TP: Dynamic Partial Reconfiguration [1]

Goals :

-Use the VIVADO environment and SDK tool from Xilinx
-Understand the steps to generate partial reconfiguration bistreams
-Discover the project and the script modes to realize the partial reconfiguration design flow

- Introduction

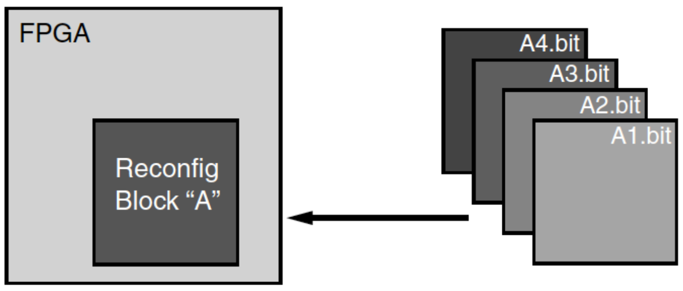
During this lab, you will re-use the integrated design environment from Xilinx, called VIVADO. This environment gathers a lot of tools for FPGA design, analysis and programming. You will also be able to understand the partial reconfiguration (PR) mechanisms and how to make it work on FPGA devices. At the end of this session, you will have acquired all the prerequisites to develop your own design .

The lab is organized as follows :
First, general explanations about partial reconfiguration and required sources files are given. Then, during the first exercise, you will understand the different partial reconfiguration mechanisms and apply them during a concrete example. You will also be able to use the Vivado PR flow using TCL scripts.

General explanations:

- Partial reconfiguration

One of the main advantage of PR is to increase design flexibility, allowing to change a part of the FPGA design while the circuit is still running. This is realized by loading a partial configuration file (partial bit file) in the FPGA without disturbing the rest of the design.



Basics of Partial Reconfiguration [1]

Such technique allows to increase the flexibility by changing block of algorithms (hardware reconfiguration) during the execution, to improve design reliability and so on.

[1] This document is based on two Vivado tutorials from Xilinx [1,2].

- Partial reconfiguration main steps [1]:
  1) Synthesize of the static and Reconfigurable modules separately (in out-of-context mode)
  2) Create physical constraints (pBlocks) to define the reconfigurable regions
  3) Set the HD.RECONFIGURABLE property on each Reconfigurable Partition
  4) Implement a complete design (static+ one or more reconfigurable modules)
  5) Save a design checkpoint for the fully routed design
  6) Remove reconfigurable modules from the design and save a static-only design checkpoint
  7) Lock the static placement and routing
  8) For every reconfigurable modules
     a. Add new reconfigurable modules to the static design and implement this new configuration
     b. Saving a checkpont for the full routed design
  9) Run a verification utility (pr_verify) on all configurations
  10) Create bistreams for each configuration

• Objectives of the lab

During this lab, we will have 2 reconfigurable modules (RM):
-A counter
-A shifter

From this, we will define two configurations per RM in a way to partially reconfigure the functionality of the counter and the shifter. For each module, we will choose to reconfigure the counter as an up counter or a down counter and the shifter as a right or left shifter.

• Project Mode vs Script Mode

The partial reconfiguration flow can be realized in two different ways but leading to the same result. First, we will realize the most common flow using the project mode based on VIVADO GUI. In a second time, we will realize the same flow but using only TCL (Tool Command Lines) scripts, providing an efficient way to generate the bitstream when all sources and files are already defined.

Exercise 1 : Partial Reconfiguration

During the project mode, you will realize the different steps in order to generate all the required files for the partial reconfiguration.

1) **Before Launching Vivado**, create the appropriate folders in your local directory. Please respect the following folder hierarchy in your directory by creating the missing ones:

   -Project_directory
   -DCP
   -Sources -xdc
   -hdl
   -Bitstream

2) Retrieve the sources files from the archive *TP_PR_Sources.zip* we provide at this website : http://perso-etis.ensea.fr/lorandel/M2_ESA_SoC.php

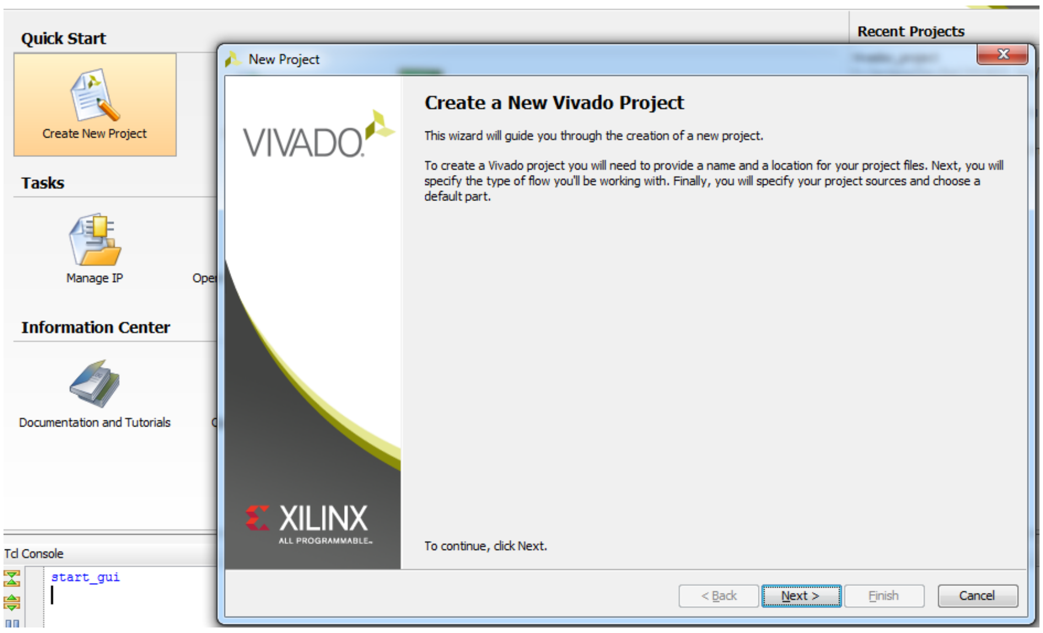
3) Launch **vivado** in a terminal using:

```
source /usr/lsa/apps/Xilinx/Vivado/2017.4/settings64.sh
```

