Advanced Workshop on Modern FPGA-Based Technology for Scientific Computing

LABORATORY

Vivado Design Flow for a Simple PS Design

Prepared by:

C. Sisterna & L. Crespo

ICTP-MLAB

Section I

Vivado Design Flow for a PS Based Design

Introduction

This lab guides you through the process of using Vivado Development Suite to create a simple SoPC design targeting just the PS part of the Zynq-FPGA in the ZedBoard. You will create the board design in the Vivado IP integrator, to export it to the SDK tool and generate the board support package and use an already done template to display the "Hello world" string in a console.

Objectives

After completing this lab, you will be able to:

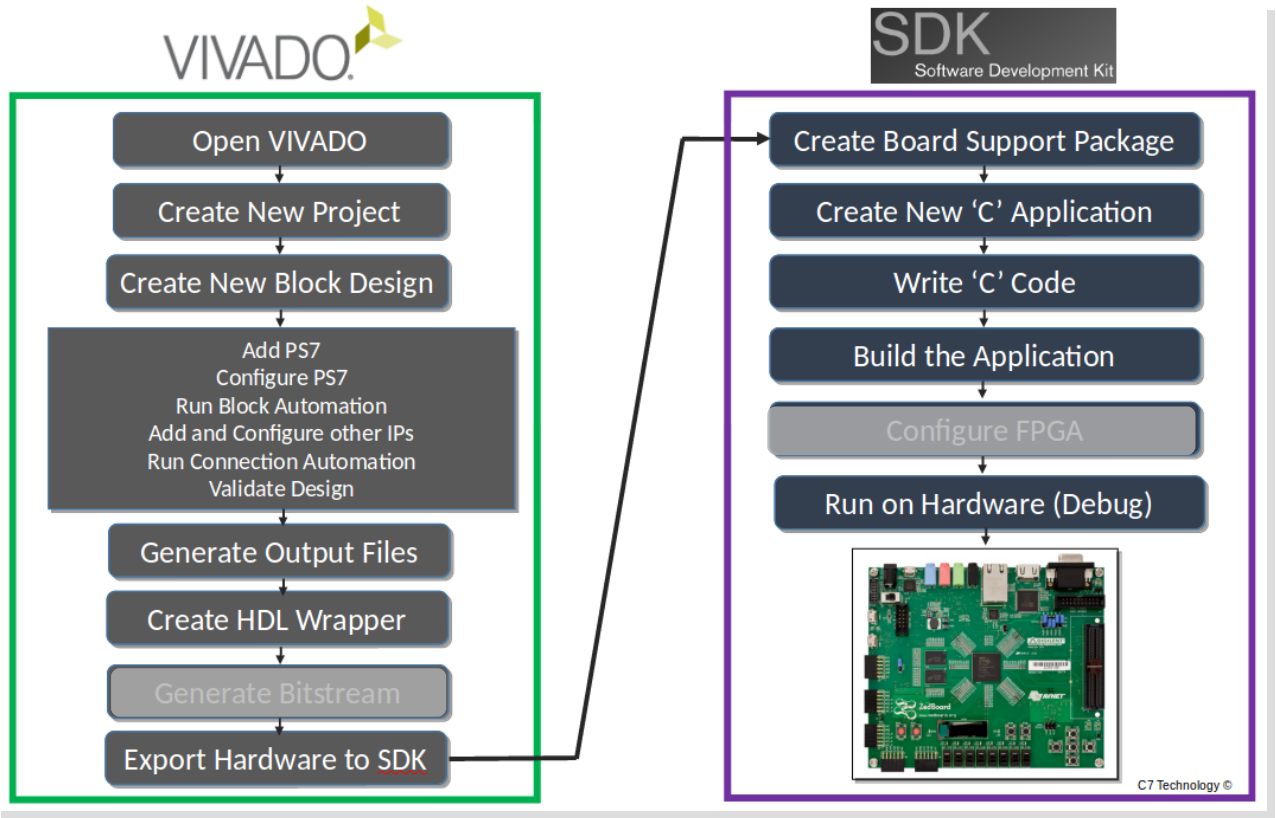
- Create a Vivado project based in the IP Integrator
- Add and configure the PS7
- Generate HDL wrapper
- Export the design to SDK
- Create an application project in SDK, creating a board support package and 'C' code
- Configure the UART to communicate the PC with the ZedBoard
- Program the PS7 using the .elf file
- Execute the 'C' code in the processor

Description

The design consists of creating a simple project in which the PS7 will be configured to communicate with the PC to display the 'Hello World' string.

Design Flow

According to the presentation in class the flow detailed below should be followed in this laboratory:



Following is a resume about each of the processes in the above flow towards this Lab:

1. The design and implementation flow begins with launching Vivado. Within Vivado the entire design, from creating a block diagram to generate the bitstream, is carried out.

2. Open the Create New Project Vivado option.

3. From Vivado GUI, select Create Block Design to launch IP Integrator. Add the ZYNQ7 Processing System IP to include the ARM Cortex-A9 PS in the project.

4. Double click on the ZYNQ7 Processing System block to configure the PS settings to make the appropriate design decisions such as selection/de-selection of dedicated PS I/O peripherals, memory configurations, clock speeds, etc.

5. At this point, you may also optionally add IP from the IP catalog or create and add your own customized IP. Connect the different blocks together by dragging signals / nets from one port of an IP to another. You can also use the design automation capability of the IP Integrator to automatically connect blocks together.

6. When finished, generate a top-level HDL wrapper for the system.

7. When a project is created by defining a board, e.g. ZedBoard, a default constraint file is added to the project. This .xdc file defines the association between the FPGA

I/Os and the peripherals existing in the ZedBoard. In case of using an FPGA I/O that is not associated to any peripherals, e.g. the JA1 PMOD connector, a customized .xdc file has to be added to the project. If there is any signal coming from the PL section to an I/O pin that is not defined in the .xdc file, then the tools will generate an error during the bitstream generation. Hence, in case of needed add a Xilinx Design Constraints (XDC) file to the Vivado project.

8. Generate the bitstream for configuring the logic in the PL, if soft peripherals or other HDL are included in the design, or if any hard peripheral IO (PS peripheral) were routed through the PL. The PL part of the FPGA can be configured from either from SDK. The configuration form the SDK is the most commonly used.

9. Once, the hardware portion of the embedded system design has been built, export the design to the SDK to create the software design. A convenient method to ensure that the hardware for this design is automatically integrated with the software portion is achieved by Exporting the Hardware. File -> Export -> Export Hardware. Assure to check the "Include Bitstream" option.

10. Lunch SDK. File -> Lunch SDK.

11. Within the SDK, for a standalone application (no operating system) create a Board Support Package (BSP) based on the hardware platform and then develop your user application. Once compiled, a *.ELF file is generated.

12. Create a new 'C' application (usually from the available templates).

13. Write your own 'C' code according to the requirements of the project.

14. In case there is logic in the PL part of the Zynq, it is needed to configure the FPGA with the respective .bit file.

15. Execute the Run on Hardware (Debug) process to program the PS part of the Zynq with the respective *.elf file, and automatically execute the 'C' code in the processor.

***Note**: the steps of the flow design printed in light gray are steps that are not necessary in this lab, but will be used in following ones.*

Create a Vivado Project

Objective Execute Vivado and create a PS7 based project targeting the ZedBoard.

1. Open **Vivado Design Suite.**
2. From the **Quick Start** menu, click **Create Project** to start the wizard

or click **File → Project → New**.

