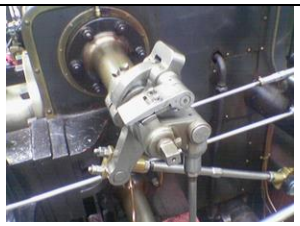
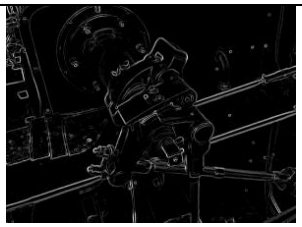
1. General description

This tutorial involves three different algorithms for edge detection, described as CAL dataflow networks: Roberts, Scharr and Sobel. Following images show

			
Input image	*Roberts output*	*Scharr output*	*Sobel output*

Firstly we are going to learn how to run a CAL simulation, and then we will exploit the Multi-Dataflow composer tool for the generation of Coarse-Grain Reconfigurable Systems. In particular, we are going to test three features of MDC tool:

Merging network: this is the baseline feature that takes different xdf networks, merges them through a datapath merging algorithm and generates a multi functional xdf network.

HDL Generation: this feature processes the generated multi-dataflow.xdf and generates the corresponding Verilog code, according to the communication protocol specified by user.

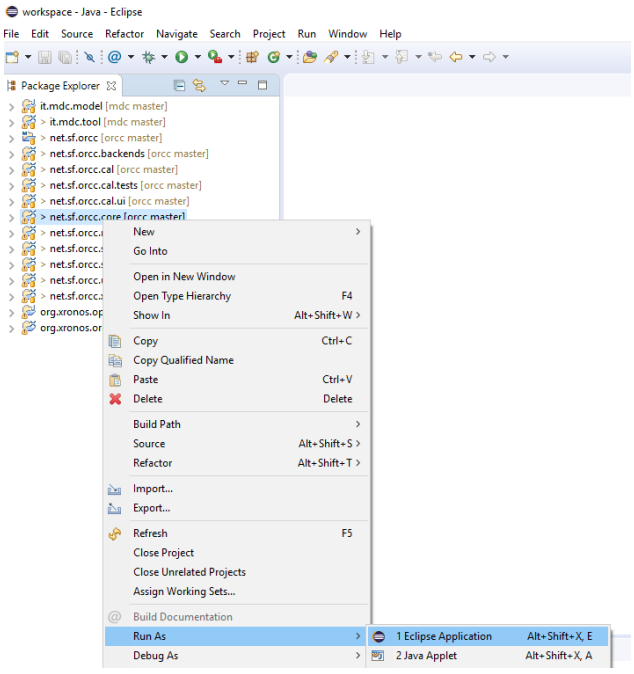
Clock Gating Application: this feature processes the generated multi-dataflow.xdf and generates the corresponding Verilog code where clock gating technique is automatically applied to switch off the clock to that part of the logic not involved in the computation.

2. Launch Runtime Eclipse Application

Firstly we have to launch the Runtime Eclipse Application Workspace. In the Package Explorer:

- Right click on net.sf.orcc.core
- Run As
- Eclipse Application

These steps open the Runtime Eclipse Application Workspace



3. Cal Project Explorer

- src (RVC-CAL source files)
  - common (common actors between different application fields)
  - edgeDetection (edge detection actors)
  - edgeDetectionXDF (overall network, input edge detection and testbench xdf network)
  - std (standard actors for sourcing and sinking)
    - stdio (source actor able to read single component binary inputs)
    - video (display actor able to display a YUV picture to video)
- reference (reference I/O)
  - edgeDetection (reference output for the edge detection applications)
  - input (reference input image)
  - protocol (communication protocol file)

3.1. Cal simulation of individual dataflow graphs

Expand the *edgeDetectionXDF* folder, and open *Testbench.xdf*.

