Implementation of Dynamically Reconfigurable Systems using commercial Xilinx FPGAs - Tutorial

Andrés Otero, Alfonso Rodriguez, Eduardo de la Torre. Centro de Electrónica Industrial - UPM

This document provides a short summary with the mains steps to be carried our for the implementation of reconfigurable systems using the Xilinx commercial toolflow. It has been tested with the Digilent Pynq Board *(FPGA ZYNQ XC7Z020-1CLG400C)*.

The system to be designed in this tutorial is composed of a static system that includes an embedded processor, as well as two reconfigurable regions (or partitions). For each of the reconfigurable regions, two reconfigurable modules are generated: an adder and a multiplier. Each of them will act as an IP peripheral of the processors. Partial bitstreams are created for the configuration of each of the modules in each of the regions.

The design process comprises two steps: first, reconfigurable modules are synthesized and then the static system is generated.

IMPLEMENTATION OF THE RECONFIGURABLE MODULES

(Repeat these steps for each reconfigurable module, Adder and Multiplier)

1. Create a new Vivado project.
2. Add the HDL file corresponding to the reconfigurable module as the new Top in the project. The entity of the module must match the entity of the *blackboxes* used in the static system.
3. Synthesize the modules (in *Out of context* mode to avoid the instantiation of IOBs)

```
synth_design -mode out_of_context
```

4. Write a checkpoint (*dcp file*) to be used during the generation of the static system (First Multiplier.dcp then Adder.dcp)

```
write_checkpoint -force PATH2TUTORIAL/Adder.dcp
```

IMPLEMENTATION OF THE STATIC SYSTEM

1. Create a new Vivado project. Enable *Partial Reconfiguration Flow* for the project:

```
set_property PR_FLOW 1 [current_project]
```

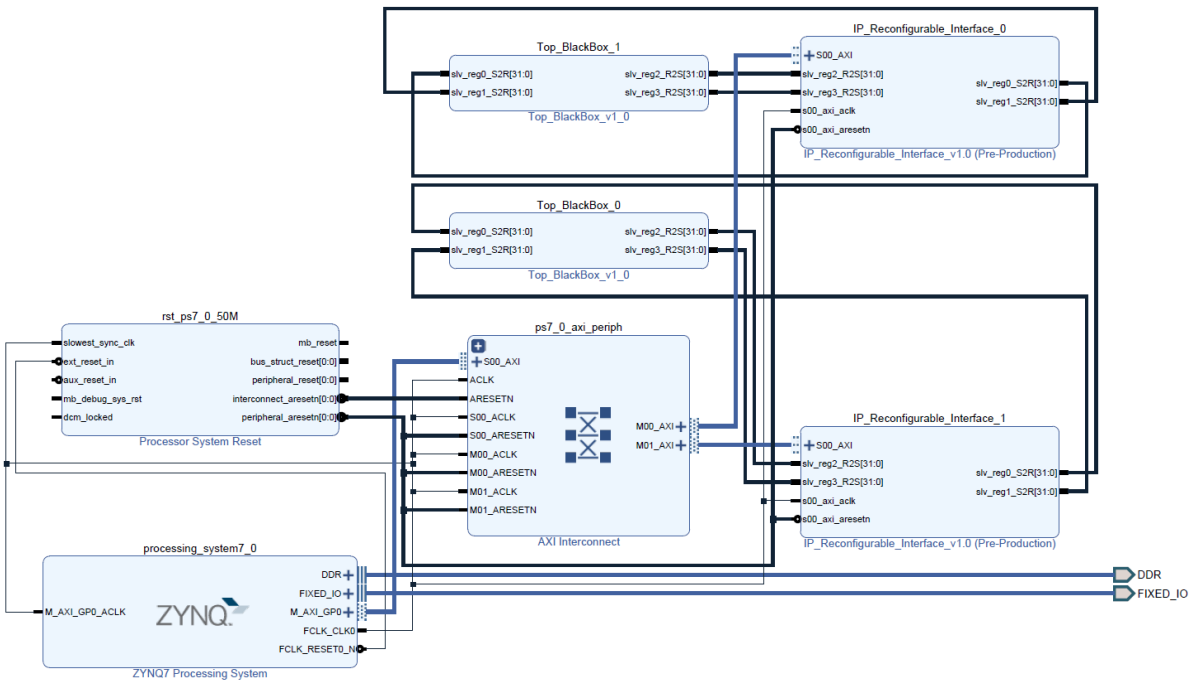
2. In Project Manager, Settings:
    - *Select VHDL as the target language.*
    - Add in IP repository the path to the *ip_repo* folder. In this repository two IPs are provided: the *IP_Reconfigurable_Interface* and the *Black_Box*, a dummy instance with the same interface as the reconfigurable modules to be implemented in the system.
3. Create a Block Design. Instantiate a *ZYNQ Processing System* IP, two *IP_Reconfigurable Interface* and two *Top_BlackBox* Instances in the design.

    Apply *Pynq Presets* in the Processing System. Enable the *M_AXI_GP0* Interface

    Run *Connection Automation* and connect manually the *Reconfigurable Interface* IPs to the *Black Box* IPs, as shown in the diagram.

