

TP : System on Chip (SoC)¹

Goals :

- Discover the VIVADO environment and SDK tool from Xilinx
- Programming of the Software part of a SoC
- Control of hardware peripheral using software running on the ARM processor

• Introduction

During this lab, you will discover the integrated design environment from Xilinx, called VIVADO. This environment gathers a lot of tools for FPGA design, analysis and programming. To program the software part of a SoC, the SDK (Software Design Kit) tool will be used. At the end of this session, you will have acquired all the prerequisites to develop a complete SoC, including both software and hardware parts.

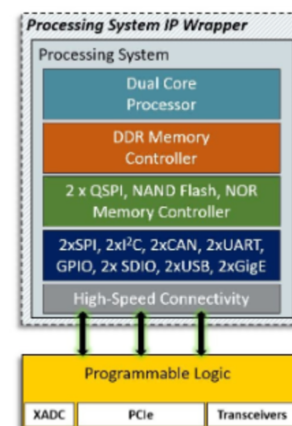
The work is organized as follows :

- Software programming of the ARM processor using SDK
- System Design under VIVADO to create your own FPGA-based system with a GPIO interface connected to leds. The SDK tool will be used for the software programming to directly manage the leds from the ARM processor.
- A last exercise that consists in adding your own peripheral (switches) to the zynq processor system, and manage them using software.

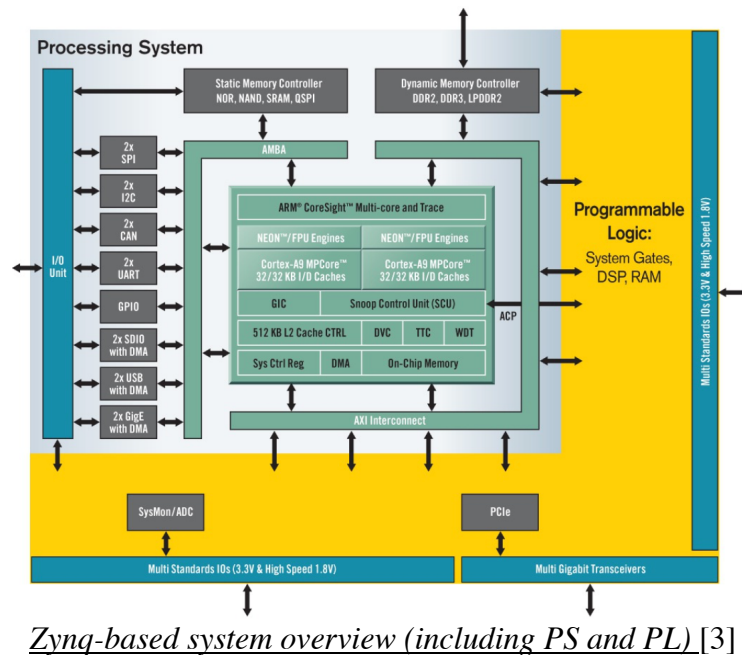
○ Zynq-Processing System [1]

Xilinx provides the Processing System IP Wrapper for the Zynq®-7000 to accelerate the design and its configuration. The Processing System IP is the software interface around the Zynq-7000 Processing System. The Zynq-7000 family consists of a system-on-chip (SoC) style integrated processing system (PS) and a Programmable Logic (PL) unit, providing an extensible and flexible SoC solution on a single die.

The Processing System IP Wrapper acts as a logic connection between the PS and the PL while assisting you to integrate custom and embedded IPs with the processing system using the Vivado IP integrator.



¹ This document is based on the Vivado Design Suite tutorial from Xilinx [2].



IMPORTANT : Connect the zedboard to the computer and power on the board before launching SDK ! Please reboot the computer with all the USB ports the board connected to it.

Exercice 1 : Programming the ARM processor

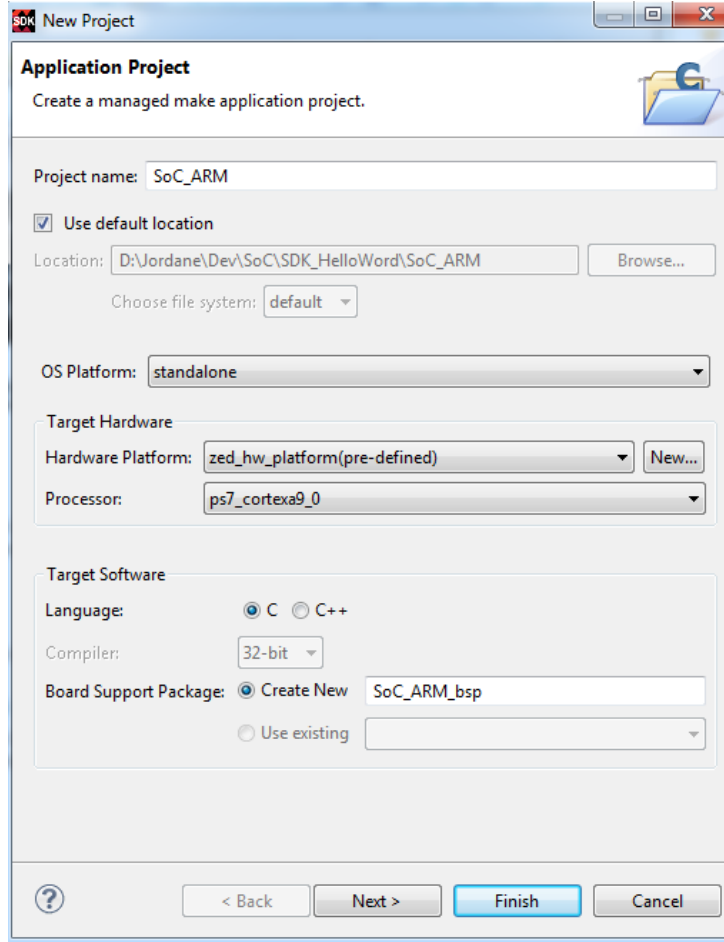
Programming the ARM processor can be easily realized using SDK tool from Xilinx. Follow the next steps :

- 1) Launch SDK with these commands

```
source /usr/lsa/apps/Xilinx/SDK/2017.4/settings64.sh
xsdk
```
- 2) Select a workspace in your local directory
- 3) Then **File -> New -> Application Project**



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



Application Project
Create a managed make application project.

Project name: SoC_ARM

☒ Use default location

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

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: zed_hw_platform(pre-defined) New...

Processor: ps7_cortexa9_0

Target Software

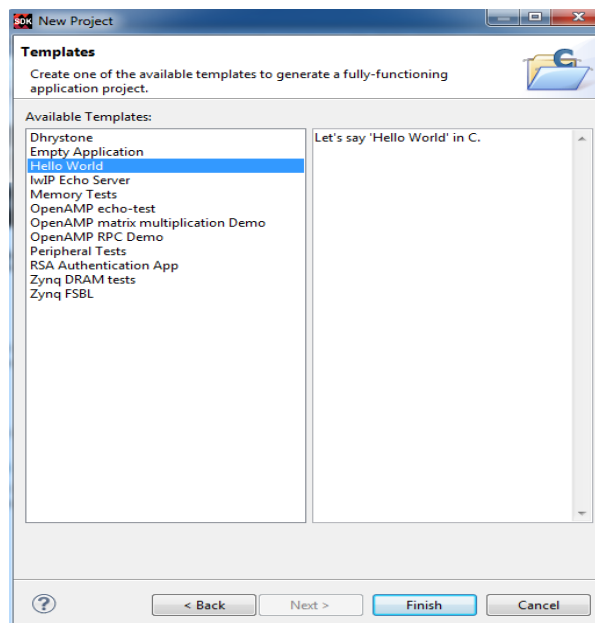
Language: ☒ C ☐ C++

Compiler: 32-bit

Board Support Package: ☒ Create New SoC_ARM_bsp Use existing

? < Back Next > Finish Cancel

4) Then **Next** and select **Hello Word Template**



Templates
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- Empty Application
- Hello World**
- IWiP Echo Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

Let's say 'Hello World' in C.