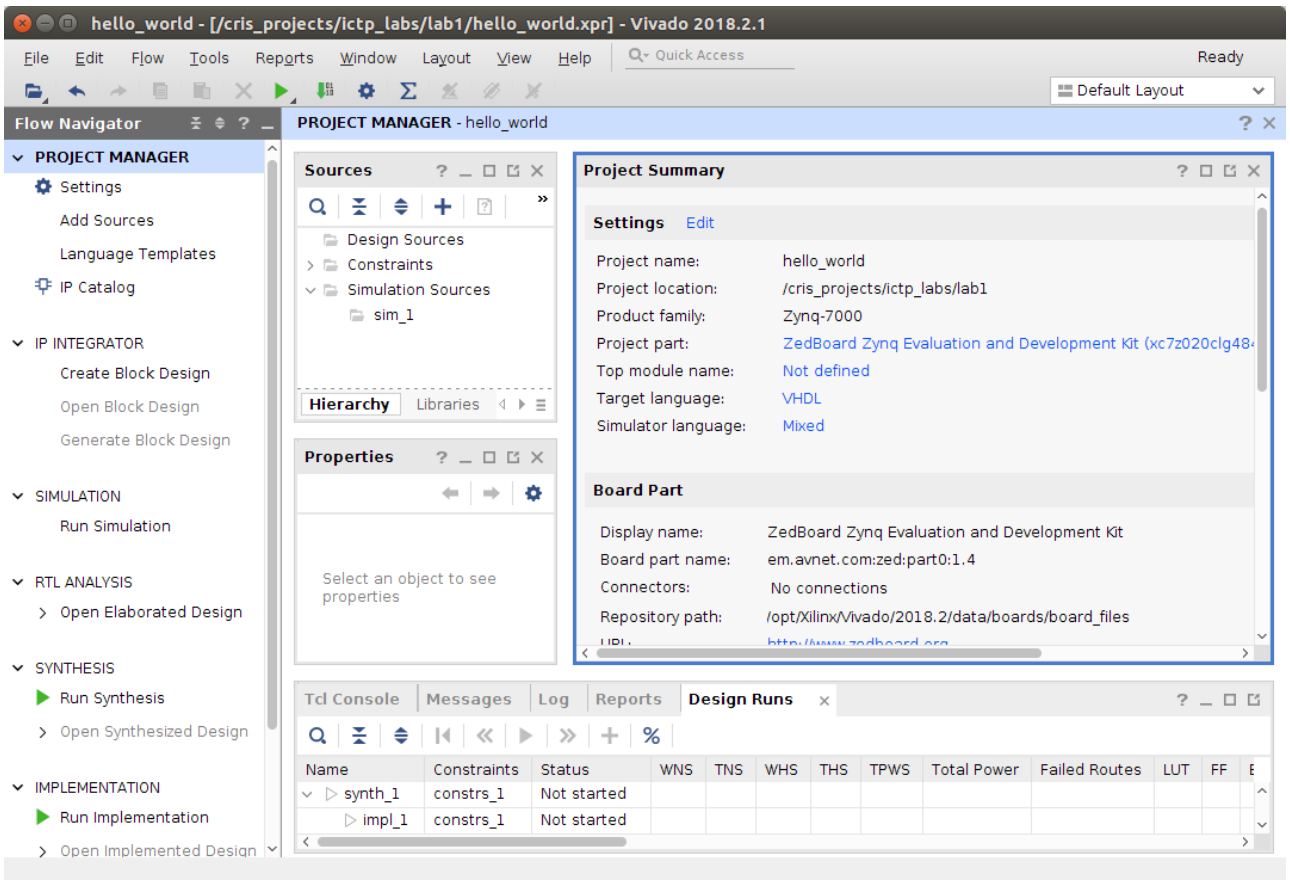
   You will see **Create A New Vivado Project** dialog box in the **New Project** window. Click **Next**. Use the information in the table below to configure the different wizard options:

Wizard Option	System Property	Settings
Project Name	Project Name	hello_world
	Project Location	**C:\...\...\labs\lab_hello_world**
	Create Project Subdirectory	Do not check this option.
Click **Next**		
Project Type	Project Type	Select **RTL Project**. Keep do not specify sources at this time box unchecked
Click **Next**		
Add Sources	Do nothing	
Click **Next**		
Add Existing IP	Do Nothing	
Click **Next**		
Add Constraints	Do Nothing	
Default Part	Specify	Select **Boards**
	Board	Select **ZedBoard Zynq Evaluation and Development Kit**
Click **Next**		
New Project Summary	Project Summary	Review the project summary
Click **Finish**		

After clicking **Finish**, the **New Project Wizard** closes and the project just created opens in the Vivado main GUI.

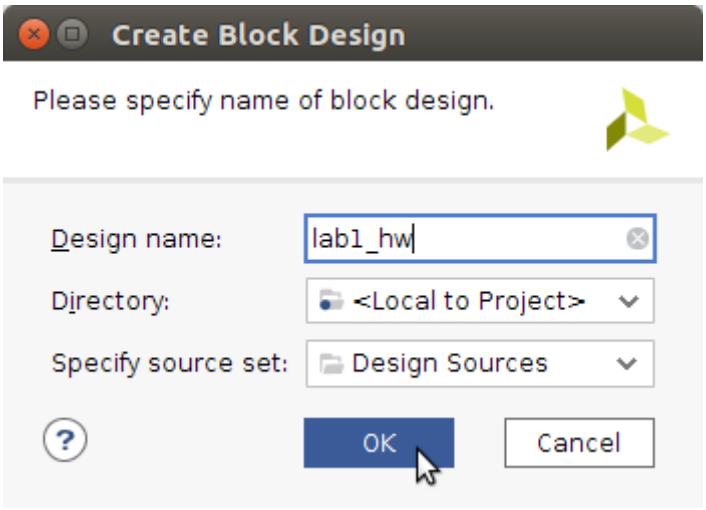
The board selected during the project creation, in this case the **ZedBoard**, has a direct impact on how the **IP Integrator,** within the **Vivado**, executes. **Vivado IP Integrator** is board aware and it will automatically assign dedicated Zynq IO ports to physical pin locations mapped to the specific board peripherals when the **Run Connection** wizard be used. Besides of doing a pin constraint, **IP Integrator** also defines the I/O standard (LVCMOS 3.3, LVCMOS 2.5, etc) to each IO pin; saving time to the designer in doing so. Therefore, the XDC file (the Xilinx Constrain File) associated to the pre-defined IO locations is not required from user when the design uses only the defined ZedBoard peripherals.

3. The Vivado Design Suite main window should look like the following figure:

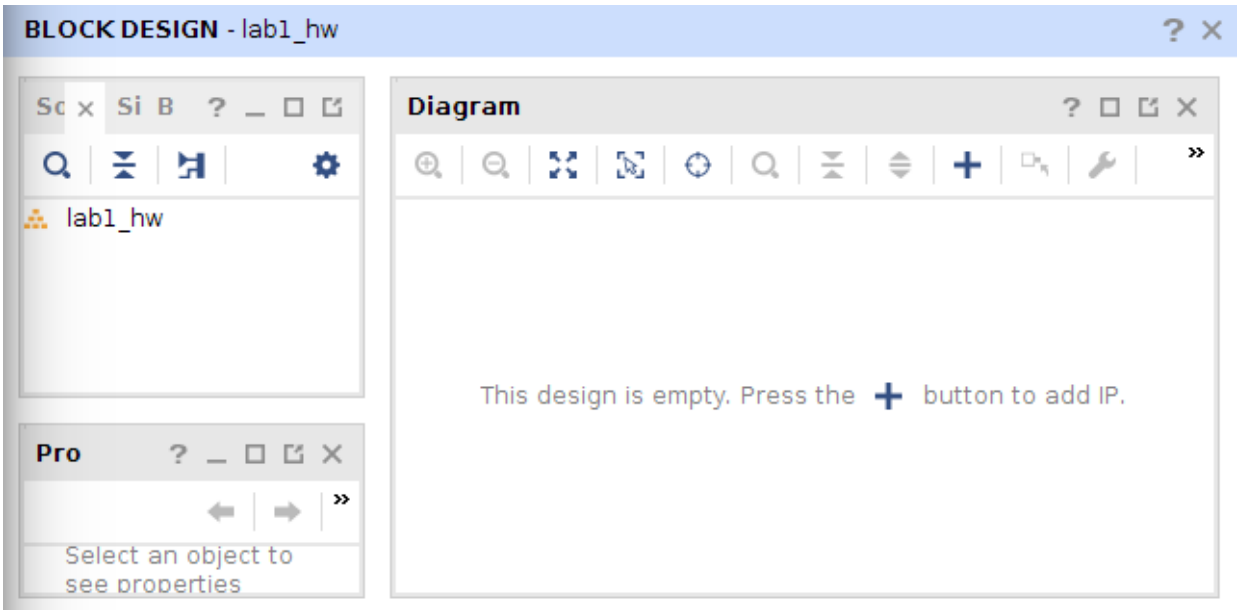


4. Next step is to use the **IP Integrator** to create an embedded processor project.

4.1. Click **Create Block Design** under in the **Flow Navigator** pane in the **Flow Navigator** pane. Type in **lab1_hw** as **Design Name** in the **Create Block Design** window, and click **OK**.