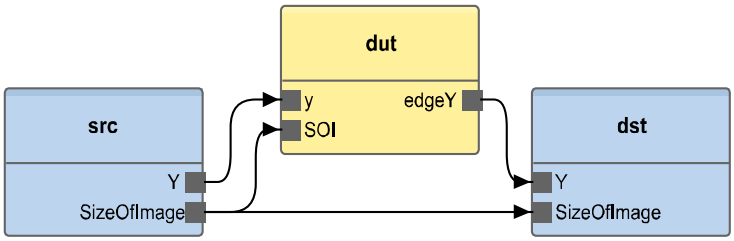


Figura 1 - testbench cal network

The design under test (DUT) instance is one of the three considered networks (Roberts, Scharr and Sobel). By opening the dut instance (double click on it) you can visualize the network currently instantiated.

To simulate the cal network:

- Click on *Testbench.xdf* → *Run as* → *ORCC Simulation that* opens a *Select simulator* window.
- Press OK and choose the input image "*test_image.bin*" in *reference/input* path.
- Click on Apply and thus on Run.

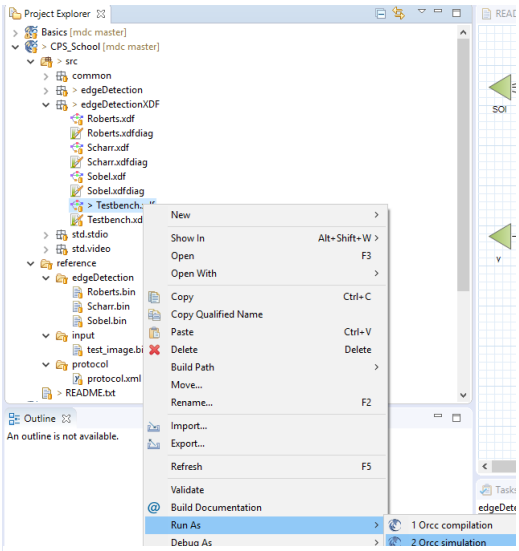


Figura 2 - Run ORCC simulation

Compare the visualized image with the expected output.

- Now let's simulate another network:
- Right click on the dut instance
- Click on Set/Update refinement
- Select a new xdf network
- OK
- Save the modified testbench

Save Testbench.xdf and run again the simulation steps.

4. MDC Merging networks

In the menu toolbar: Run → Run Configuration (it opens the Run Configuration window).

Select Orcc Compilation

Create a new configuration

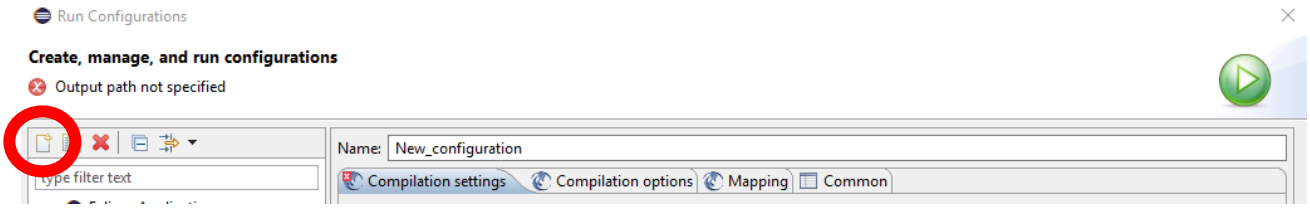


Figura 3 - Create new configuration

Project:

- Name: edge_det_merging
- Project: CPS_School
- Select a backend: MDC
- Output folder: MDCoutput/school

Options:

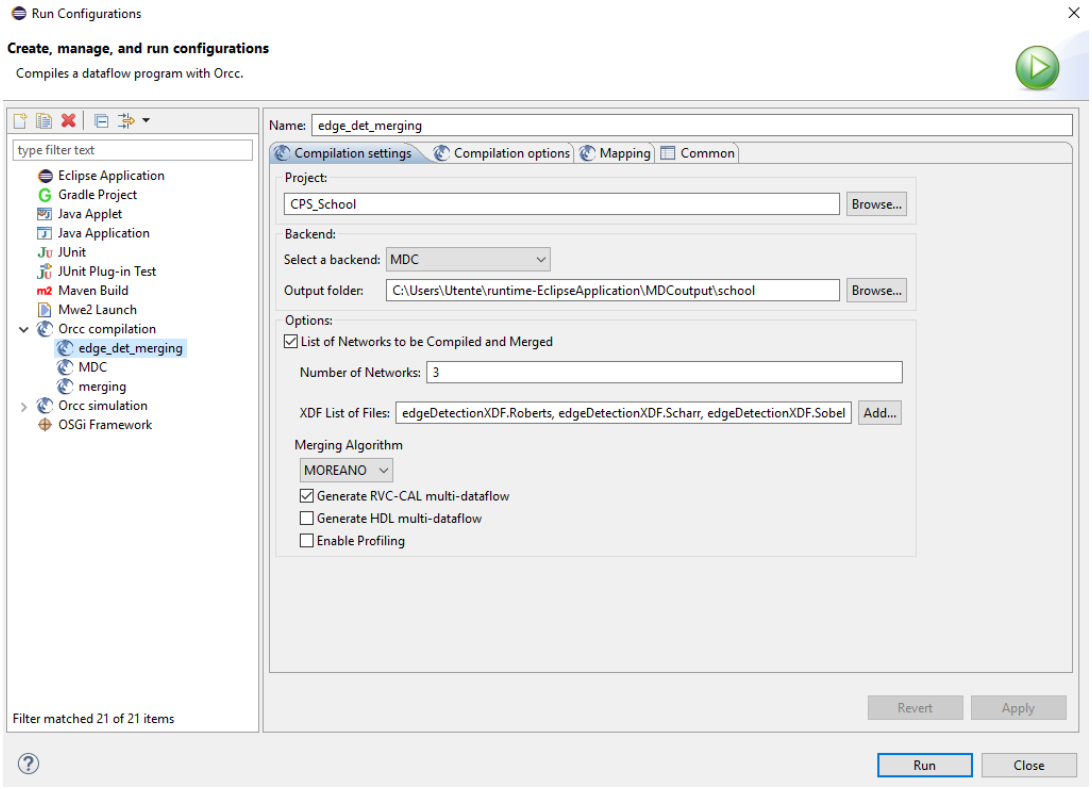
- Tick *List of Networks to be Compiled and Merged*
    - Number of Networks: 3
    - XDF List of Files: Roberts.xdf, Scharr.xdf, Sobel.xdf
- Merging Algorithm: MOREANO
- Tick Generate RVC-CAL multi-dataflow
- CAL type: DYNAMIC
- 

Figura 4 - merging configuration

Now you can apply settings and run the merging process!

This step merges the three input xdf networks into one multi functional xdf network, multi_dataflow.xdf. The automatic insertion of ad hoc switching elements (sboxes) guarantees the correct flow of the data with respect to the current functionality to be executed.

Do a refresh of the project: right click on CPS_School > Refresh

The created ***mdc_date_hour*** folder (e.g. see mdc_14_9_2017_17_5_93 folder in **Errore. L'origine riferimento non è stata trovata.**) contains the xdf network, and mdc_date_hour.cal folder contains cal files of sboxes.

5. HDL Generation

To generate the HDL corresponding to the previous generated multi_dataflow.xdf open the Run Configuration windows (In the menu toolbar: Run → Run Configuration) and select the previous created "edge_det_merging" configuration.

- Since we don't need to generate the multi_dataflow.xdf again, un-tick the *Generate RCV-CAL multi-dataflow*.
- Tick *Generate HDL multi-dataflow*
- Preferred HDL protocol: CUSTOM full (beta)
- Tick *Specify a Custom Hardware Communication Protocol* and choose protocol.xml in reference/protocol
- Run