? < Back Next > Finish Cancel

5) Finally, click on **Finish** to generate the project

Now, the project is created and you can view the project hierarchy directly in the upper left window called Project Explorer.

Project description :

-SoC_Project

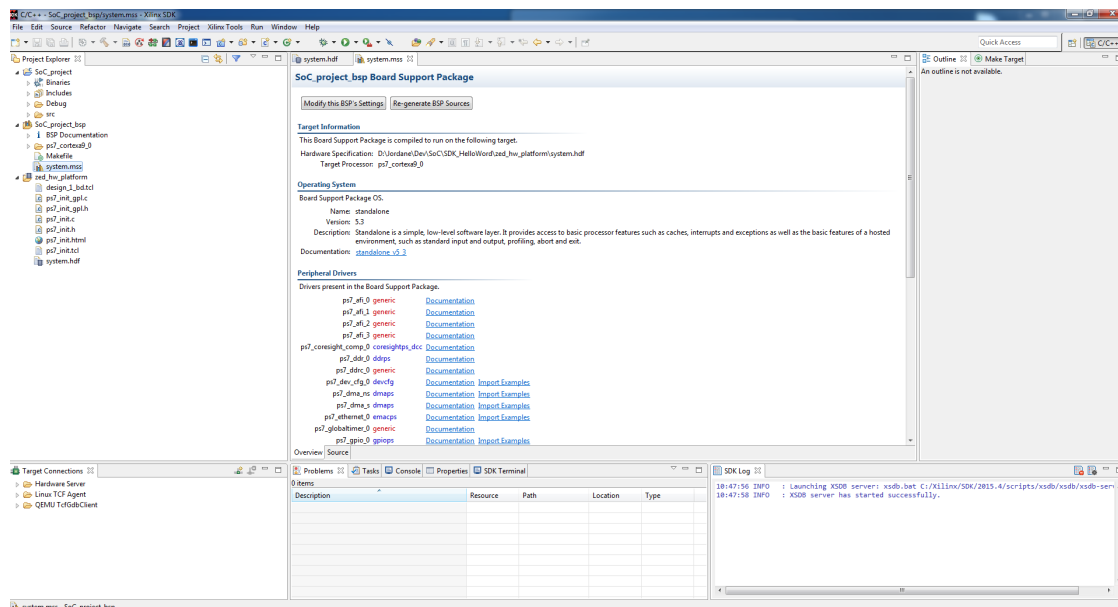
Folder of C source files and header files. It also contains binaries after the build of the project.

-SoC_Project_bsp

Contains the Board Support package with all drivers/API/documentation for the ARM processor.

-Zed_hw_platform

Contains initialisation files and additional information related to the target hardware platform i.e. Zedboard. Look at the **system.hdf** file.



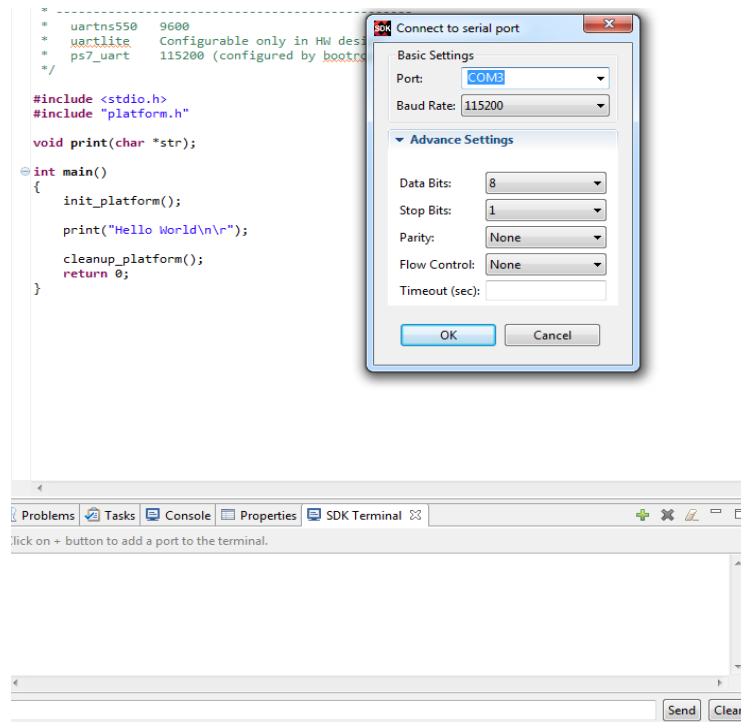
Q : What are the addresses indicated in the system.hdf file and to what do they correspond ?

-Building the project :

Project -> Build Project (if not done automatically)

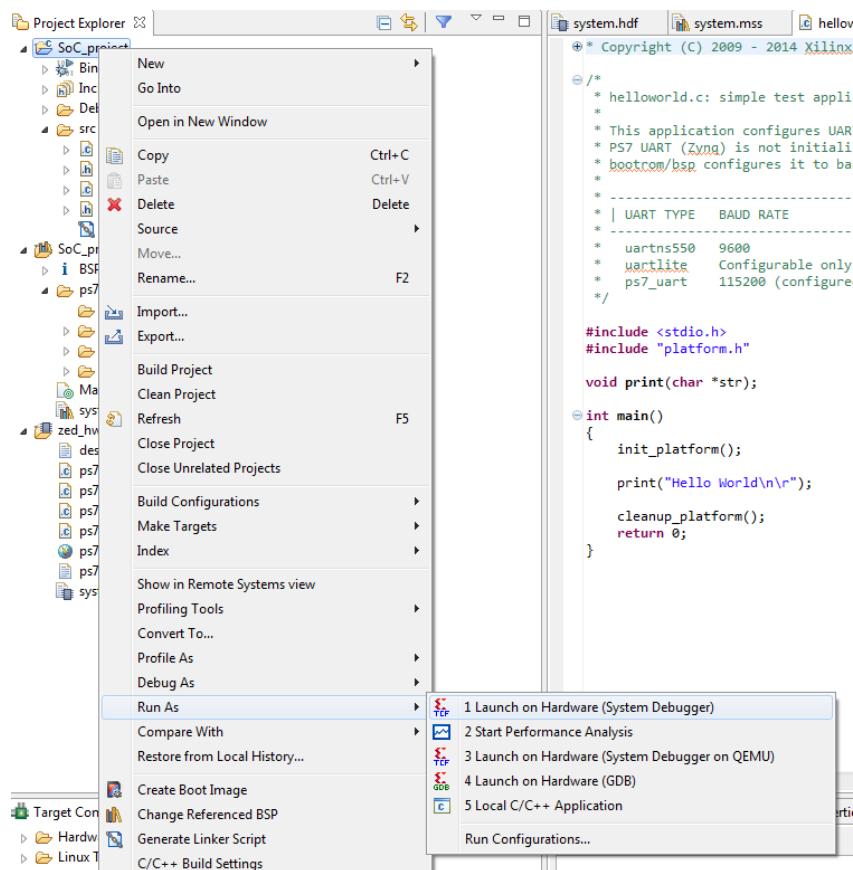
-Defining the Terminal parameters :

On the SDK Terminal, click on the green cross and select the used COM PORT and a baud rate of 115200.