4.2. A new blank Block Diagram canvas will be presented. This canvas will be used to create the design to be implemented into the Zynq device.

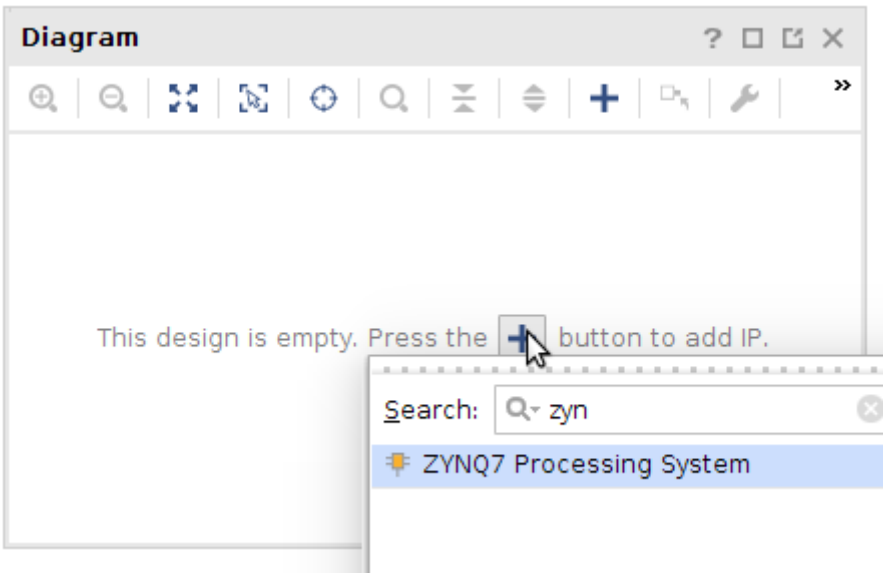


4.3. You can design a new embedded system in **Vivado** using **IP Integrator** by adding a **ZYNQ7 Processing System** block. By adding this block, you can configure one of the ARM Cortex-A9 processor cores for your application. You can also place additional IP blocks to increase the capabilities of the embedded system.

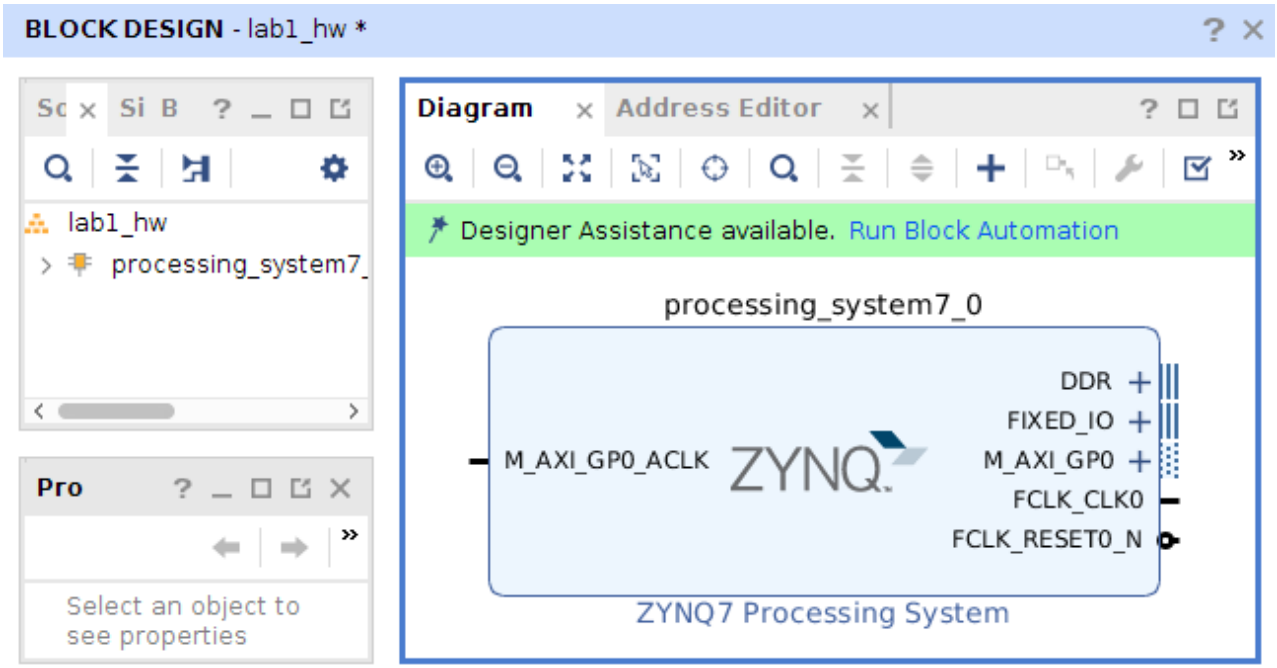
To insert a **ZYNQ7 Processing System**(PS7) block you can either click the **Add IP** icon + or do a right click on the canvas blank space and select **Add IP** from the available options.

4.4. An small window will come up showing the available IPs (that is, they are the **Intellectual Property** cores, **IP**, that are already available. We will see later, in other lab, how to create and add our own IP. To search and add the PS IP core, we can either scroll down to the very bottom of the IP list or search the IP using the keyword **zynq**. Double click on the **ZYNQ7 Processing System** IP to select and add it to the canvas.

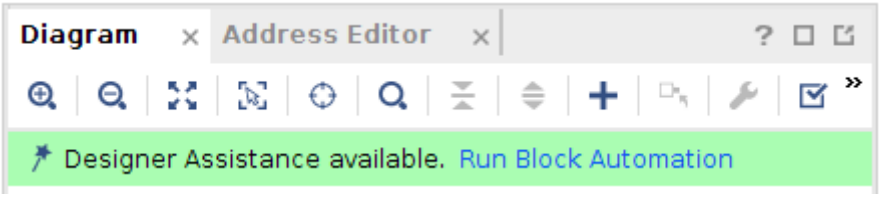
5. The **Zynq7 PS** IP block is placed in the block diagram canvas. The I/O ports shown