   and then

```
vivado
```

4) Then **Create New Project > Next**



Create a New Vivado Project

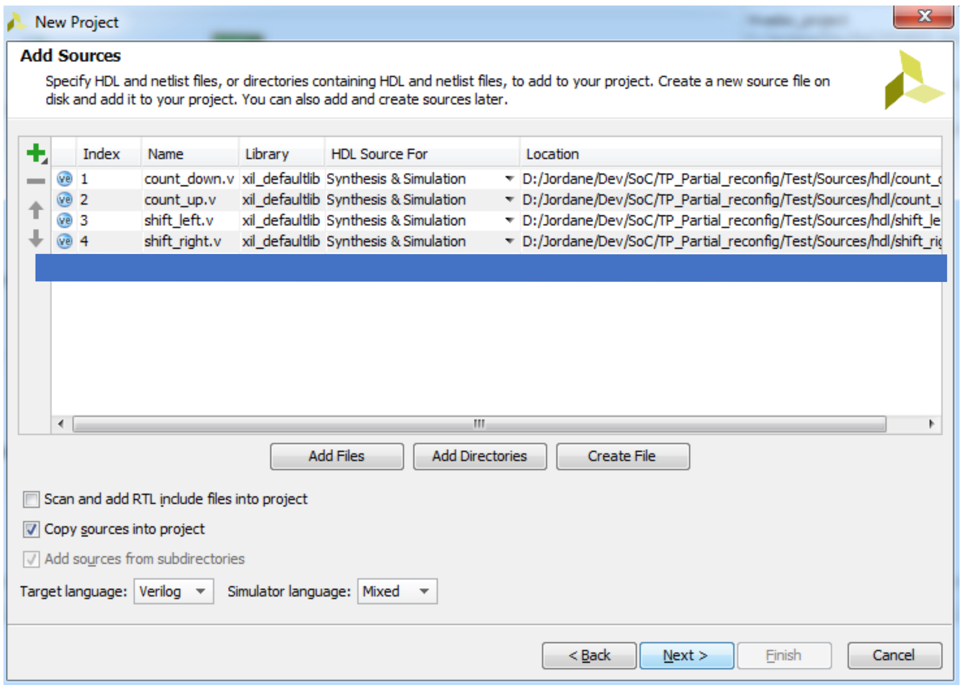
This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

5) Define a project name and the project location (in your local directory where you create the previous folder[2])
6) In page Add source files, add the HDL files we provided, excepted Top.v.

[2] Avoid any special character in the path to the project folder as well as in the name of the directory.


New Project

Add Sources

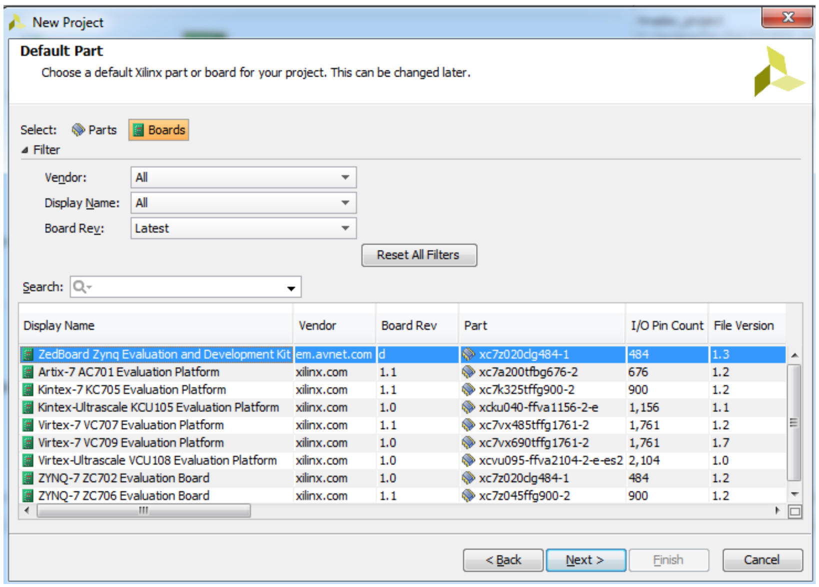
Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Index	Name	Library	HDL Source For	Location
1	count_down.v	xil_defaultlib	Synthesis & Simulation	D:/Jordane/Dev/SoC/TP_Partial_reconfig/Test/Sources/hdl/count_
2	count_up.v	xil_defaultlib	Synthesis & Simulation	D:/Jordane/Dev/SoC/TP_Partial_reconfig/Test/Sources/hdl/count_
3	shift_left.v	xil_defaultlib	Synthesis & Simulation	D:/Jordane/Dev/SoC/TP_Partial_reconfig/Test/Sources/hdl/shift_le
4	shift_right.v	xil_defaultlib	Synthesis & Simulation	D:/Jordane/Dev/SoC/TP_Partial_reconfig/Test/Sources/hdl/shift_ri

☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

Target language: Verilog   Simulator language: Mixed

7) In page Add constraint, add the .xdc files that we provided
8) Select the zedboard as target platform and **FINISH**


New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts [✓ Boards]

Filter

Vendor: All
Display Name: All
Board Rev: Latest

Search:

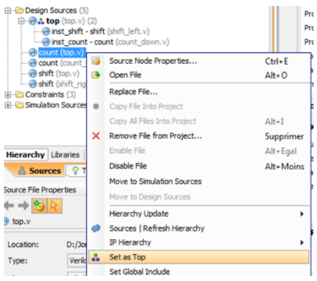
Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676	1.2
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.2
Kintex-Ultrascale KCU105 Evaluation Platform	xilinx.com	1.0	xcku040-ffva1156-2-e	1,156	1.1
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.2
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.7
Virtex-Ultrascale VCU108 Evaluation Platform	xilinx.com	1.0	xcvu095-ffva2104-2-e-es2	2,104	1.0
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.2
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.2

I) SYNTHESIS

We have to synthesize every reconfigurable module (RM) individually in a specific mode called **out-of-context**. To this purpose, we will create design checkpoint (*.dcp file) file for every RM. This file contains the corresponding netlist of the considered RP after the synthesis.

A) For every reconfigurable module :

-Set as Top_level



-In the TCL Console :

synth_design –top XXX –part xc7z020clg484-1 –mode out_of_context

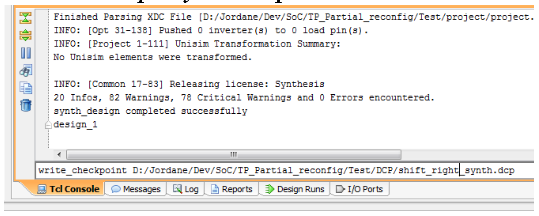
where XXX is the name of the module (count or shift).

-Create a DCP (Design CheckPoint) for each partial module directly after the synthesis using this tcl command :

write_checkpoint FULLPATH/DCP/XXX_synth.dcp

where XXX is the name of count_up/count_down/shift_right/shift_left.

Note: you will need to add the full path to the final directory before the dcp. For example: C:/Users/…/…/count_up_synth.dcp



```
Finished Parsing XDC File [D:/Jordane/Dev/SoC/TP_Partial_reconfig/Test/project/project.
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

INFO: [Common 17-83] Releasing license: Synthesis
20 Infos, 82 Warnings, 78 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
design_1

write_checkpoint D:/Jordane/Dev/SoC/TP_Partial_reconfig/Test/DCP/shift_right_synth.dcp
```

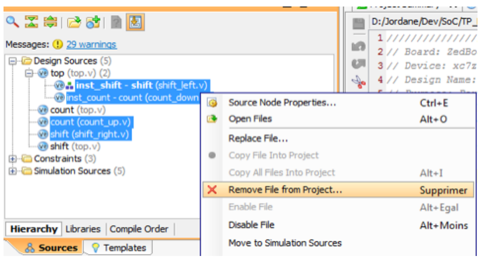
-Close the design

close_design

END for every module

Now we will have multiple design checkpoints for the corresponding reconfigurable modules in the DCP folder. We can now synthesize the main project with no definition for RM.

-Remove all the source files (*shift_left.v*, *shift_right.v*, *count_up.v*, *count_down.v*), excepted the top.v file



A) Synthesis the main file top.v

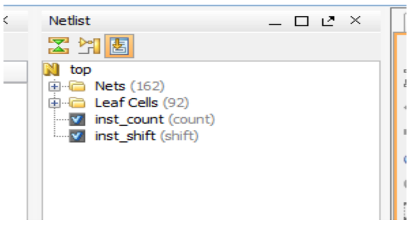
-Set the top.v file as **top level**.

-click on run synthesis in the Flow Navigator:

launch_runs synth_1

or use the tcl command : **synth_design –top top –part xc7z020clg484-1**

B) After the synthesis of the main file, **Open Synthesized Design**
As indicated in the figure on the right, you can notice that the partial RM called inst_count and inst_shift are defined as blackbox.