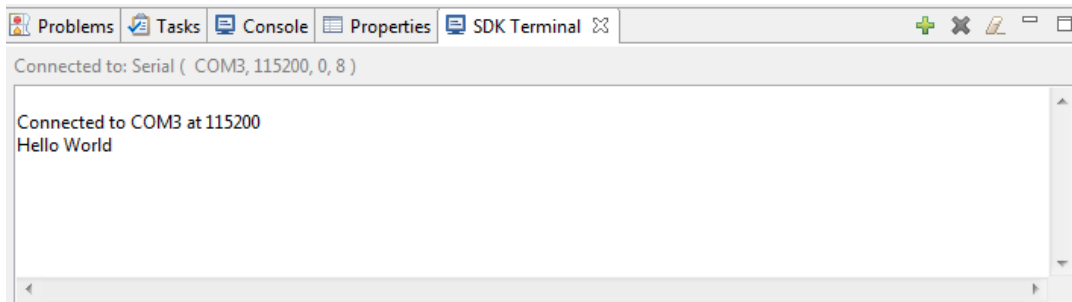


-Running the application :

Right Click on the **SoC_Project** and select **Run As** and select **Launch on Hardware (System Debugger)** :

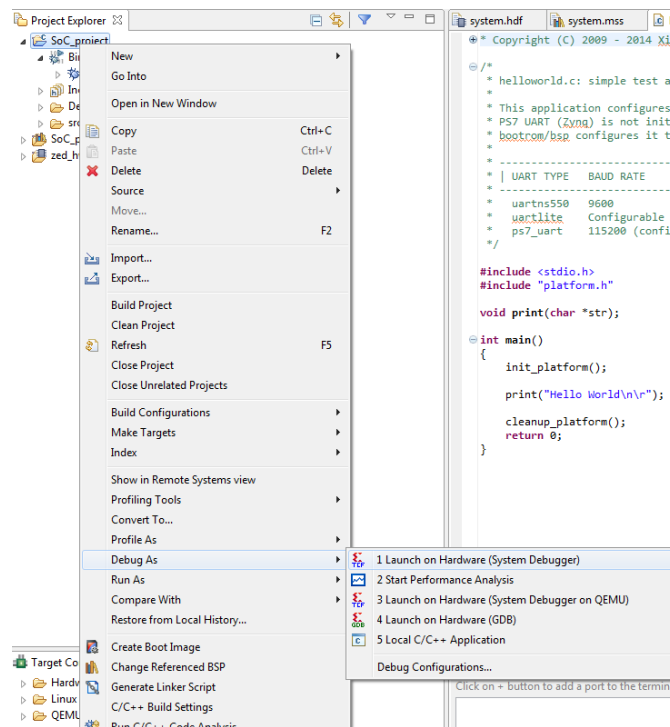


As you can see in the SDK Terminal, Hello Word message was sent by the processor.



Debug an application

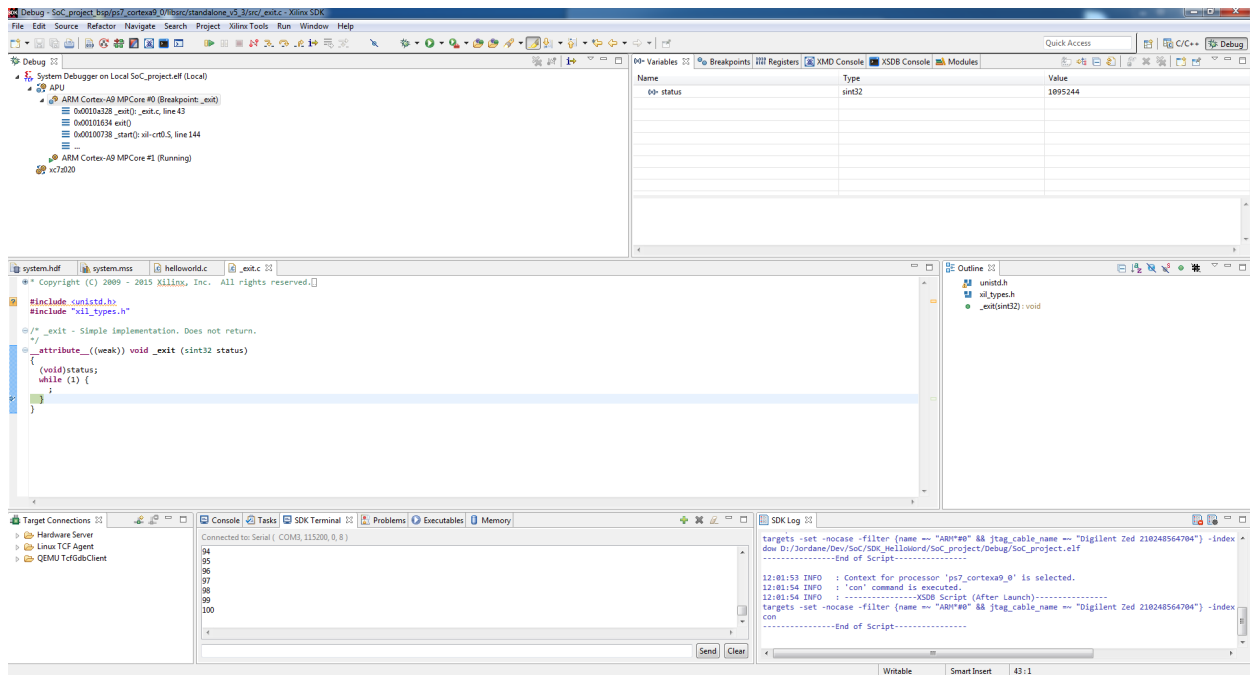
- 1) Modify the main.c file in order to print the value going from 0 to 100 on the serial link.
- 2) Build the project **Project/Build**
- 3) Debug your application : Right click on the project, **Debug As / Launch on Hardware (System Debugger)** (**System Debugger**)
- 4) **Click Yes** for the perspective switch.



If necessary, in the SDK terminal, open the COM port as previously.

Now, during the debug, you can perform a step by step analysis of the software using F5 or F6. You can also add breakpoint, execute the entire program (F8), access the variables, and core registers, etc.