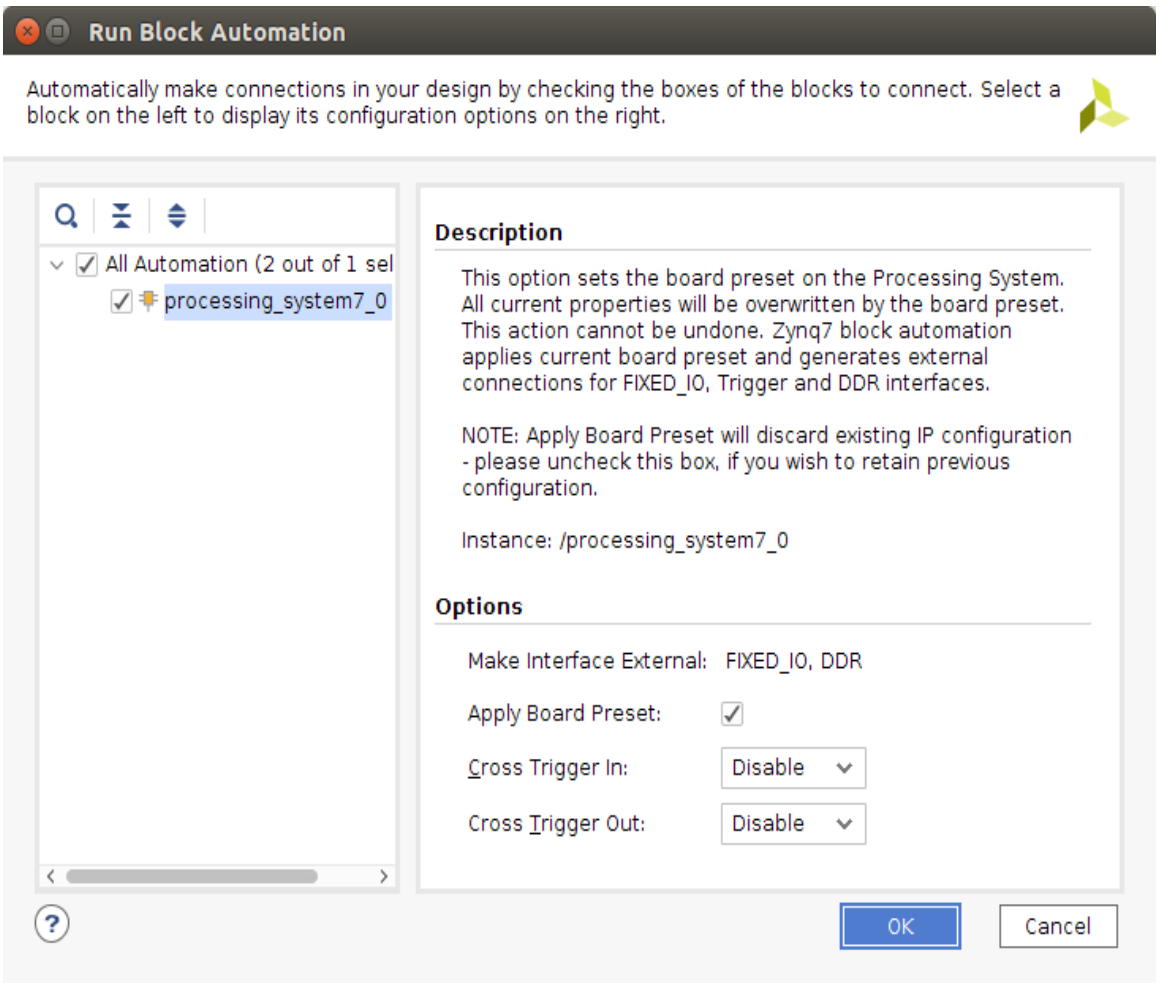
in the block diagram, DDR, FIXED_IO, M_AXI_GPO, etc., are defined by the default settings for this block as specified by the target development board (in this case the ZedBoard).



6. Click **Run Block Automation**, available in the green information bar.



7. Then, select **/processing_system7_0**. Make sure Apply Board Presets is checked and select **OK** in the **Run Block Automation** window (leave everything else as default).



8. After finishing previous step, the block diagram should look like the following:

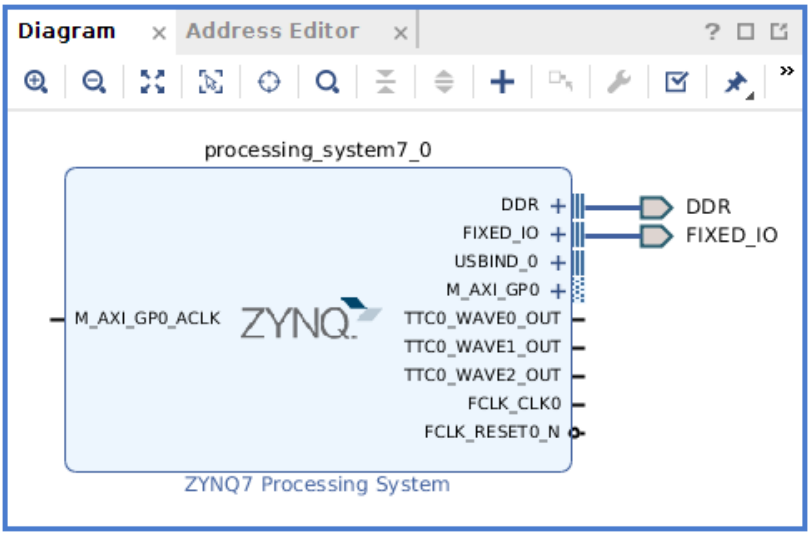
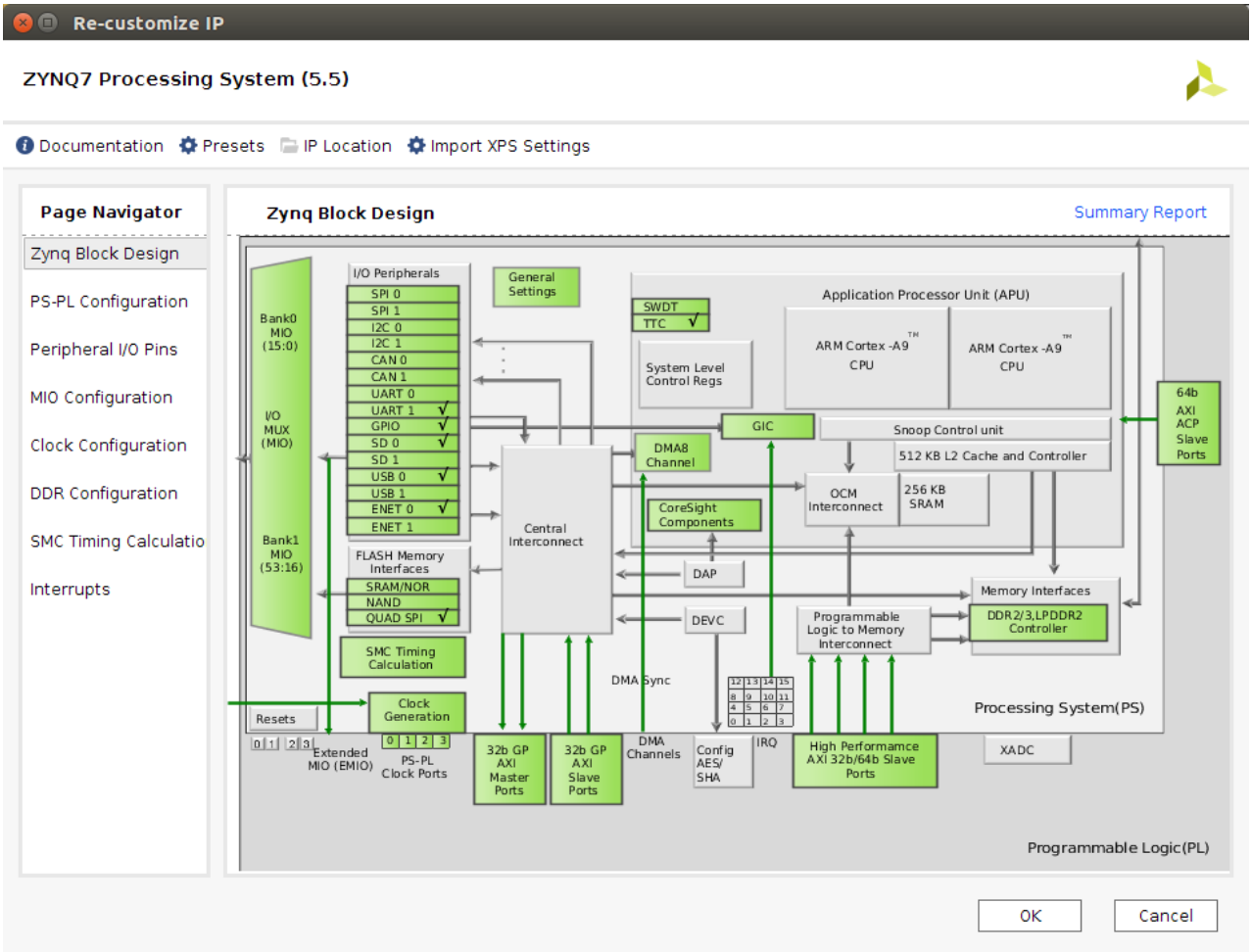