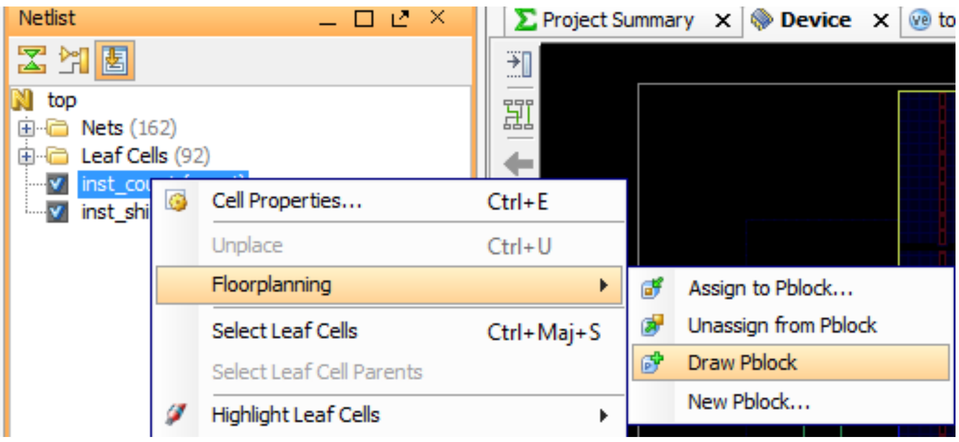


-save the top design

```
write_checkpoint FULLPATH/DCP/top_synth.dcp
```

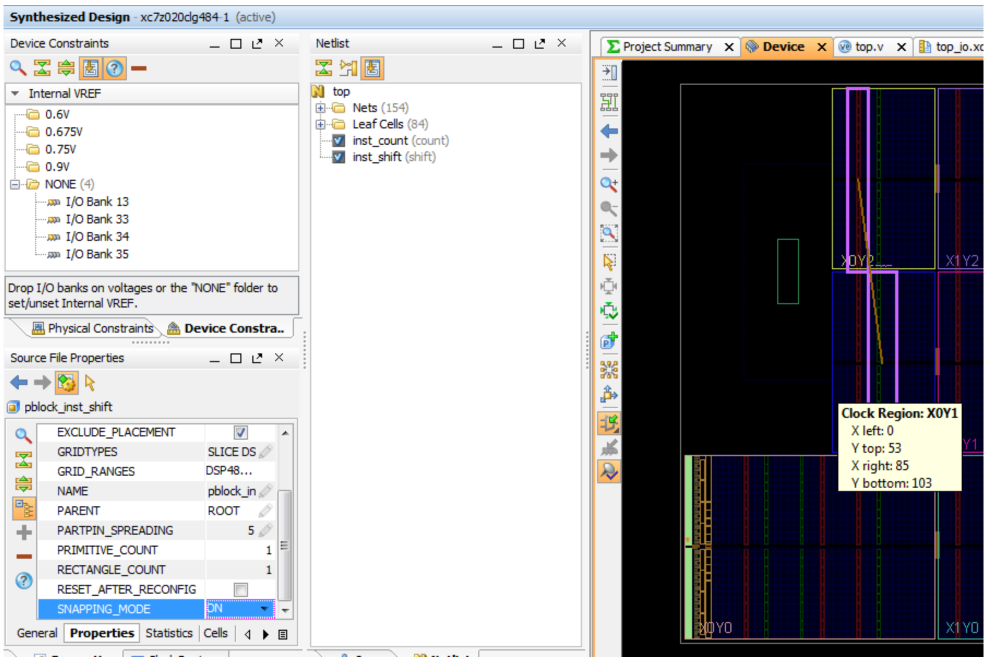
Defining the reconfigurable areas :

C) Drawing Pblock for the 2 modules
-Right click on 1 of the 2 modules **< Floorplanning < draw pblock**
Note: Perform the same floorplanning as shown below:



-Let the default name for the 2 pblocks
Note: Have a look at the physical ressource estimates

-Specify properties SNAPPING_MODE to ON for the 2 pblocks, and check that the RESET_AFTER_RECONFIG is enabled.



Drop I/O banks on voltages or the "NONE" folder to set/unset Internal VREF.

pblock_inst_shift

EXCLUDE_PLACEMENT	✓
GRIDTYPES	SLICE DS
GRID_RANGES	DSP48...
NAME	pblock_in
PARENT	ROOT
PARTPIN_SPREADING	5
PRIMITIVE_COUNT	1
RECTANGLE_COUNT	1
RESET_AFTER_RECONFIG	☐
SNAPPING_MODE	ON

Clock Region: X0Y1
X left: 0
Y top: 53
X right: 85
Y bottom: 103

-Save the design and the modified constraint file as well

Defining the first configuration :

D) Allocate Design Checkpoints (DCP) to blackboxes

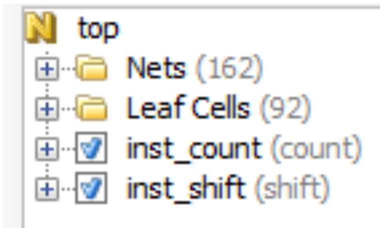
We can load the configuration of one of the PR modules to the blackbox by the following tcl command :

read_checkpoint -cell rp1 FULLPATH/DCP/rp1_synth.dcp

Note : you will need to add the full path before the dcp for example :
C:/Users/lorandel/Desktop/rp1_synth.dcp
RP1 corresponds to inst_count or inst_shift

-Load the inst_count with count_up.dcp
-Load the inst_shift with the shift_left.dcp

You can note that the 2 blackboxes have been filled with the synthesized netlist as illustrated:



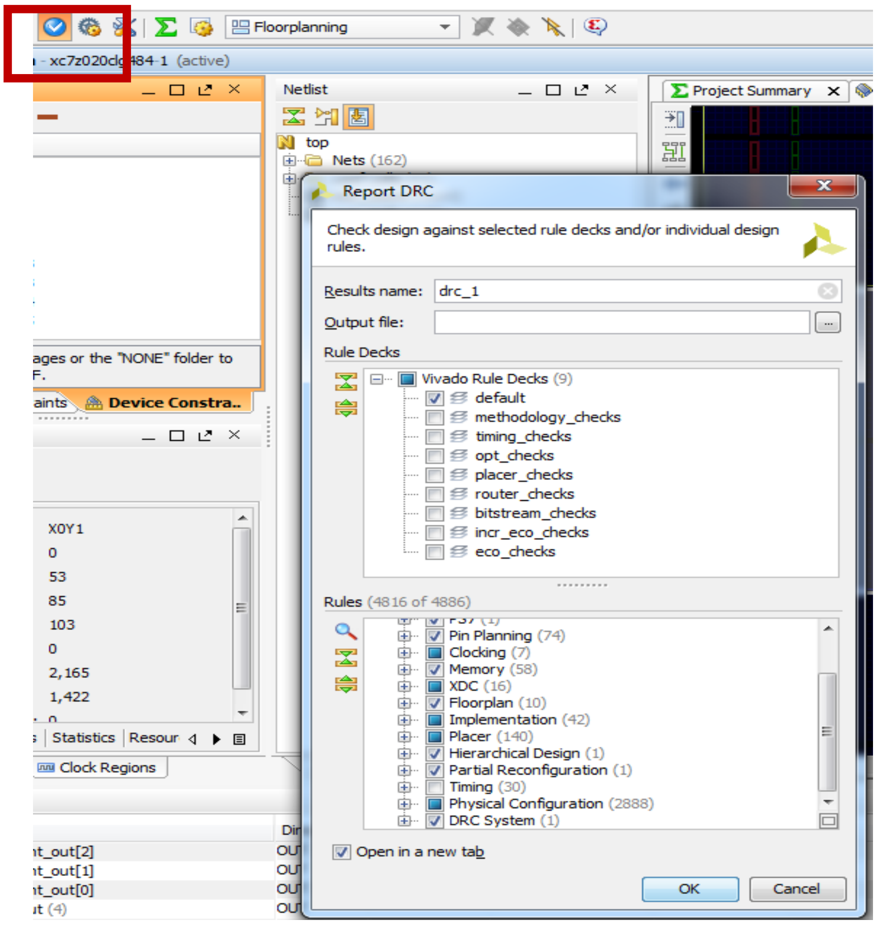
-Define the pblocks as reconfigurable using

set_property HD.RECONFIGURABLE true [get_cells rp1]

With rp1 = inst_count or inst_shift.

-Save the design
-Check if the pblock are valid using **report DRC**

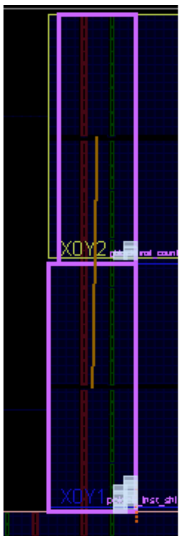
Normally, the analysis will deliver no DRC violations. If you have one or more errors, try to find them ☺ ! Re-run the drc analysis until you obtain any error. Finally, if you still have some errors, call the teacher!



E) Generation of the 1st configuration
-Use these tcl commands to optimize, place and route the first configuration:

```
opt_design
place_design
route_design
```

Now the first configuration is fully placed and routed. You can see in the floorplanning view, the cells that are used by the design in every RM:



F) Save the results

Save the full design checkpoint and create report files using these commands:

```
write_checkpoint –force FULLPATH/DCP/Config_shift_left_count_up_route.dcp
write_checkpoint -cell inst_count FULLPATH/DCP/count_up_route.dcp
write_checkpoint -cell inst_shift FULLPATH/DCP/shift_left_route.dcp
```

At this point, we have created a fully implemented the first configuration from which we can generate full and partial bitstreams. The static portion of this configuration must be used for all subsequent configurations and has to be isolated. For generating the static design, the current reconfigurable modules must be removed.

G) Generate the static configuration

Clear out reconfigurable module logic by issuing the following commands:

```
update_design -cell inst_shift -black_box
update_design -cell inst_count -black_box
```

We now have to write a design checkpoint of the remaining static-only design.

```
lock_design –level routing
write_checkpoint -force FULLPATH/DCP/static_route_design.dcp
```

This static-only checkpoint would be used for any future configurations, but in this lab, we simply keep this design open in memory.

-You can now close the design

H) Generation of other configurations

The static design is now established, and we will use it as context for implementing further reconfigurable modules.

```
open_checkpoint FULLPATH/DCP/static_route_design.dcp
```

-We can now add other synthesized netlist to the RMs

```
read_checkpoint -cell inst_shift FULLPATH/DCP/shift_right_synth.dcp
read_checkpoint -cell inst_count FULLPATH/DCP/count_down_synth.dcp
```

-Use these tcl commands to generate the second configuration:

```
opt_design
place_design
route_design
```

-Save the results

```
write_checkpoint FULLPATH/DCP/Config_shift_right_count_down_route_design.dcp
write_checkpoint –cell inst_shift FULLPATH/DCP/Shift_right_route_design.dcp
write_checkpoint –cell inst_count FULLPATH/DCP/Count_down_route_design.dcp
```

I) Generate the bitstreams

-Verify configuration :

Before generating bitstreams, verify all configurations to ensure that the static portion of each configuration match identically, so the resulting bitstreams are safe to use in silicon. The PR Verify feature examines the complete static design up to and including the partition pins, confirming that they are identical. Placement and routing within the reconfigurable modules is not checked, as different module results are expected here.

-Run the pr_verify command from the Tcl Console:

pr_verify FULLPATH/DCP/Config_shift_left_count_up_route.dcp FULLPATH/DCP/ Config_shift_right_count_down_route.dcp –file FULLPATH/config1_config2.log

If successful, this command returns the following message. Have a look to the log file if necessary.
INFO: [Vivado 12-3253] PR_VERIFY: check points ../DCP/Config_shift_right_count_up_route.dcp and ../DCP/Config_shift_left_count_down_route.dcp ***are compatible***
By default, only the first mismatch (if any) is reported. To see all mismatches, use the -full_check option.

