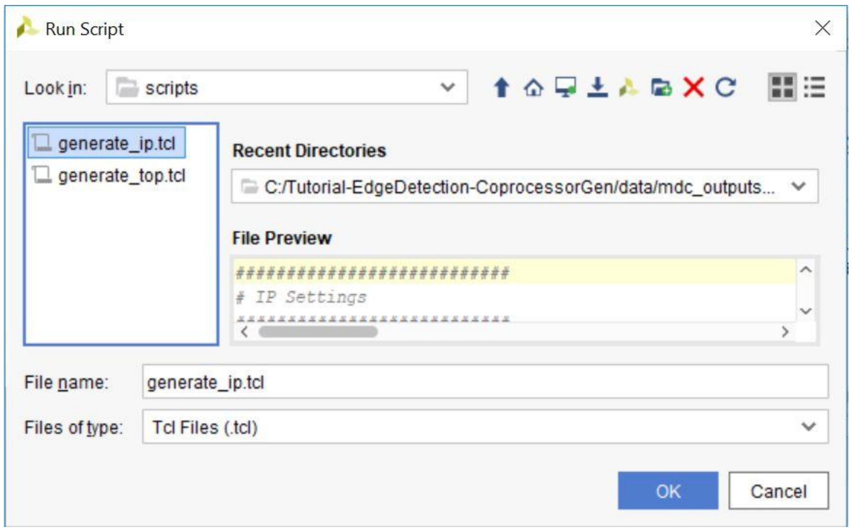
By default these scripts consider as root the folder where mm_accelerator and scripts folder are saved.
If vivado is launched in that folder, they don’t need any modification. If Vivado is launched in a different folder (e.g Windows users), the users should open the scripts and replace root path “.” (set root ".") with were necessary folders are saved.

4. Processor - Coprocessor System Generation

Launch Vivado.
Change directory to the root folder of the generated scripts, as:
.../data/mdc_outputs/CGR_coprocessor

Now, in the main menu (only in Linux environment): Tools → Run tcl Script… and choose .../CGR_coprocessor/scripts/generate_ip.tcl.

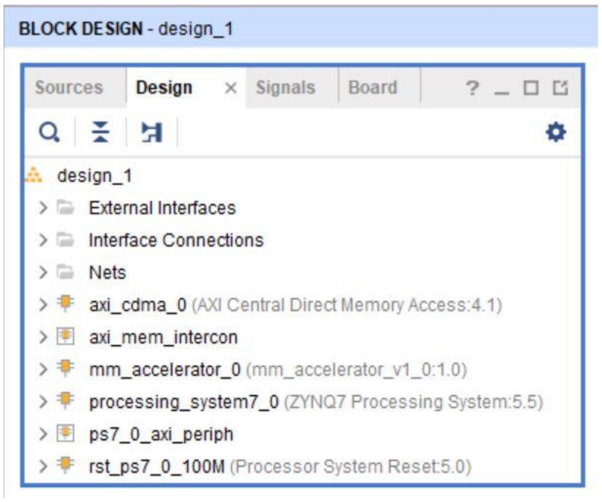


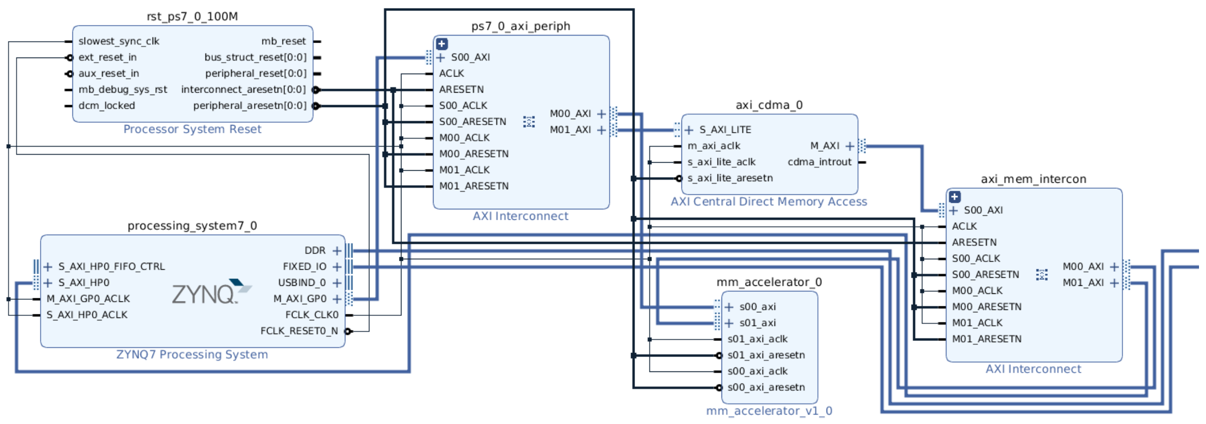


```
###########################
# IP Settings
###########################
```

When the IP is generate, it is possible to see the HDL files in the Source panel.

Then, launch the other script file at the same manner: Tools → Run tcl Script… and choose .../CGR_coprocessor/scripts/generate_top.tcl.
At this point, the Processor-Coprocessor System is ready for the synthesis or simulation.


BLOCK DESIGN - design_1
Sources
Design
Signals
Board
design_1
External Interfaces
Interface Connections
Nets
axi_cdma_0 (AXI Central Direct Memory Access:4.1)
axi_mem_intercon
mm_accelerator_0 (mm_accelerator_v1_0:1.0)
processing_system7_0 (ZYNQ7 Processing System:5.5)
ps7_0_axi_periph
rst_ps7_0_100M (Processor System Reset:5.0)



rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M01_ACLK
M01_ARESETN
M00_AXI
M01_AXI
AXI Interconnect
axi_cdma_0
S_AXI_LITE
m_axi_aclk
s_axi_lite_aclk
s_axi_lite_aresetn
M_AXI
cdma_introut
AXI Central Direct Memory Access
axi_mem_intercon
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M01_ACLK
M01_ARESETN
M00_AXI
M01_AXI
AXI Interconnect
processing_system7_0
S_AXI_HP0_FIFO_CTRL
S_AXI_HP0
M_AXI_GP0_ACLK
S_AXI_HP0_ACLK
ZYNQ
DDR
FIXED_IO
USBIND_0
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
mm_accelerator_0
s00_axi
s01_axi
s01_axi_aclk
s01_axi_aresetn
s00_axi_aclk
s00_axi_aresetn
mm_accelerator_v1_0