    Create the *HDL Wrapper* for the block design.

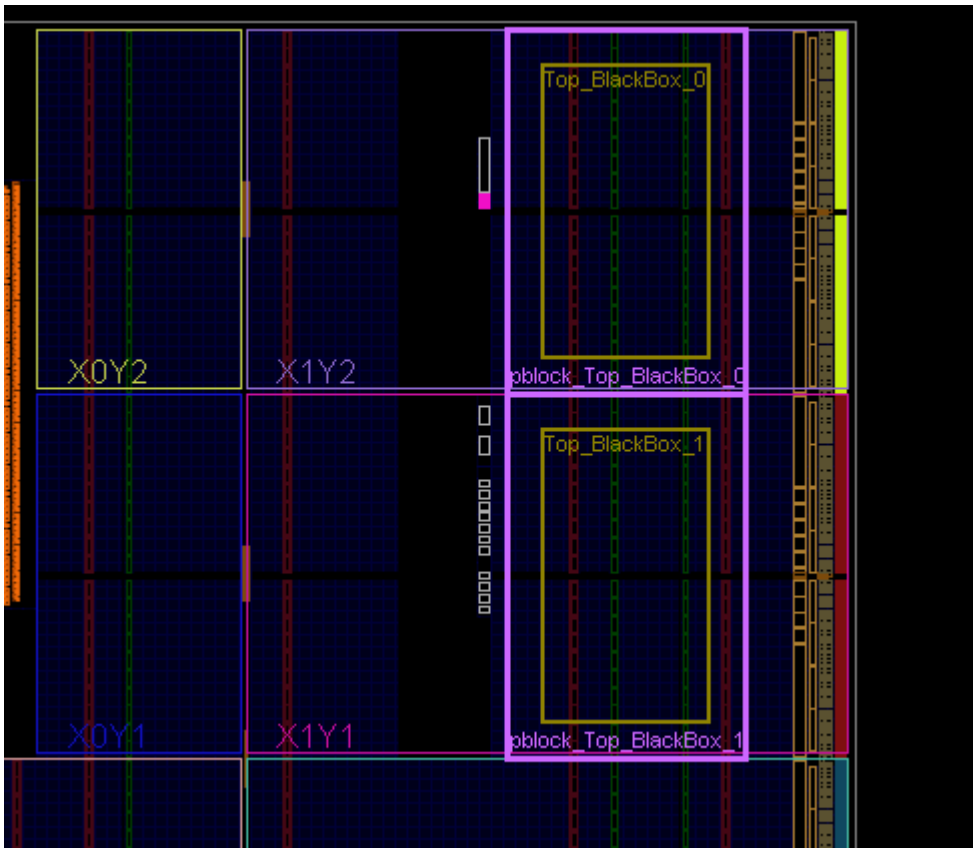
    Generate Output Products, in *Out of context per IP*



4. Run Synthesis and *Open Synthesized Design*
5. Set partitions of the design as reconfigurable (for the *Black Boxes* that will correspond with the IPs to be reconfigured)

```
set_property HD.RECONFIGURABLE TRUE [get_cells design_1_i/Top_BlackBox_0]
set_property HD.RECONFIGURABLE TRUE [get_cells design_1_i/Top_BlackBox_1]
```

6. Change to the *Floorplanning* Layout. Draw two pBlocks manually (Using *Draw Pblock*) according to the design rools for reconfigurable systems. Assign the *Top_BlackBox_0* and *Top_BlackBox_1* netlists to the pBlocks 1 and 2 respectivetly. To do so, select the netlists, right-click on it, Floorplanning, Assign to Pblock... A possible layout is shown in the next image.



7. Set the following properties to the pBLOCKs:

```
set_property RESET_AFTER_RECONFIG 1 [get_pblocks pblock_Top_BlackBox_0]
set_property SNAPPING_MODE ON [get_pblocks pblock_Top_BlackBox_0]
set_property RESET_AFTER_RECONFIG 1 [get_pblocks pblock_Top_BlackBox_1]
set_property SNAPPING_MODE ON [get_pblocks pblock_Top_BlackBox_1]
```

Then, for each Reconfigurable Module (Checkpoint) to be instantiated in each Reconfigurable Region of the design, repeat the following steps:

8. Set each Pblock as a Black box

```
update_design -cells [get_cells design_1_i/Top_BlackBox_0] -black_box
update_design -cells [get_cells design_1_i/Top_BlackBox_1] -black_box
```

9. Assign the checkpoint (Netlist) of the Reconfigurable Module to both *Black_Boxes* (First Multiplier.dcp then Adder.dcp)

```
read_checkpoint -cell design_1_i/Top_BlackBox_0 PATH2TUTORIAL/Multiplier.dcp
read_checkpoint -cell design_1_i/Top_BlackBox_1 PATH2TUTORIAL/Multiplier.dcp
```

10. Implement the design

```
opt_design
place_design
route_design
```

11. Generate full bitstreams and store checkpoints (partial bitstreams are also generated)

```
write_bitstream -force PATH2TUTORIAL/StaticMult.bit
write_checkpoint -force PATH2TUTORIAL/StaticMult.dcp
```