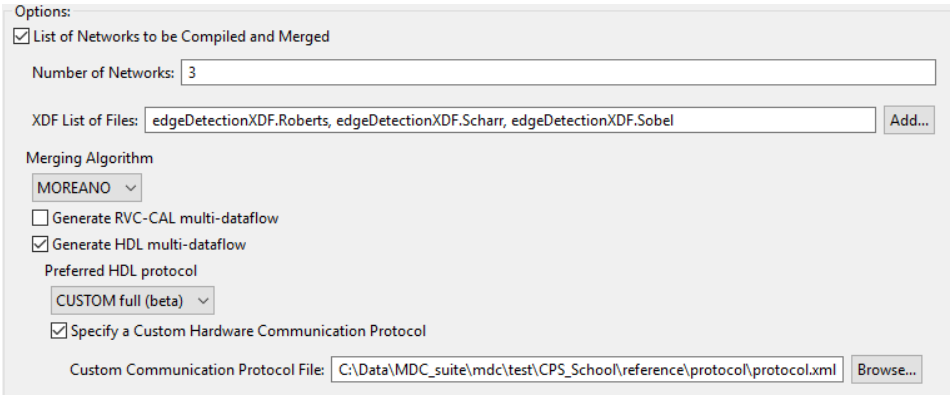


Figura 5 - generate HDL

This step generated the following output in the output folder:

- Folder: hdl/Verilog
  - multi_dataflow.v – it is the top Verilog module, corresponding to the multi_dataflow.xdf network.
  - sbox1x2.v – switching element with one input and two outputs.
  - sbox2x1.v – switching element with two inputs and one output.
  - configurator.v – is in charge of set the sboxes selectors according to the input ID network.
- configNetID.txt: reports an ID value for each input network involved in the merging process.
- report.txt: reports the number of actors of each input network and the number of actors in the merged network.

Other folders were already present in the output folder.

5.1. HDL simulation

Open new terminal, navigate to *school* folder and launch Vivado.

- To move to a folder type: cd <name_folder>
- To move out of a folder type: cd ..
- To launch Vivado type: vivado

To create the project and import the necessary files, in the menu toolbar: Tool → Run Tcl Script... → Select tcl/*generate_project.tcl*

In the source window you can navigate the project files. If you expand dut: multi_dataflow you can see the Verilog modules corresponding to the cal actors. According to the communication protocol we selected, actors exchange data through FIFO modules. Sboxes actors properly route data.

The given testbench reads the input test image and compare the computed image output with the expected one. If output data does not match the expected one, an error message is displayed and simulation stops.

This testbench is able to run only one functionality per time, thus we need to modify the testbench file, reading the proper image for comparison and setting the ID to the proper value (according to the configNetID file).