Note : To go back to the C/C++ perspective, click on the C/C++ (next to debug on the upper right corner)



Conclusion

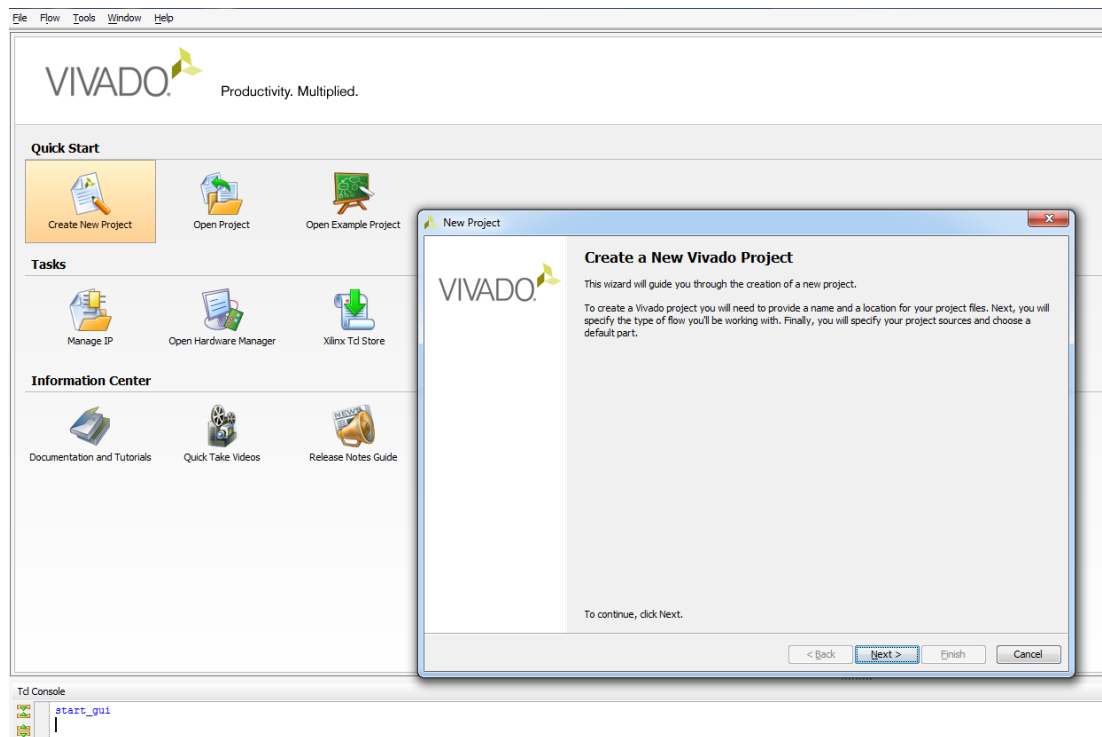
Through this example, you can understand how to easily program the arm processor. However, let's imagine you have to add more hardware peripherals, then you have to design your own zynq-based system under Vivado and finally program your software.

Exercise 2 : System Design under Vivado and Software programming under SDK

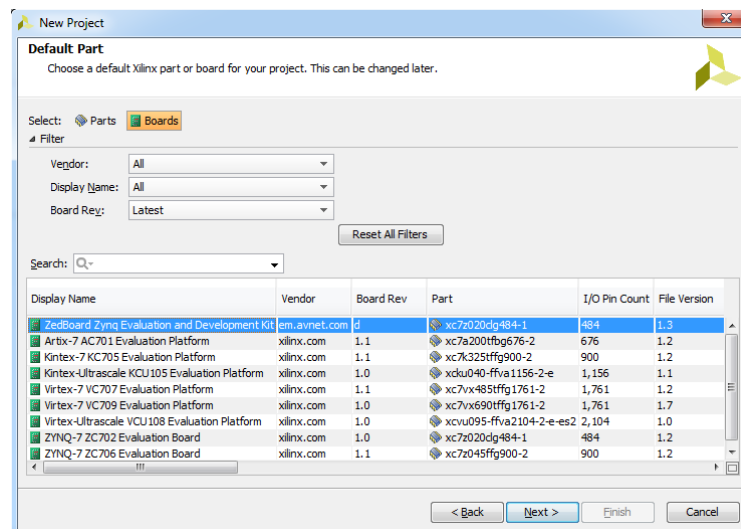
In this exercise, we want to directly manage a hardware peripheral from the ARM processor. Before writing the software, we have to design the appropriate system under Vivado.

I. System Design using VIVADO

- 1) Invoke the Vivado IDE by clicking the Vivado desktop icon or by typing **vivado** at a terminal command line
- 2) From the Getting Started page, select **New Project**. The New Project wizzard opens. Click **Next**.



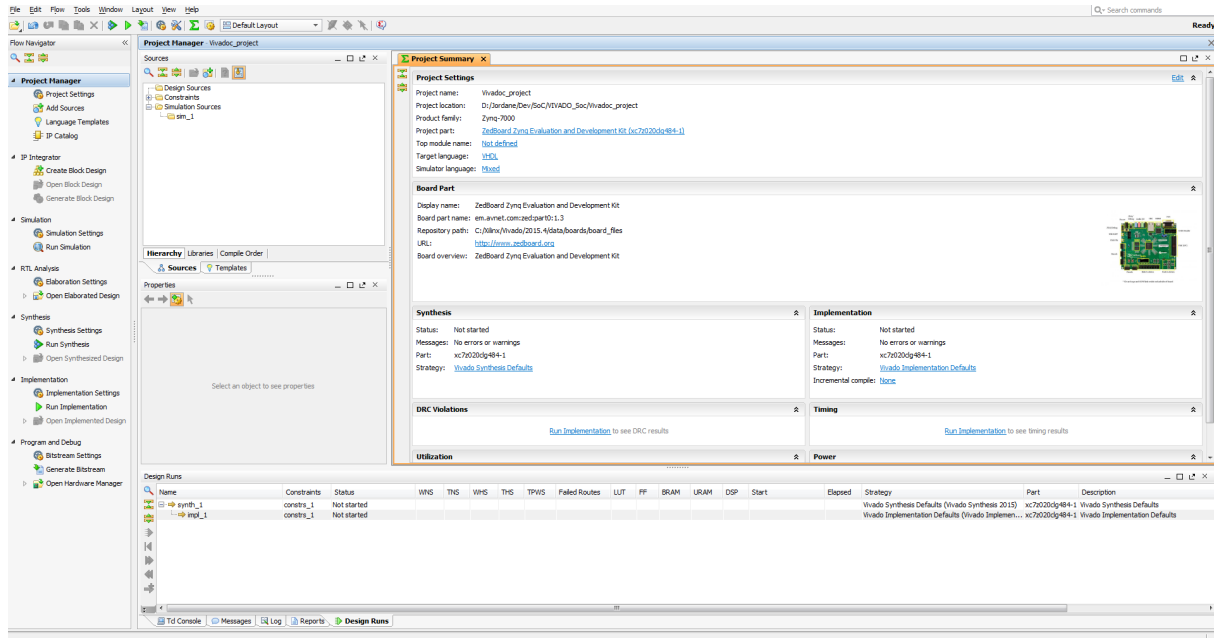
- 3) In the **Project Name** dialog box, type the project name and location (personal directory²). Ensure that **Create project subdirectory** is checked. Click **Next**.
- 4) In the **Project Type** dialog box, select **RTL Project**. Click **Next**.
- 5) In the **Add Sources** dialog box, ensure that the **Target language** is set to **VHDL**. Click **Next**.
- 6) In **Add Existing IP** dialog box, click **Next**.
- 7) In **Add Constraints** dialog box click **Next**.
- 8) In the **Default Part** dialog box Select **Boards** and choose **zedboard**. Click Next.