Now, the configurations have been verified, we can generate bitstreams and use them to target the zedboard.

-Open the first configuration
open_checkpoint FULLPATH/DCP/Config_shift_right_count_down_route_design.dcp

- Generate full and partial bitstreams for this design. Be sure to keep the bit files in a unique directory related to the full design checkpoint from which they were created.
write_bitstream FULLPATH/Bitstreams/Config_shift_right_count_down
close_project

At this point, we can notice that the three bitstreams were created in the bitstream directory:

- **Config_shift_right_count_down.bit** This is the power-up, full design bitstream.
- **Config_shift_right_count_down_pblock_inst_count_partial.bit** This is the partial bit file for the count downt module.
- **Config_shift_right_count_down_pblock_inst_shift_partial.**This is the partial bit file for the shift_left module.

You can perform the same steps for the second config
-Open the first configuration
open_checkpoint FULLPATH/DCP/Config_shift_left_count_up_route_design.dcp
write_bitstream FULLPATH/Bitstreams/Config_shift_left_count_up
close_project

BLANK CONFIGURATION

-Generate a full bitstream with black boxes, plus blanking bitstreams for the reconfigurable modules. Blanking bitstreams can be used to "erase" an existing configuration to reduce power consumption.

```
open_checkpoint FULLPATH /DCP/static_route_design.dcp
write_bitstream FULLPATH/Bitstreams/Blank
close_project
```

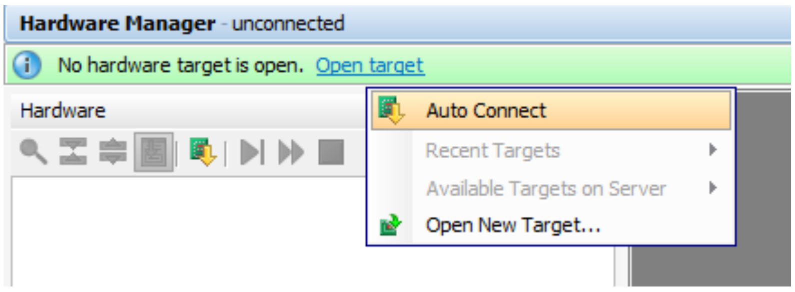
The base configuration bitstream will have no logic for either reconfigurable partition.

J) Partially reconfigure the FPGA

☺

HERE WE ARE !!!

-Connect the zedboard to the computer and power on the board.
-From VIVADO, **select Flow > Open Hardware Manager**
-Select **Open a new hardware target** on the green banner. Follow the steps in the wizard to establish communication with the board.


Hardware Manager - unconnected
No hardware target is open. Open target
Hardware
Auto Connect
Recent Targets
Available Targets on Server
Open New Target...

-Select **Program device** on the green banner and select the xc7z FPGA device. Navigate to the Bitstreams folder to select **Config_shift_right_count_down.bit**, then click **OK** to program the device.

You should now see the bank of GPIO LEDs performing two tasks. Four LEDs are performing a counting-down function (MSB is on the left), and the other four are shifting to the right. Note the amount of time it took to configure the full device.

At this point, you can partially reconfigure the active device with any of the partial bitstreams that you have created:

- Select **Program device** on the green banner again. Navigate to the Bitstreams folder to select **Config_shift_left_count_up_pblock_inst_shift_partial.bit**, then click **OK** to program the device.

The shift portion of the LEDs has changed direction, but the counter kept counting down, unaffected by the reconfiguration. Note the much shorter configuration time.

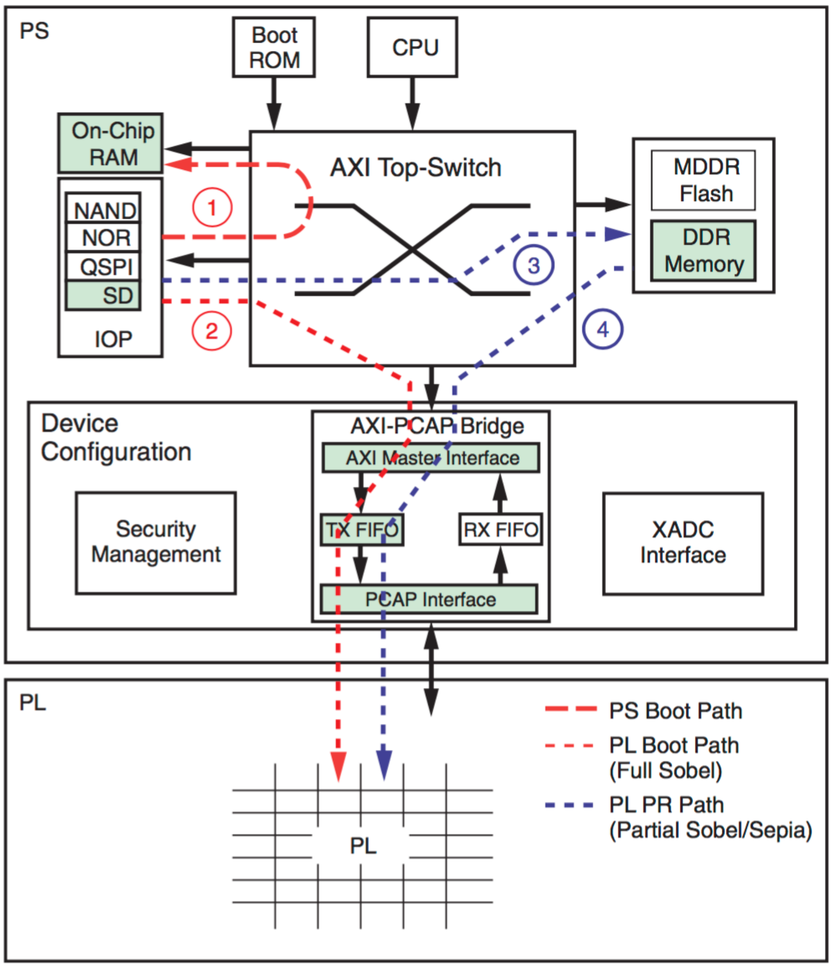
-Select **Program device** on the green banner again. Navigate to the Bitstreams folder to select **Config_shift_left_count_up_pblock_inst_count_partial.bit**, then click **OK** to program the device.