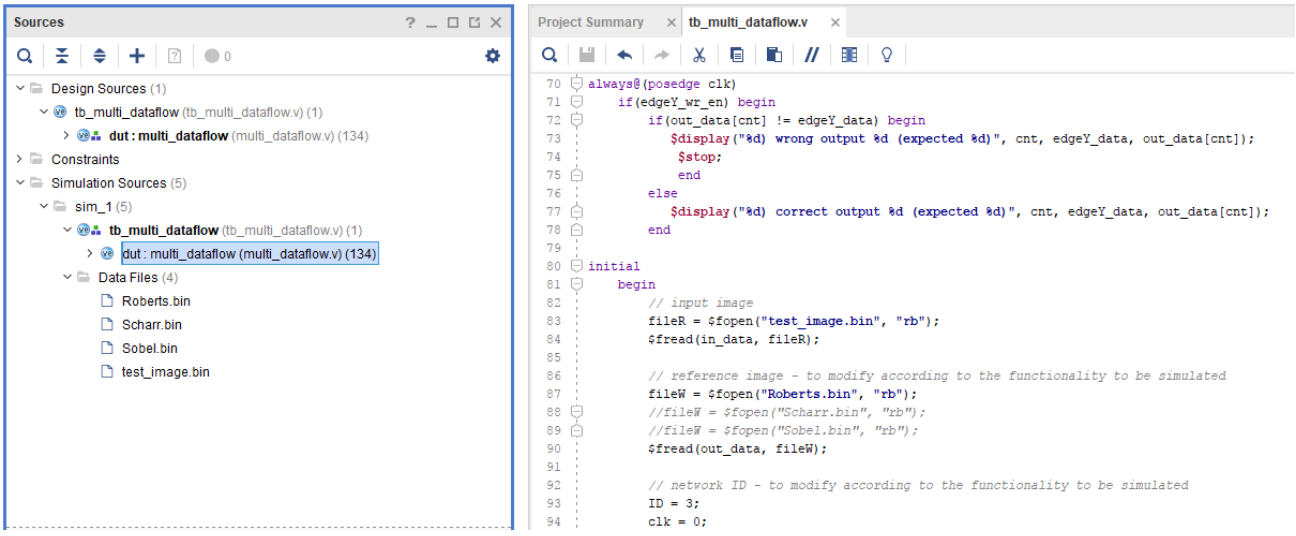


Figura 6 - multi-dataflow Verilog testbench

Then, to run simulation: Tool → Run Tcl Script... → Select tcl/launch_simulation.tcl file.

In the tcl console you can verify as the output data match the expected ones.

6. Clock Gating Application

Let's go back to eclipse.

To generate the clock gated version of the previous HDL open the Run Configuration windows (In the menu toolbar: Run → Run Configuration) and select the previous created "edge_det_merging" configuration.

- Tick *Compute Logic Regions* (related to *Generate HDL multi-dataflow*)
- *Type of Design flow*: FPGA
- *Number of FPGA Dedicated Power Saving Cells*: 32
- *Logic Regions Power Saving Technique*: CLOCK_GATING

- Run

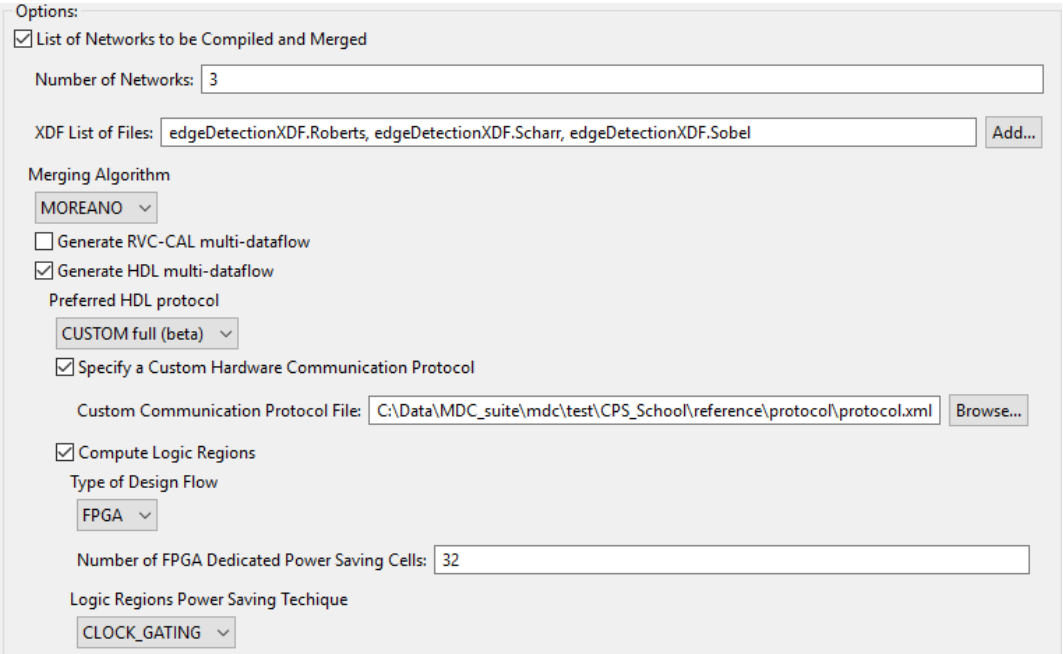


Figura 7 - Clock gated HDL generation

In the output folder we find an additional report file, reportLogicRegions.txt. This file reports:

- all of the sets of actors that are always active together (called Logic Regions).
- for each LR, the list of networks that activate it.
- for each network, the list of LRs that it activates.