² Be sure to avoid any special characters (space, é, à, -,...). Only underscore is allowed.

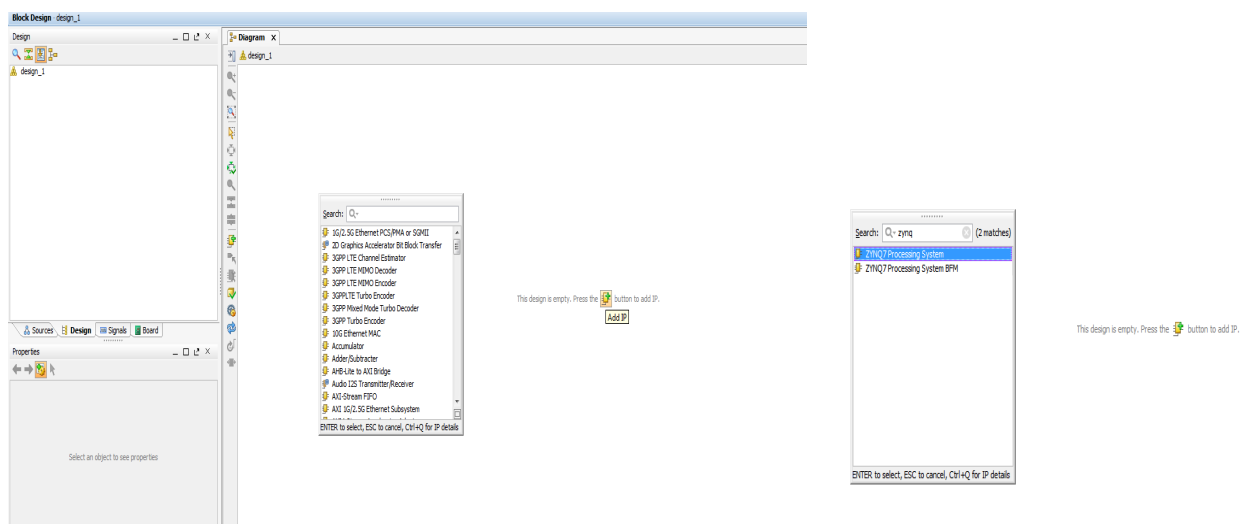
- 9) Review the project summary in the **New Project Summary** dialog box before clicking **Finish** to create the project.

Your project is now configured and the main view of VIVADO is presented as illustrated below:



-Creation of the system

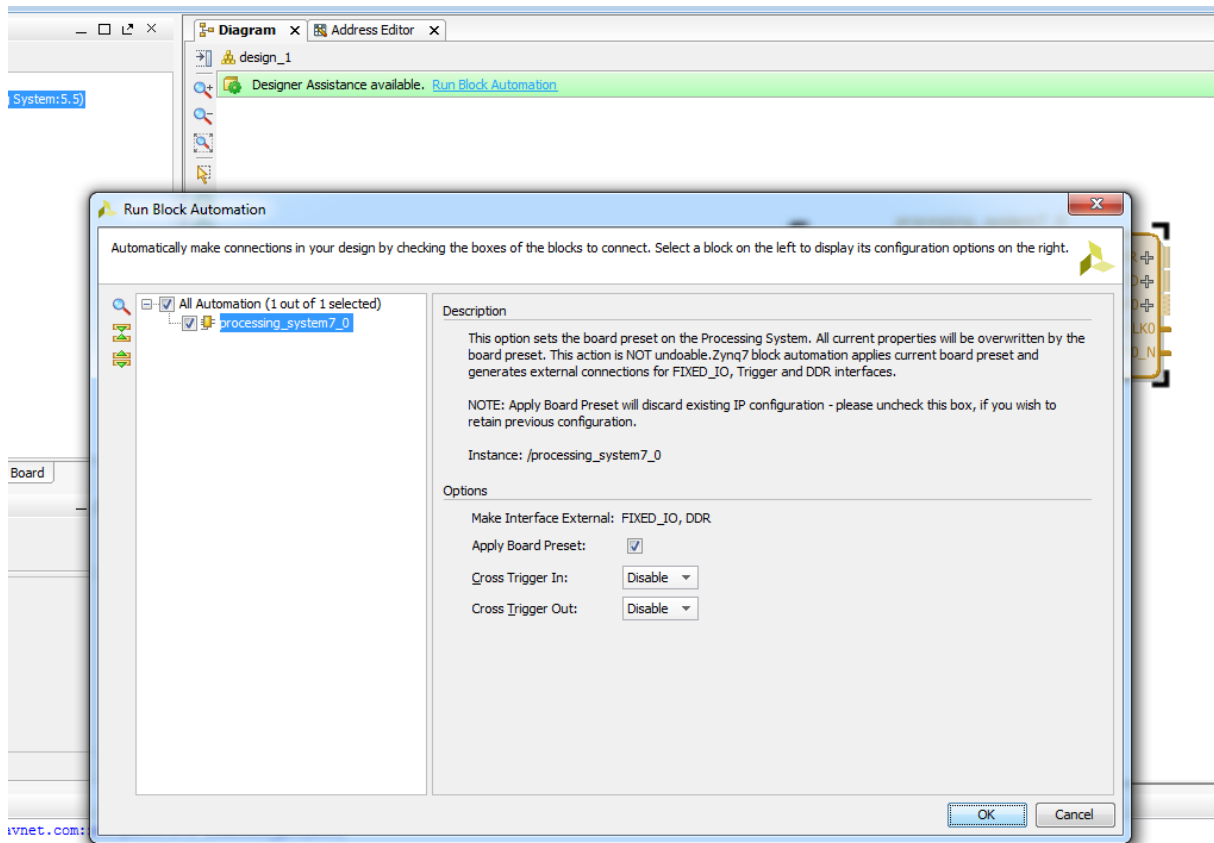
- 1) In the Flow Navigator (on the left side), select the **Create Block Design** option
- 2) Select a name for your design e.g. **zynq_design_1** and **OK**
- 3) In the Diagram, click on **Add IP**. A list of available IPs is presented.
- 4) Search for **zynq** and double click on **ZYNQ7 Processing System**



Note that the VIVADO IP Integrator configures the Zynq processing system properly with respect to the target board (Zedboard)

- 5) A message in green is indicated at the upper front of the Diagram View.

- ➔ It enables to run an assistance to automatically connect the main signals of the zynq processing system to the FPGA pins.
- ➔ Click on Run Block Automation
- ➔ Read the message and click **OK**



6) Now, we can add other peripherals to go in the processing logic (PL).