The counter is now counting up, and the shifting LEDs were unaffected by the reconfiguration. This process can be repeated with the other bit file to return to the original configuration, or with the blanking partial bit files to stop activity on the LEDs (they will stay on).

Exercise: Dynamic Partial Reconfiguration using PCAP

We now want to avoid the loading of partial bitstreams from the JTAG. To this purpose, we will use the (PCAP) Processor Configuration Access Port, as detailed in the course, and the DevCFG controller to configure the FPGA directly from an application on the processor.

How these elements are linked together:



The Device Configuration interface (DevC), illustrated in the figure above, has three main blocks: an AXI-PCAP bridge (center) for interfacing the PL configuration logic, device security management (left), and an XADC interface (right). In our case, only the AXI-PCAP bridge is of interest.

The AXI-PCAP bridge converts 32-bit AXI formatted data to the 32-bit PCAP protocol and vice versa. A transmit and receive FIFO buffer data between the AXI and the PCAP interface.

A DMA engine moves data between the FIFOs and a memory device, typically the OCM (on-chip memory), the DDR memory, or one of the peripheral memories. The 32-bit PCAP interface is clocked at 100 MHz and supports 400 MB/s download throughput for non-secure PL configuration and 100 MB/s for secure PL configuration where data is sent only every 4th clock cycle. To transfer data across the PCAP interface a DevC driver function needs to be called. The driver will take care of setting the correct PCAP mode and initiating the DMA transfer. The function call will only return after both the AXI and the PCAP transfers are complete

Required materials:

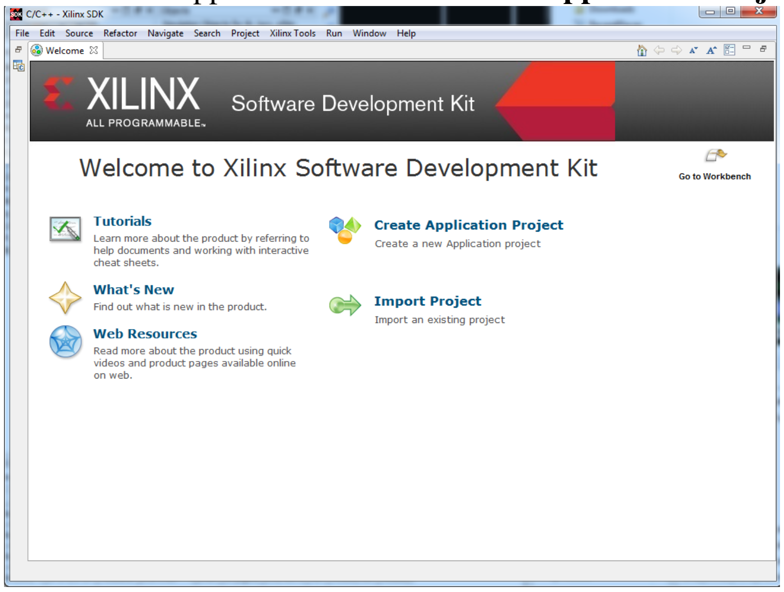
- An application template of the source code is available. Call the teacher.
- Place the files into a folder in a local repository (Pay attention, avoid any special characters in the path!!!)
- Bitstreams must be converted into Binary files to be download to the PL through PCAP (**See Appendice 1) ;** If needed, the teacher can give you access to binary files
- Binary files must be copied into the SD card
- Replace the SD card into the zedboard and power on the board

❑ Application program

In the terminal, launch SDK and select the folder, where you placed the source code, as workspace.

```
source /usr/lsa/apps/Xilinx/SDK/2017.4/settings64.sh
 xsdk
```

1) Create a new application: **File -> New -> Application Project**



Welcome to Xilinx Software Development Kit

Tutorials
Learn more about the product by referring to help documents and working with interactive cheat sheets.