Diagram
Address Editor
processing_system7_0
DDR
FIXED_IO
USBIND_0
M_AXI_GP0
M_AXI_GP0_ACLK
TTC0_WAVE0_OUT
TTC0_WAVE1_OUT
TTC0_WAVE2_OUT
FCLK_CLK0
FCLK_RESET0_N
DDR
FIXED_IO
ZYNQ7 Processing System

Processing System (PS) Customization

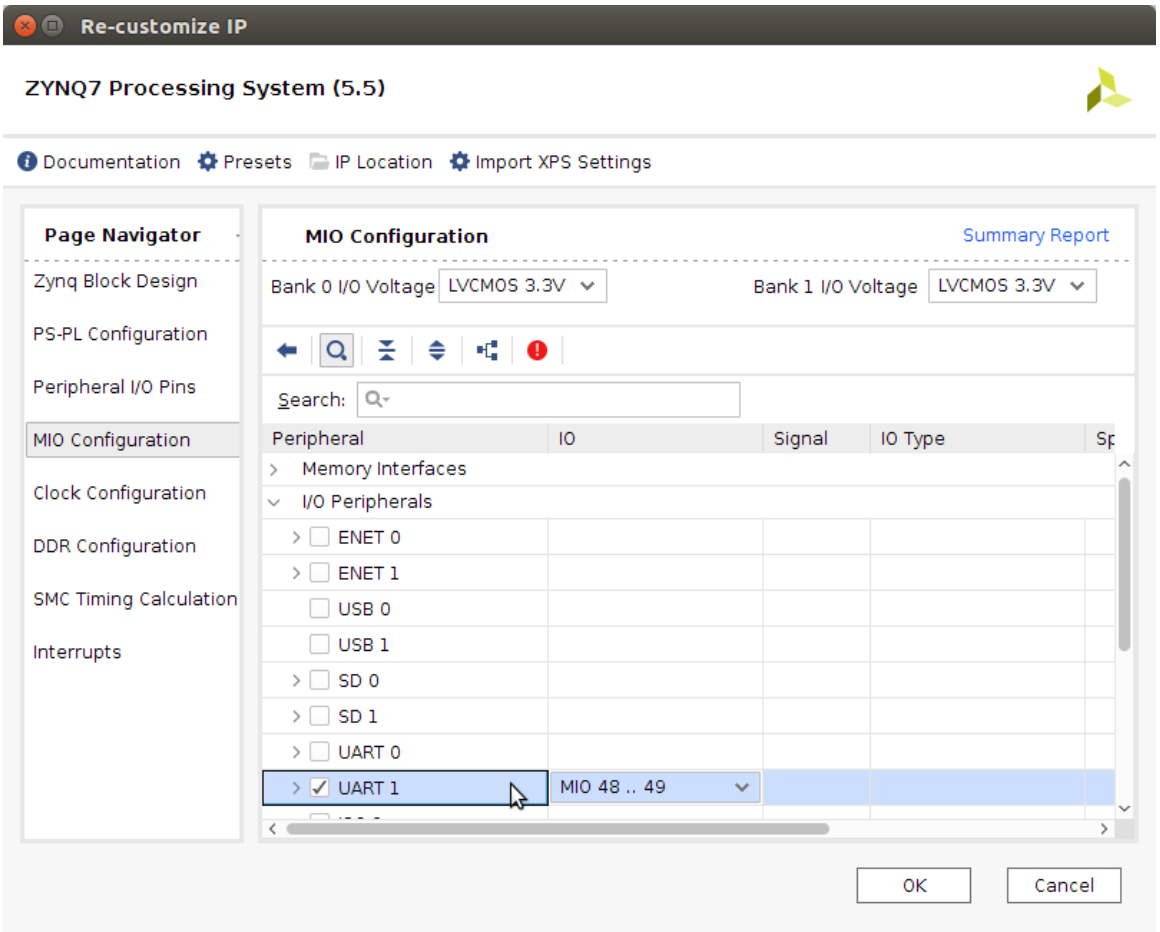
Objective Customizing the Zynq Processing System settings. For this particular Lab many of the defaults setting of the PS7 will not be necessary, therefore they will be modified in the following steps.

9. Double click in the **Zynq7 PS** block to open the customization window (see figure below). In this window the **Processing System** part of the **Zynq** device can be configured.



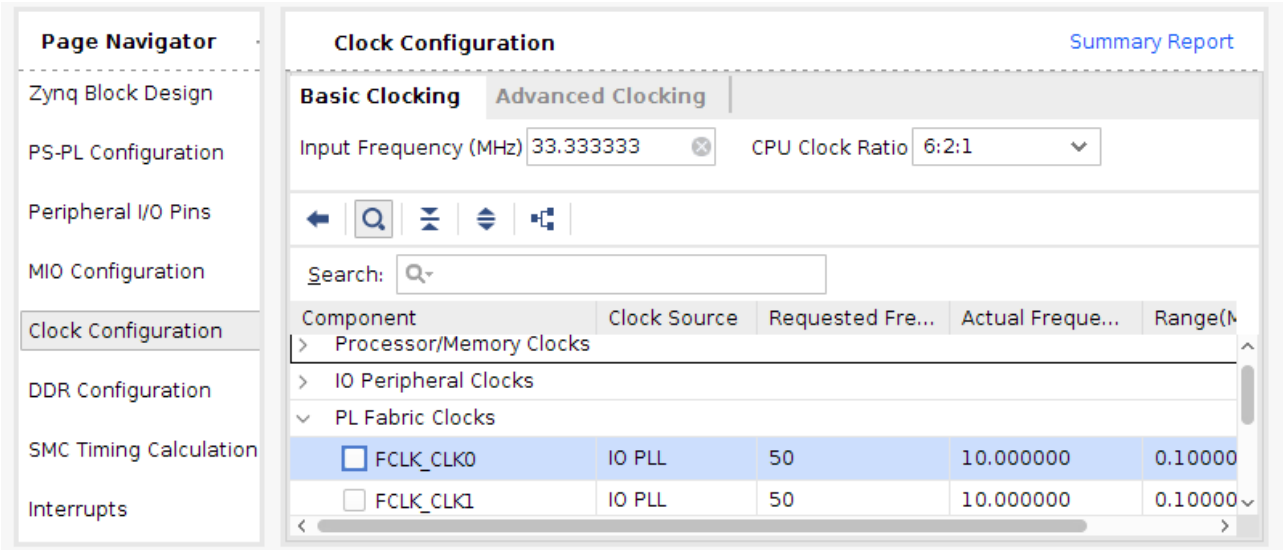
All the blocks colored in bright green can be customized. To select a block either double click on it or select the respective configuration option on the **Page Navigator** pane (the column on the right).

10. Let's begin with some of the configuration. Click on the **MIO Configuration** option under the **Page Navigator** pane. Expand I/O Peripherals, and **unselect** all the peripherals but the **UART1**. The **PS-UART1** will be used to communicate the Zynq device with the PC. This communication will be carried out by using a serial terminal software like Putty or TeraTerm.