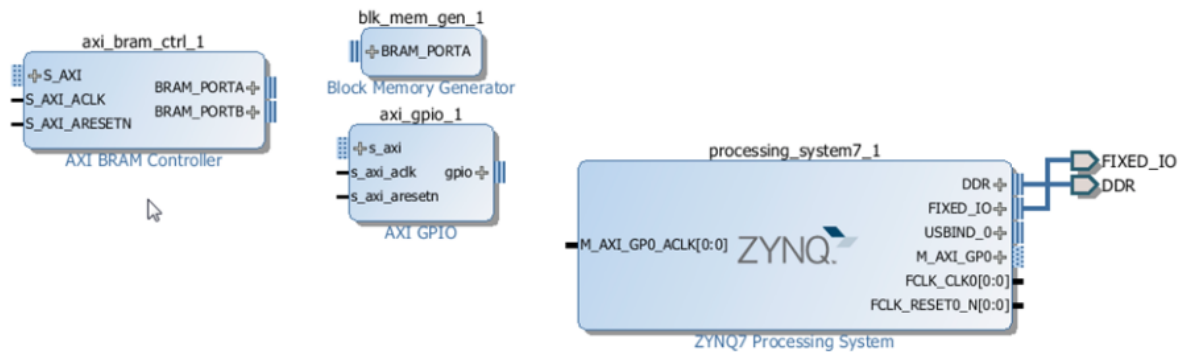
Right-click the Block Design window, and select **Add IP**. Type the following names in the search field of the IP integrator catalog individually:

- GPIO
- AXI BRAM Controller
- Block Memory Generator

Q : For you, what is the goal of the AXI BRAM Controller ?

Q : What is the goal of the GPIO IP?

Now, the Block Design window must match the figure illustrated below (relative positions of the IPs can be slightly different):

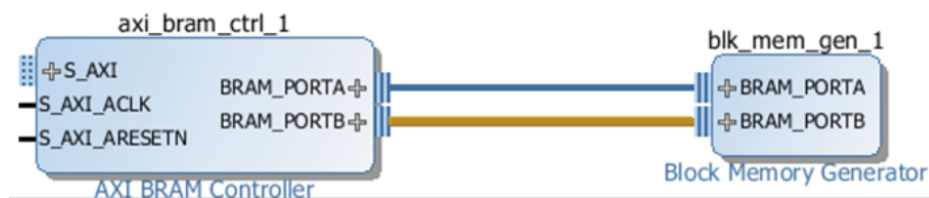


-Customize the instantiated IPs

- 1) Double-click, or right-click the **Block Memory Generator** IP, and select **Customize Block**
- 2) In the Basic tab of the dialog box, set:
 - **Mode** to **BRAM Controller**
 - **Memory Type** to **True Dual Port RAM**

Then click **OK**

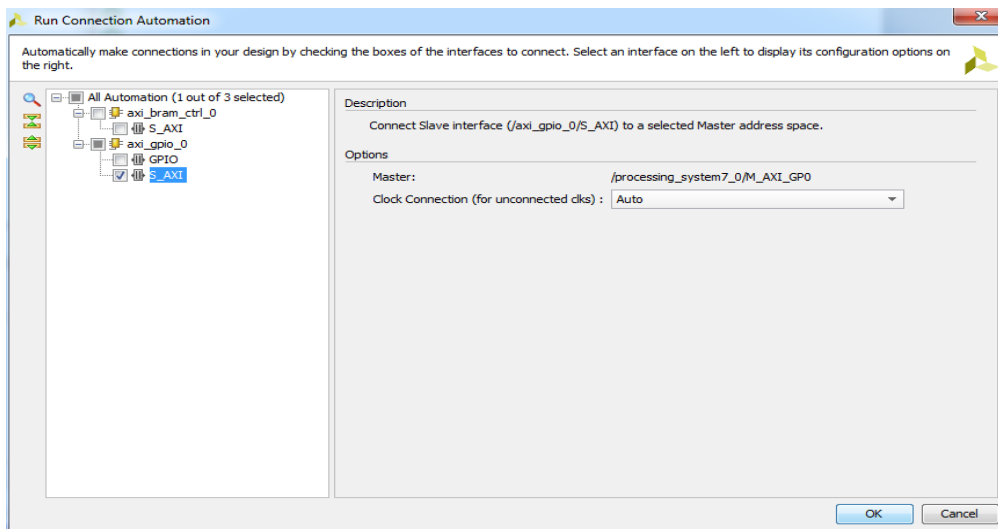
- 3) Connect the Block Memory Core to the AXI4 BRAM Controller by clicking the connection point and dragging the line between the IP, as indicated below :



- 4) The Block Designer Assistance helps in connecting the GPIO and AXI BRAM Controller to the Zynq-7000 PS. Select **Run Connection Automation** to connect the BRAM controller and GPIO IP to the Zynq PS and to the external pins on the Zedboard

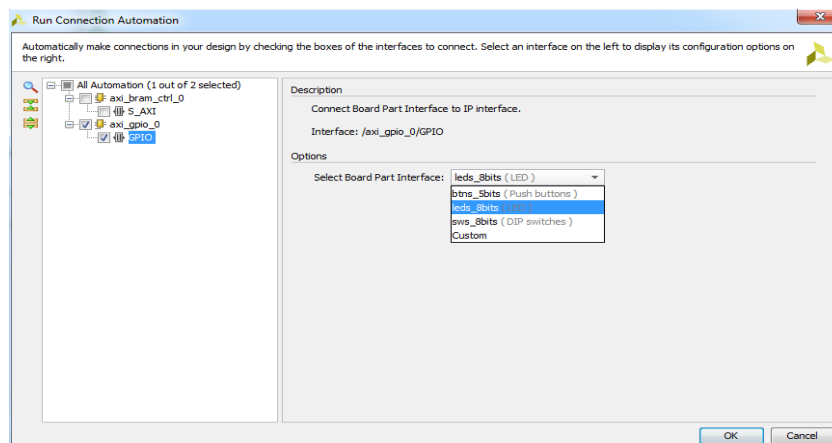
4.1- we connect the GPIO AXI interface to the AXI Master Interface (i.e the Zynq PS)

Click **OK**.



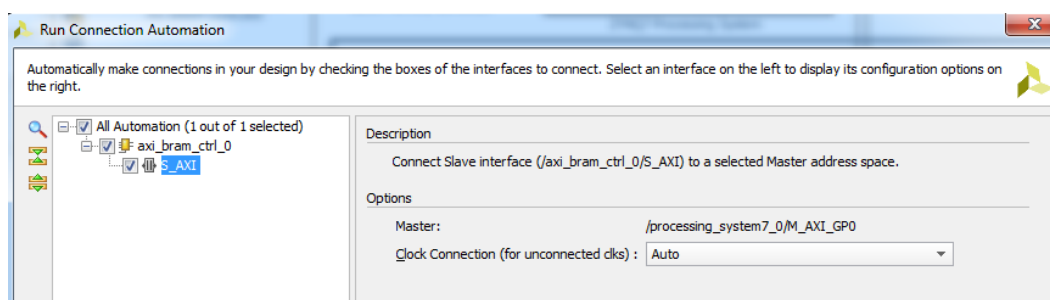
Note that this action instantiates an AXI Interconnect IP as well as a Proc Sys Reset IP and makes the interconnection between the AXI interface of the GPIO and the Zynq P-7000 Processing System (PS).

4.2- Select **Run Connection Automation** again, and select the configuration as indicated by the following figure :



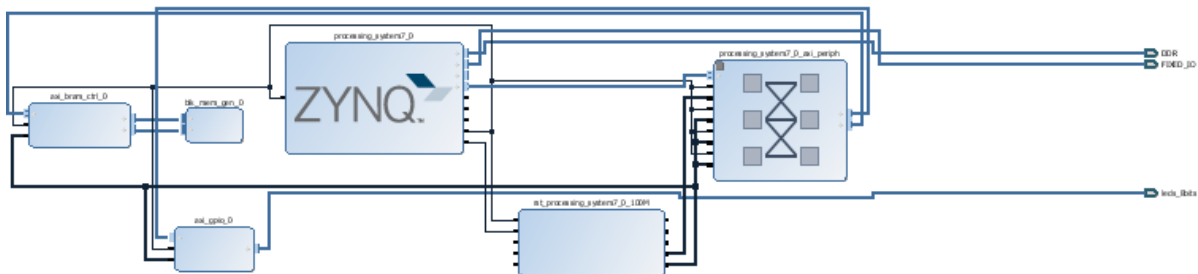
Q : What connection is realized ?