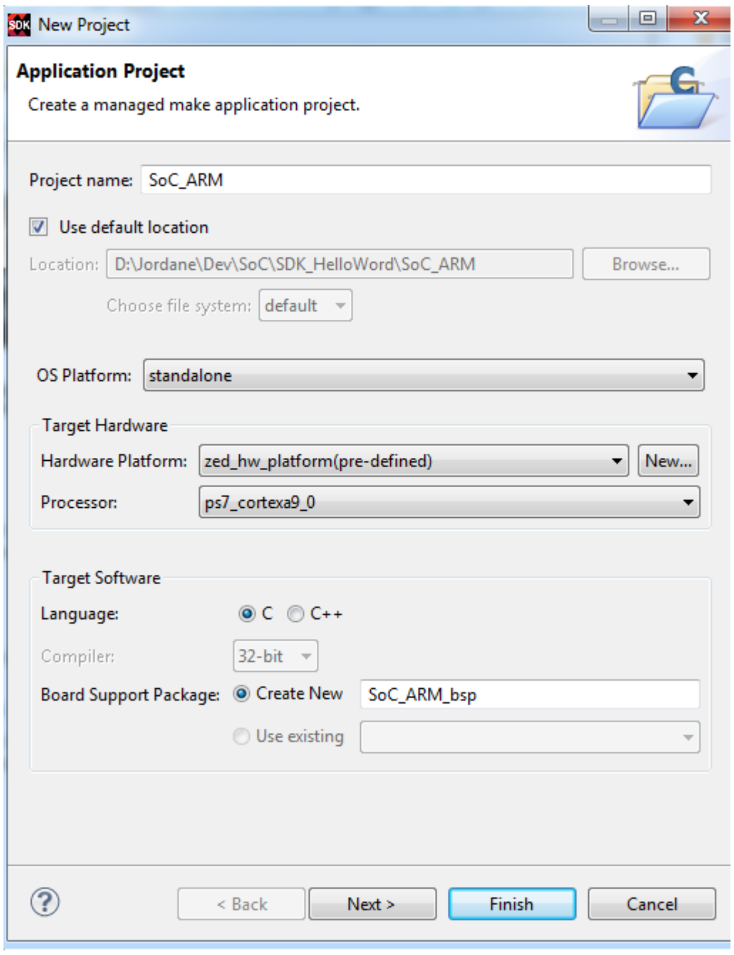
What's New
Find out what is new in the product.

Web Resources
Read more about the product using quick videos and product pages available online on web.

Create Application Project
Create a new Application project

Import Project
Import an existing project

2) Configure your project as indicated in the figure below :


New Project

Application Project

Create a managed make application project.

Project name: SoC_ARM

[✓] Use default location

Location: D:\Jordane\Dev\SoC\SDK_HelloWord\SoC_ARM

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: zed_hw_platform(pre-defined)

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

Compiler: 32-bit

Board Support Package: [✓ Create New] SoC_ARM_bsp

Use existing

3) Then **Next** and select **Empty Application** Template

4) The easiest way to add the source file .c in the project is to directly copy the .c file in the project directory (Workspace_directory/project_name/src). Once done, under SDK, you only have to refresh (F5) the project explorer and check that the file was correctly detected and added to the project.

❑ Program Analysis

Q : According to you, using few words, explain the program functionality ?

Q : wht

```
#define PARTIAL_COUNT_UP_FILE_ADDR     0x01000000
#define PARTIAL_COUNT_UP_FILE_SIZE     0x771A*4
```

Q : What is the objective of the SD_TransferPartial() function ?

Q : What is the objective of the XDcfg_TransferBitfile() function ?

❑ Modification of the program

1. Modify the main_student.c program where it is indicated in order to be able to load all the binary files into the DDR memory and then, according to the number obtained from the user in the serial terminal, perform the dynamic partial reconfiguration of the reconfigurable module in the FPGA, as follow:

Case '1' => load partial bitstream of the down counter
Case '2' => load partial bitstream of the up counter
Case '3' => load partial bitstream of the shift right module
Case '4' => load partial bitstream of the shift left module

5) Debug your application : Righ click on the project, **Debug As / Launch on Hardware (System Debugger)**
6) **Click Yes** for the perspective switch.
7) Validate the functionality ;
8) If ok; Call the teacher ;)

Conclusion & Summary

In this tutorial, you:

- Synthesized a design and realize the partial reconfiguration implementation
- Created two configurations with common static results
- Implemented these two configurations, saving the static design to be used in each
- Created checkpoints for static and reconfigurable modules for later reuse
- Examined framesets and verified the two configurations
- Created full and partial bitstreams
- Configured and partially reconfigured an FPGA using JTAG and PCAP interface

During this practical work, you have realized an entire partial reconfiguration design flow using Vivado. The Vivado GUI project Mode with some TCL scripts was used.
You also learnt how partial reconfiguration can be performed <u>dynamically</u> on FPGA devices through the PCAP and what are the underlying mechanisms.

Bibliography

[1]Xilinx, '*Vivado Design Suite User Guide: Partial Reconfiguration*', User Guide UG909, v2015.4 November 2015,
https://www.xilinx.com/support/documentation/sw_manuals_j/xilinx2014_1/ug940-vivado-tutorial-embedded-design.pdf

[2]Xilinx, '*Vivado Design Suite Tutorial: Partial Reconfiguration*', User Guide UG947, v2013.3 October 2013,
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug947-vivado-partial-reconfiguration-tutorial.pdf

[3]Zedboard, '*ZedBoard (ZynqTM Evaluation and Development) Hardware User's Guide*', v2.2 jan. 2014,
http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

Training resources

Vivado partial reconfiguration video : https://www.xilinx.com/video/hardware/partial-reconfiguration-in-vivado.html

Appendice 1

- Convert Bitstreams into Binary file

1) Launch Vivado.
2) In the welcome page, in the TCL console, type the following TCL command:

For every bitstream :

```
write_cfg –format BIN –interface SMAPx32 –disablebitswap –loadbit “up 0x0
FULLPATH/Bitstream/filename.bit” FULLPATH/Bitstream/filename.bin
```

FULLPATH means the entire path to the folder where bitstreams are located e.g. /usr/…