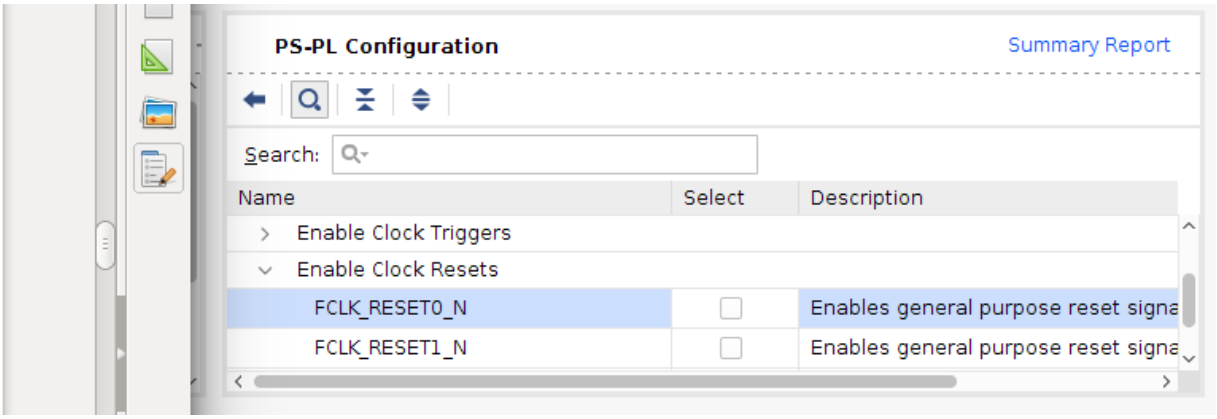
We can go to the **Peripherals I/O Pins** to see the Zynq I/O pins associated with the UART (this information is irrelevant in this lab, but it could be useful in some cases) .

11. Next, click on the **Clock Configuration** option, expand **PL Fabric Clocks,** and de-select **FCLK_CLK0.** Since there will not be any logic in the PL part of the Zynq it not necessary to supply any clock to the PL.

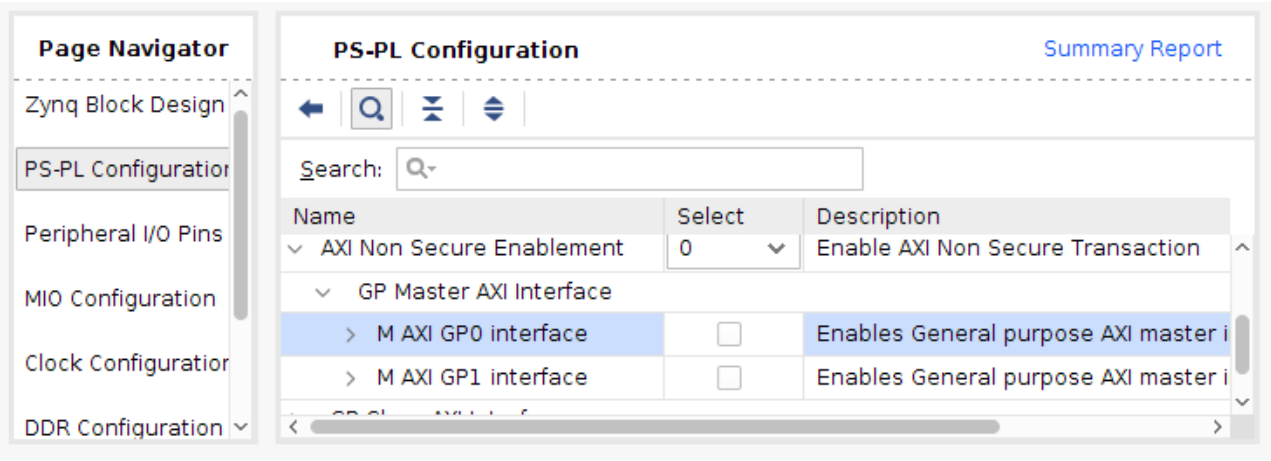


Note: There are few cases in which the FCLK_CLK0 is not used, this case is one of them. In the next labs we will be using FCLK_CLK0 and we will activate it.

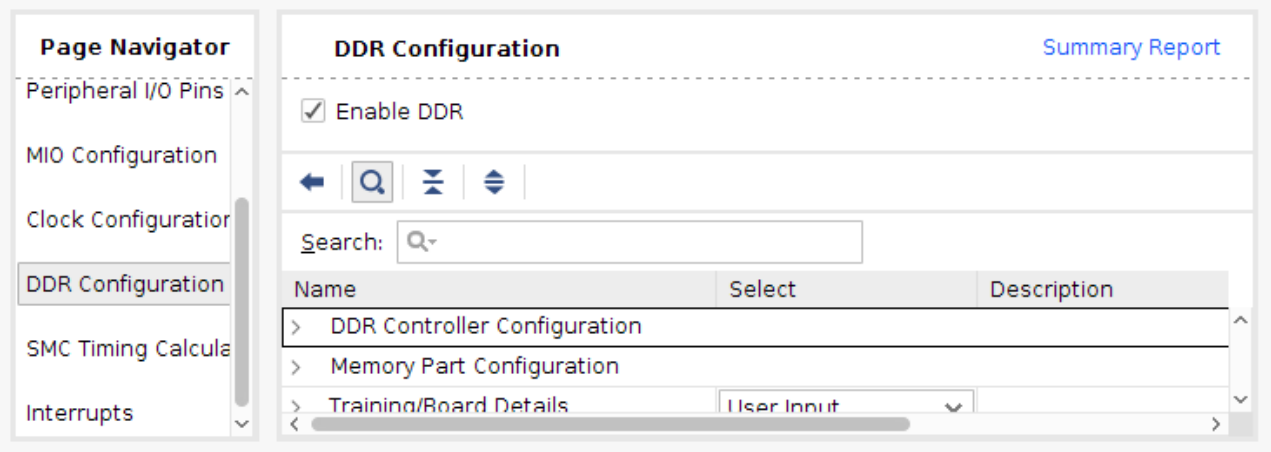
12. Next, click on the **PS-PL Configuration** option, expand **General**, then **Enable Clock Resets,** then de-select **FCLK_RESET0_N.**



13. Next, in the same **PS-PL Configuration** options (just below of **Enable Clock Resets**), expand **AXI Non Secure Enablement,** then expand **GP Master AXI Interface** and de-select **M AXI GP0 Interface.** Again, this is a special case where there is no logic in the PL, therefore there is no need for PS-PL interface, we will use this interface in upcoming labs.