4.3- Click **Run Connection Automation** again, and select the remaining option



This completes the connection between the Zynq-7000 Processing System and the AXI BRAM Controller.

Finally, you must obtain a system that looks like the figure below.

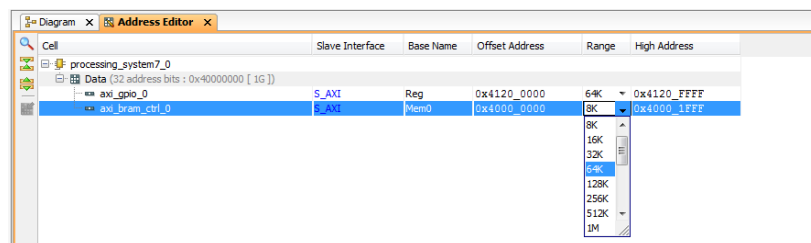


5) Memory base addresses

The Address Editor tab shows the memory map of all the IP present in the design. In this case, there are only two IPs, the AXI GPIO and the AXI BRAM Controller. The IP integrator assigns the memory maps for these IP automatically.

6) BRAM Controller configuration

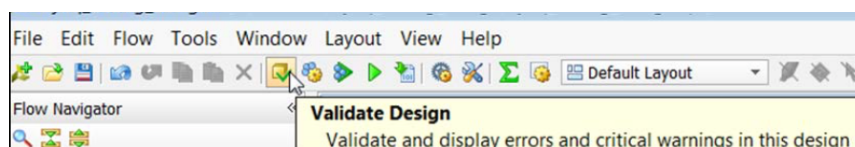
Double-click on the BRAM controller, Change the range of the AXI BRAM Controller to **64K**.



7) Design Validation

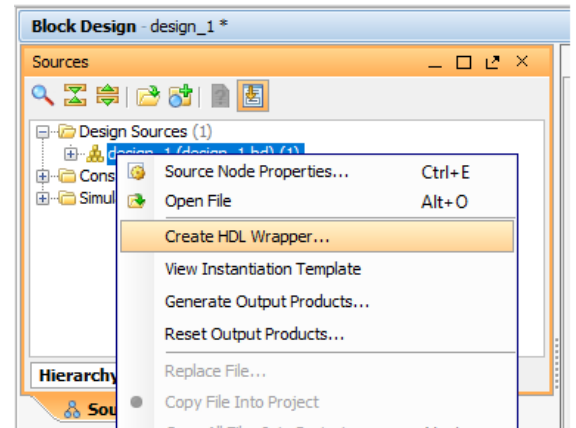
In order to validate the design, save your design by pressing **Ctrl-S**, or alternatively, select **File > Save Block Design**.

From the toolbar, run Design-Rules-Check (DRC) by clicking the **Validate Design** button. Alternatively, you can do the same from the menu by selecting **Tools > Validate Design**.



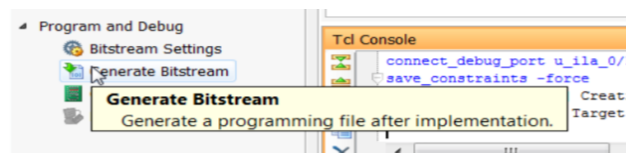
-VHDL Code Generation

- 1) In the Source Window, In the Source window, right-click on the top-level subsystem design, and select **Generate Output Products**. This action generates the source files for the IP used in the block diagram and the relevant constraints file.
- 2) In the Sources window, select the top-level subsystem source, and select **Create HDL Wrapper** to create an example top level HDL file. If any dialog box appears, click **OK** or **YES**.



-Synthesis to Bitstream Generation

- 1) Now, click on **Generate Bitstream** to launch synthesis and implementation steps until the generation of BIT file.



Note: If a dialog box appears indicating **No implementation Results Available**, click **YES**. These steps may take several minutes to complete (around 5-10 min). Be patient ☺

- 2) At the end, you can open the implemented design.

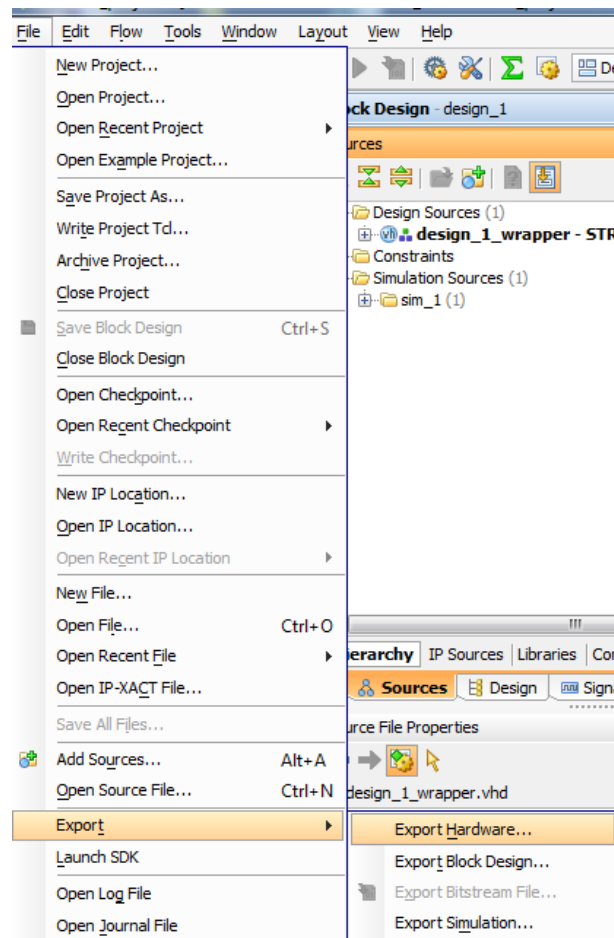
Q : Try to find which hardware resources are used in our design ?

Q : Does the design respect the timing constraint ? What WSN means ? Is it possible to have a negative value of WSN ?

-Export Bitstream to SDK

In this step, you export the hardware description to SDK.

- 1) From the main Vivado File menu, select Export Hardware for SDK




➔ The Export Hardware for SDK dialog box opens.

- 2) Ensure that **Export Hardware**, Include Bitstream, is checked and **OK**
- 3) File, **Launch SDK** and **OK**

II. Software Development using SDK


Under SDK, we want to manage the GPIO directly from the software running on the ARM processor.

- 1) Create a new **application project** based on our hardware called `design_1_wrapper_hw_platform_0` (if your previous zynq based design was called like this by default). Click **Next**.

 New Project

Application Project

Create a managed make application project.



Project name: GPIO_project

☒ Use default location

Location: D:\Jordane\Dev\SoC\VIVADO_Soc\Vivadoc_project\Vivadoc_

Browse...