When we execute a network only LRs belonging to that network are active, all of the others are clock gated. If a LR is shared by all of the networks it is always on, and it is never clock gated.

If we open the new multi_dataflow.v, in the hdl/verilog folder, we can see a clock gating cell (clock_gating_cell) instantiation for each switchable LR.

Each clock gating cell receives as input the system clock and an enable signal, and generates a clock signal that can be switched off according to the enable. Each switchable LRx is connected to a generated switchable clock ck_gated_x.

The enable generator (enable_generator) set the clock enables according to the current ID.

The clock_gating_cell.v and enable_generator.v files are automatically generated with the other necessary files.

With this new run the ID associated to each network could be different, thus take a look to the configNetID.txt.

6.1. HDL simulation

Close previous created project, and run again the *generate_project.tcl* script: Tool → Run Tcl Script... → Select tcl/*generate_project.tcl*

Before simulation, open the tb_multi_dataflow.v and check if the ID value and the expected image are correct if necessary modify them. Let's choose Scharr.

- We uncomment line: fileW = $fopen("Scharr.bin", "rb");
- and set ID value according to data in configNetID.txt.
- Launch simulation script: Tool → Run Tcl Script... → Select tcl/launch_simulation.tcl file.

In the Tcl console you can check the correctness of the output data. Let's see also some waveforms. Firstly we look, in the reportLogicRegions.txt, for a LR which is not used by Sobel and for an actor belonging to this region (e.g. LR2, Delay_1). Then in Vivado:

- In the scope window, expand dut and look for actor_Delay_1, and add it to the wave window (right click on it → add to Wave Window).
- Restart simulation, in the toolbar: Run → Restart...
- Run simulation, in the toolbar: Run → Run All...

You can appreciate as in this actor, the clock is off, and there is not any switching activity, since it does not receive any input data.

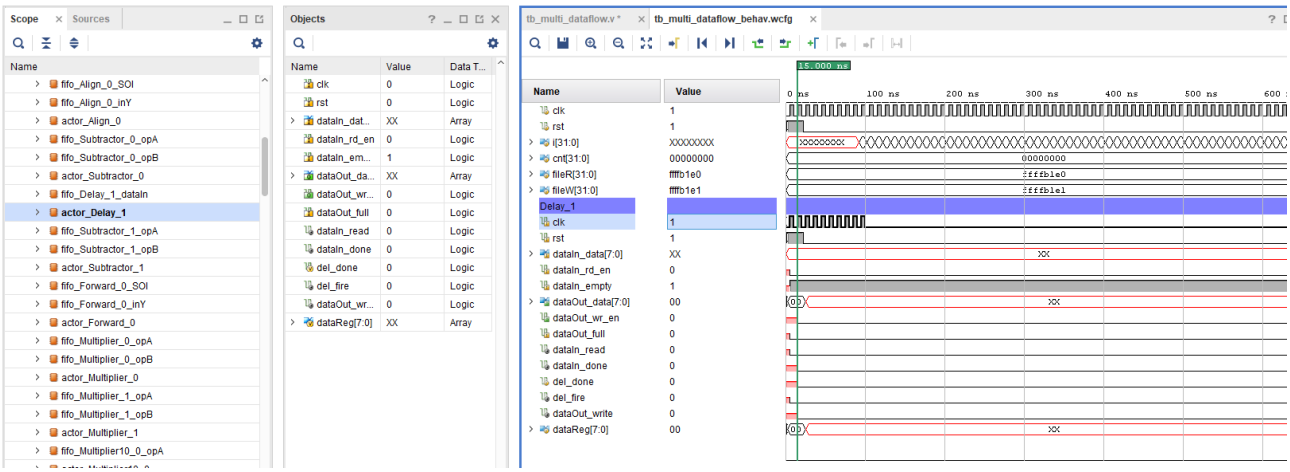


Figura 8 - Waveform windows with an example of a clock gated actor.