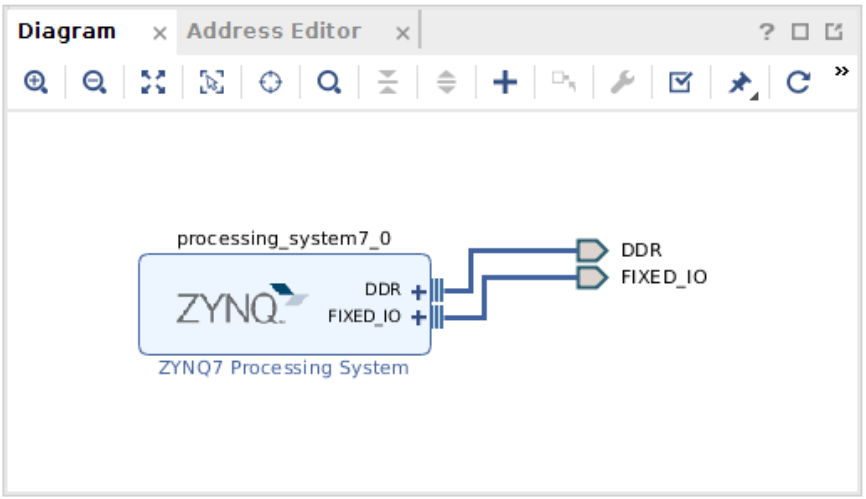


14. In the **DDR Configuration** option, be sure that the **Enable DDR** configuration is selected.



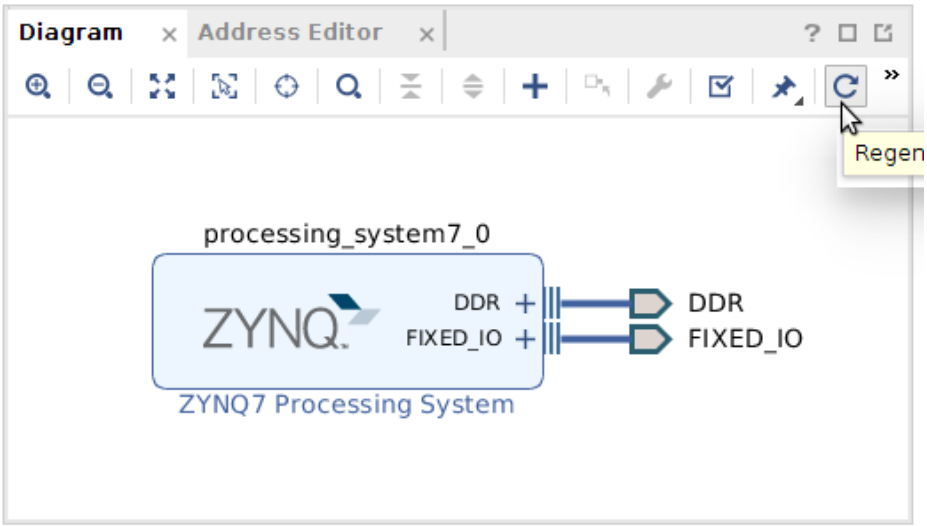
15. Click **OK** to exit the **PS** customization window.

16. Back in the block design canvas of the project, you will notice that the **ZYNQ7 Processing System** block diagram has been simplified due to de-selecting the options mentioned earlier (comparing with figure in step 8).



Notice that the **FIXED_IO** and the **DDR** are now connected to output ports (they are connected to the peripherals and to the DDR memory). This automatic connection is due to the fact that this project was configured initially targeting the ZedBoard, therefore Vivado Design Suite knows which are the Zynq I/O Pins connected to peripherals and memories.

17. To get a nicer block diagram, you can click on the Regenerate Layout Icon



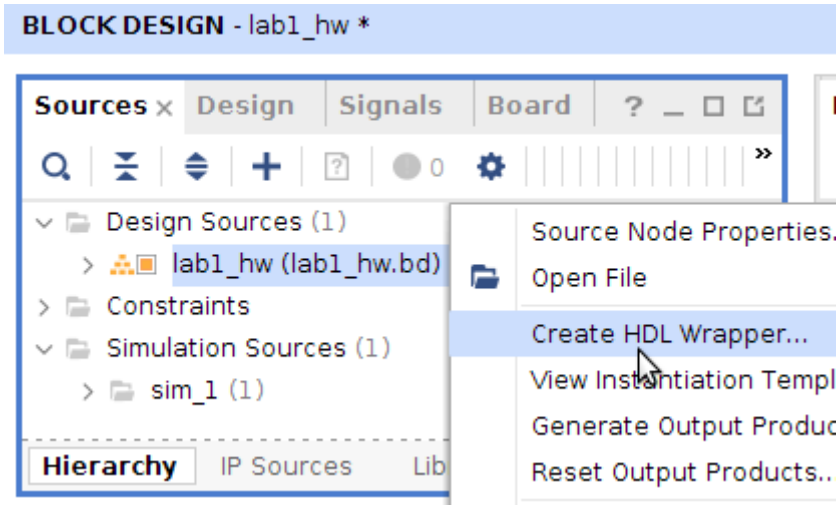
This icon will be very useful when building complicated block designs.

18. Click the **Save** button to save the current block diagram.

19. Select **Tools-> Validate Design**, or click in the icon, to check any possible error while creating the block diagram. There should be no warnings or errors.

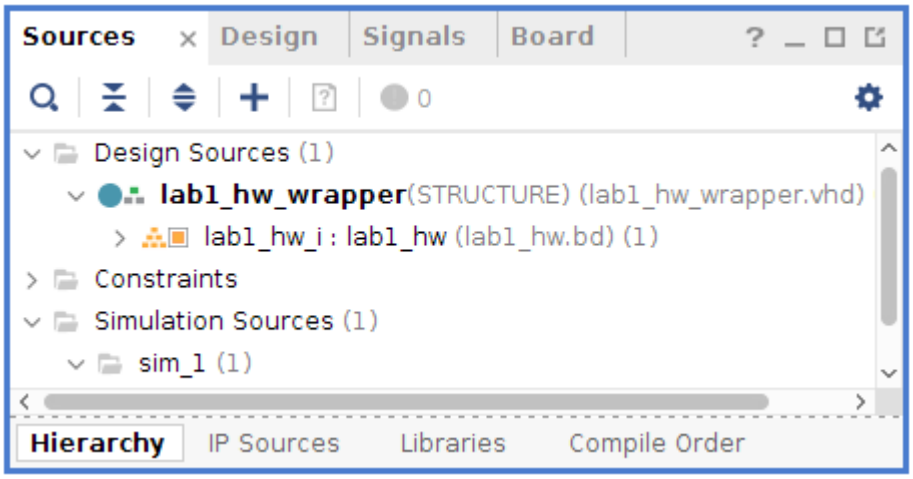
20. In the central pane, click on the **Sources** tab. The **lab1_hw.bd** (board design) file contains all the settings and configuration of the block diagram created in the block diagram editor window. Right click on **lab1_hw.bd** and select **Create Output Products**. Leave the options as defaults, then click **Generate**. Click **OK** in the upcoming window.

21. Right click on **lab1_hw.bd** and select **Create HDL Wrapper** to create the top level VHDL/Verilog file from the block diagram.



In the **Create HDL Wrapper** window make sure you select the "**Let Vivado manage wrapper and auto-update**" option to generate the VHLD/Verilog file. This will let Vivado update the HDL wrapper when you change the block diagram design. Click **OK** to generate the wrapper.

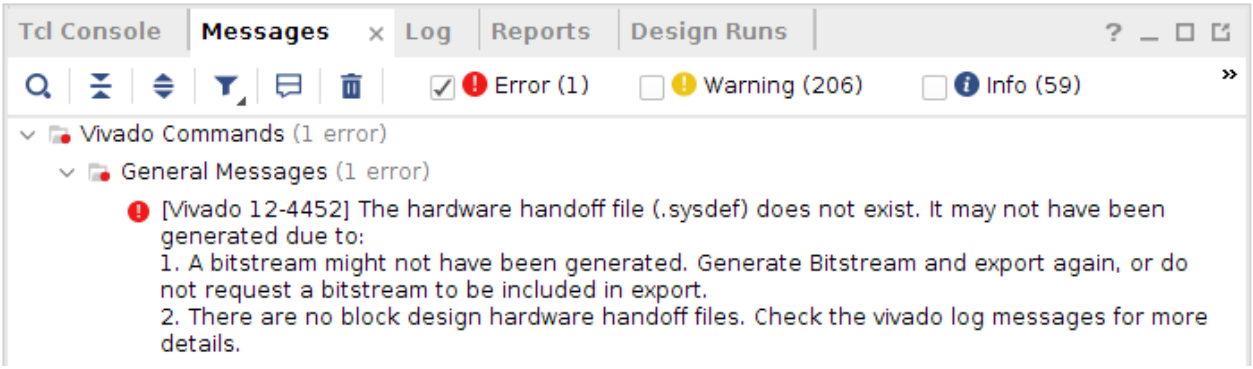
22. Notice that **lab1_hw_wrapper.vhd** (or .v, depending on the HDL type selected during project definition) was created and placed at the top of the design sources hierarchy.



The **Zynq PS Hardware** has now successfully generated. Since there are no additional HDL sources to add to the design, the hardware can be exported directly to the **SDK**.

23. Click **File** > **Export** > **Export Hardware**. The **Export Hardware** dialog box opens. Leave the **Include Bitstream** box unchecked since there is not logic in the PL to be configured. Click **Save** in case you are asked. Also, leave the **Export** option with the default directory that is showed. Click **OK** to continue.

Note: you might get an error message similar to this:



This error is to the fact we have not generated a bitstream file (file that is used to program the PL part of the Zynq), since in this particular lab we do not use the PL, we do not generate the bitstrea, therefore, we can skip this error message.