Now, steps 8 to 11 are repeated for the second set of modules:

8. Set each Pblock as a Black box

```
update_design -cells [get_cells design_1_i/Top_BlackBox_0] -black_box
update_design -cells [get_cells design_1_i/Top_BlackBox_1] -black_box
```

8. Preserve Static Routing. This step is not done for the first set of modules, only for the subsequent implementations to force the tool to keep the static routing decided in the first iteration:

```
lock_design -level routing
```

9. Assign the checkpoint (Netlist) of the Reconfigurable Module to both *Black_Boxes* (First Multiplier.dcp then Adder.dcp)

```
read_checkpoint -cell design_1_i/Top_BlackBox_0 PATH2TUTORIAL/Adder.dcp
read_checkpoint -cell design_1_i/Top_BlackBox_1 PATH2TUTORIAL/Adder.dcp
```

10. Implement the design

```
opt_design
place_design
route_design
```

11. Generate full bitstreams and store checkpoints (partial bitstreams are also generated)

```
write_bitstream -force PATH2TUTORIAL/StaticAdd.bit
write_checkpoint -force PATH2TUTORIAL/StaticAdd.dcp
```

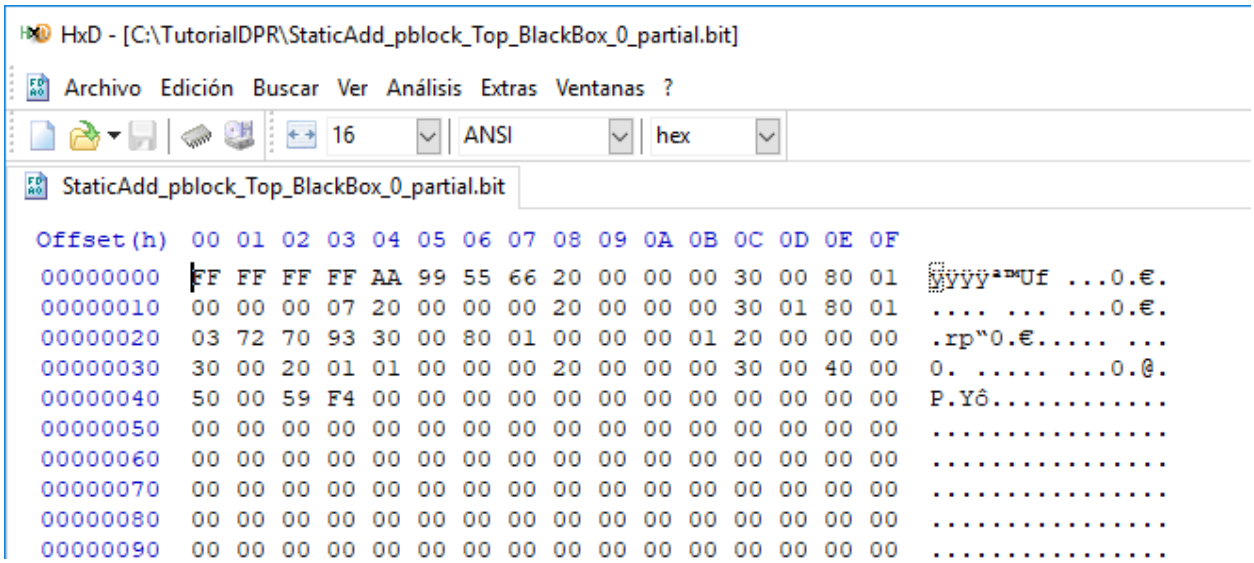
12. Verify the design (it may cause the shutdown of Vivado) and save the implementation constraints in the XDC file

```
pr_verify PATH2TUTORIAL/StaticMult.dcp PATH2TUTORIAL/StaticAdd.dcp
save_constraints
```

IMPLEMENTING THE SOFTWARE APPLICATION

1. Export Hardware to SDK, without including bitstream. Launch SDK
2. Create a new Application Project, Empty. Import the source file *PCAP_Reconfigure.c*
3. Modify the partial bitstream files with an hex editor (HxD) to remove the header. We also need to modify the sizes included by default in the *PCAP_Reconfigure.c* file depending on the size of the partial bitstreams.

   Add the math library in the options for the *gcc linker*.



4. Opent the terminal in SDK
5. Download system for debugging. Select the full bitstream Dowload partial bitstreams to memory. In application, Advanced Options, select the partial bitstream files to be downloaded in the following addresses:

File	Address
C:\TutorialDPR\StaticAdd_pblock_Top_BlackBox_0_partial.bit	0x10000000
C:\TutorialDPR\StaticAdd_pblock_Top_BlackBox_1_partial.bit	0x14000000
C:\TutorialDPR\StaticMult_pblock_Top_BlackBox_0_partial.bit	0x18000000
C:\TutorialDPR\StaticMult_pblock_Top_BlackBox_1_partial.bit	0x1c000000

Further information can be found in:

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug909-vivado-partial-reconfiguration.pdf