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

New...


Processor: ps7_cortexa9_0

Target Software

Language: ☒ C ☐ C++

Compiler: 32-bit

Board Support Package: ☒ Create New GPIO_project_bsp ☐ Use existing



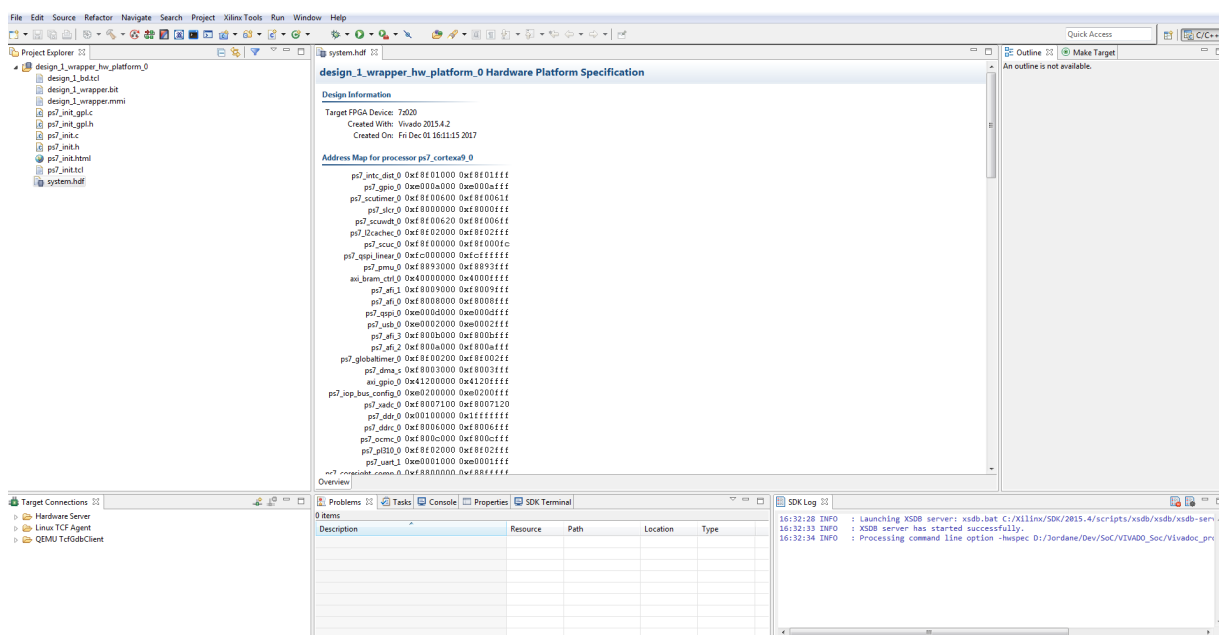
< Back

Next >

Finish

Cancel

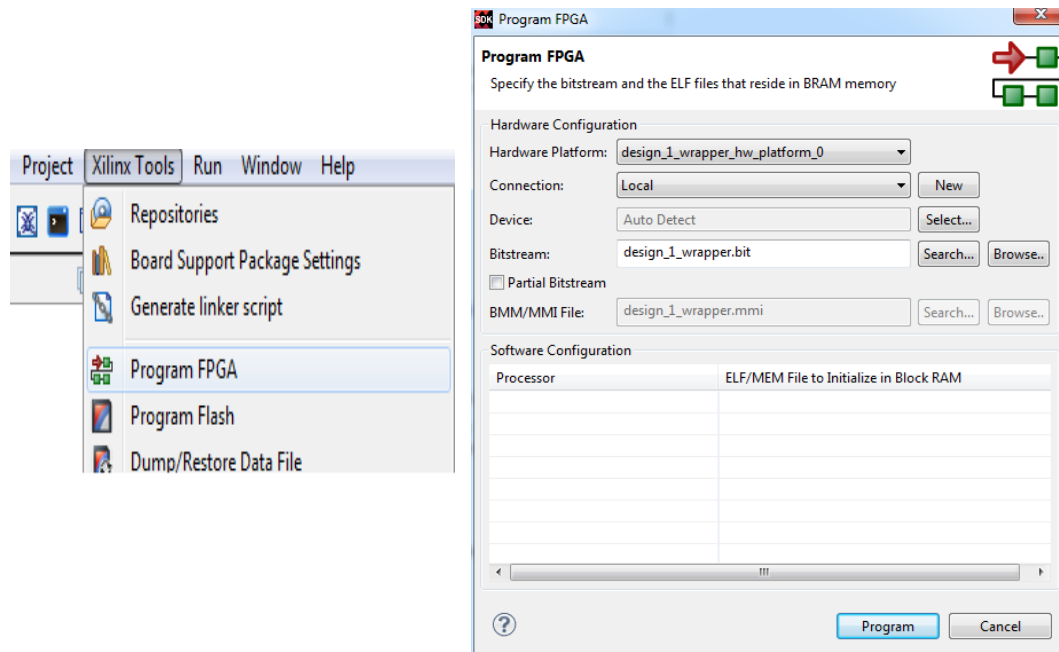
2) Choose the **HelloWord Application Project** and click on **Finish**



- 3) Look at the **system.mss file**. The two first peripheral drivers must tell you something. The other peripheral drivers are included by default for the Zynq processing system.
- 4) Develop a program that enable to send the value of a counter to the leds. The counter is incremented every second.

To help you, have a look at the 'xgpio.h', 'sleep.h' and 'xil_types.h' header files, located in the include folder of the BSP.

- 5) Once your program is written, go to **Xilinx tools -> Program FPGA**
- 6) Right click on the **project -> Run As -> Launch on Hardware (System Debugger)**



-Conclusion

In this example, you have learnt how to design a zynq processing system and add a peripheral using VIVADO. A software code running on the ARM processor directly controls the leds.

Exercise 3 : To go further...

Exercise 3.1

Based on the previous examples, we want now read the values from the switches of the board and write the value to the leds and the Serial port.

The steps :

- Modify the previous system design under VIVADO
- Perform synthesis to bistream generation steps.
- Export bitstream to SDK.

-Develop the appropriate code to realize the functionality.

Exercise 3.2

Try to develop your own 16-bit multiplier in VHDL and integrate it into a Vivado project.

When your VHDL code is ready, Go to

- Tools > *Create and Package IP* and click *NEXT*
- Select *Create a new AXI4 peripheral*

When the bitstream is generated, you will have now to write the good C/C++ application code under SDK in order to test your hardware multiplication.

Conclusion & Summary

During this practical work, you have learned a lot of techniques for SoC design and programming onto FPGA device.

During the next course, you will learn how to fully exploit the reconfigurable architecture using dynamic partial reconfiguration of the FPGA.

Bibliography

[1]Zedboard, ‘*ZedBoard (Zynq™ Evaluation and Development) Hardware User’s Guide*’, v2.2 jan. 2014,

http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[2]Xilinx, ‘*Vivado Design Suite Tutorial : Embedded Processor Hardware Design*’, User Guide UG940, v2014 April 2014,

https://www.xilinx.com/support/documentation/sw_manuals_j/xilinx2014_1/ug940-vivado-tutorial-embedded-design.pdf

[3]Xilinx, ‘*All Programmable SoC with Hardware and Software Programability*’,

<https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>