24. Click **File** > **Launch SDK.** The **Launch SDK** dialog box opens. Click **OK** to continue (leave the default values). If prompted, click Save to save the project. Important: keep open the block design while lunching **SDK**.

Create an Application Project in SDK

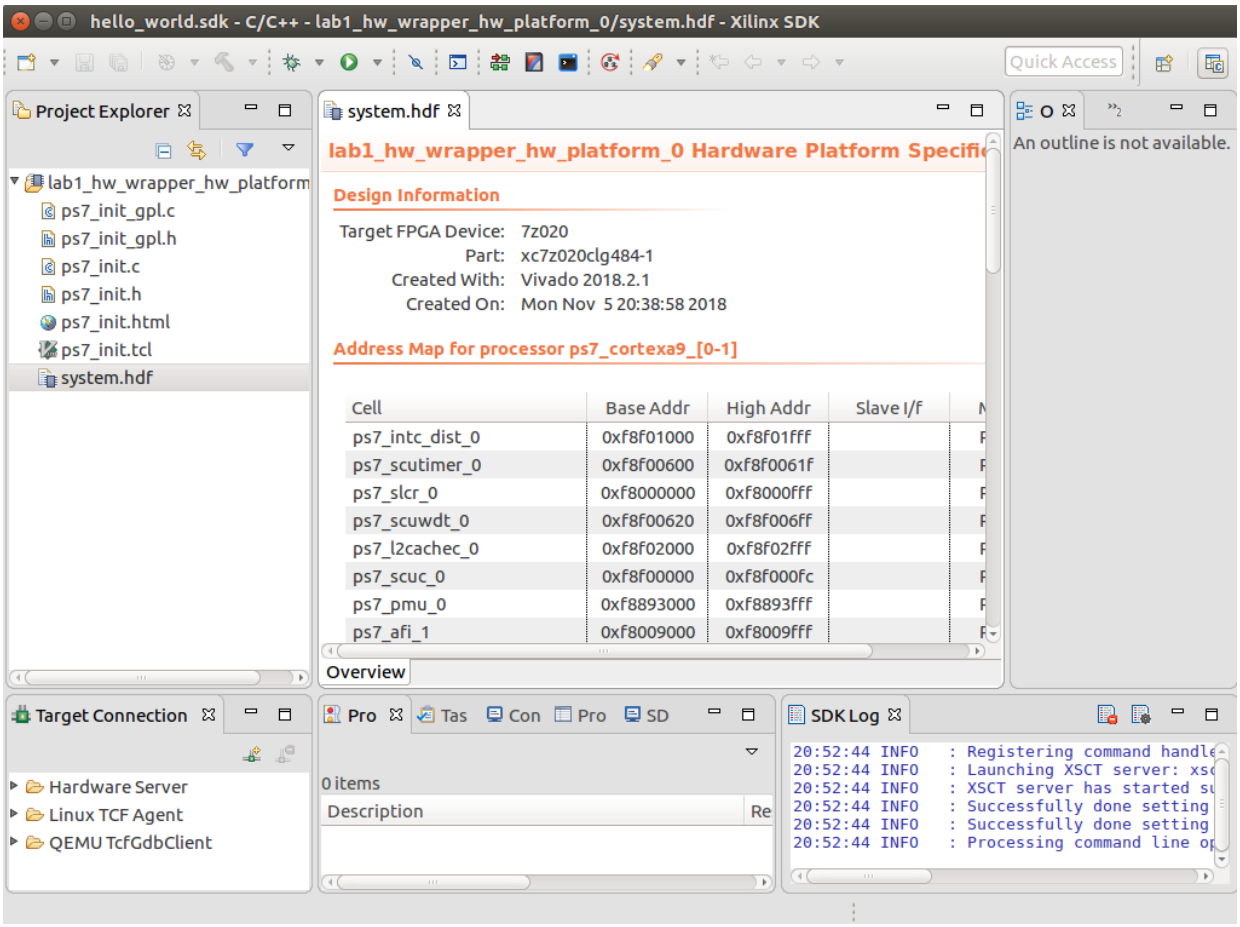
Objective Create a 'Hello World" application in SDK.

1. Let's review what we just have done:

   The Vivado design tool exported the Hardware Platform Specification of the project design (**system.hdf** in this case) to the SDK. In addition to **system.hdf**, there are four more files relevant to SDK got created and exported. They are **ps7_init.c, ps7_init.h, ps7_init.tcl,** and **ps7_init.html**.

   The **ps7_init.c** and **ps7_init.h** files contain the initialization code for the Zynq Processing System and initialization settings for DDR, clocks, PLLs, and MIOs. SDK uses these files to initialize the processing system (PS) so that the applications can be run using the configured processing system peripherals.

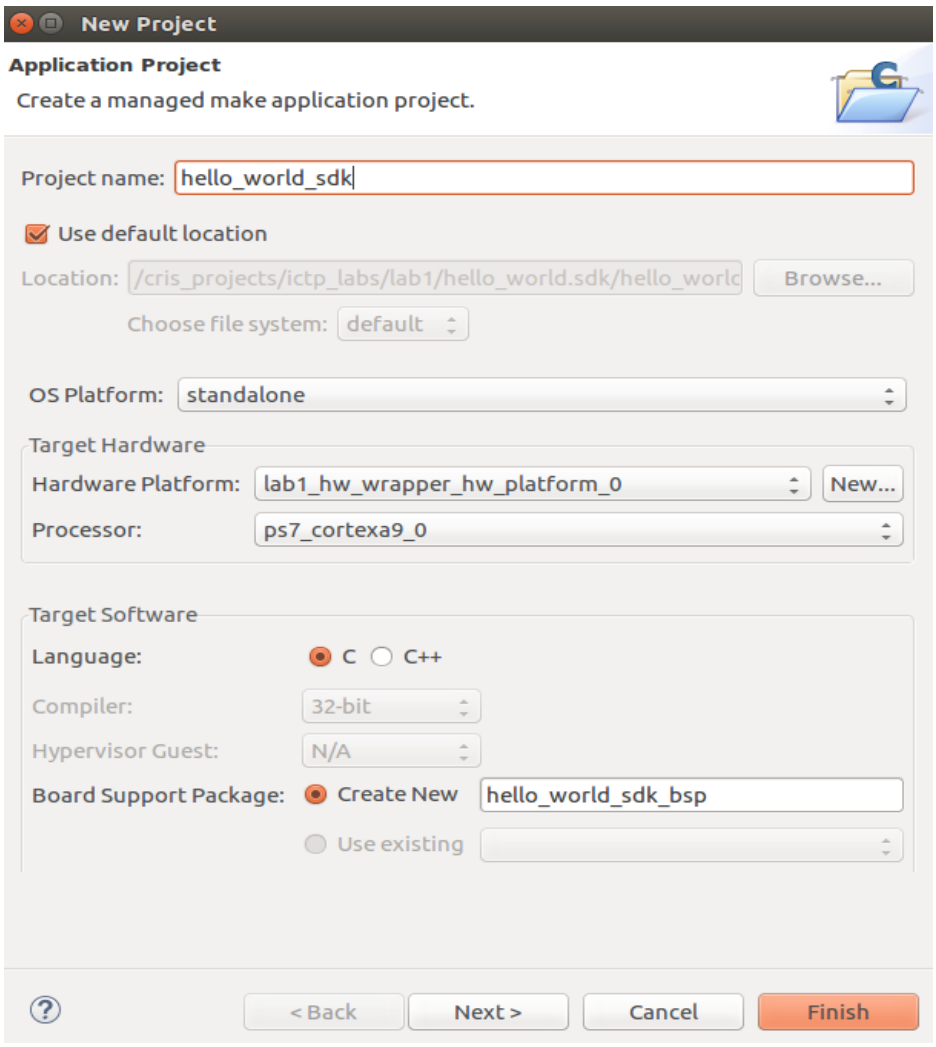
   The **system.hdf** file opens by default when SDK is launched. The address map of your system read from this file.

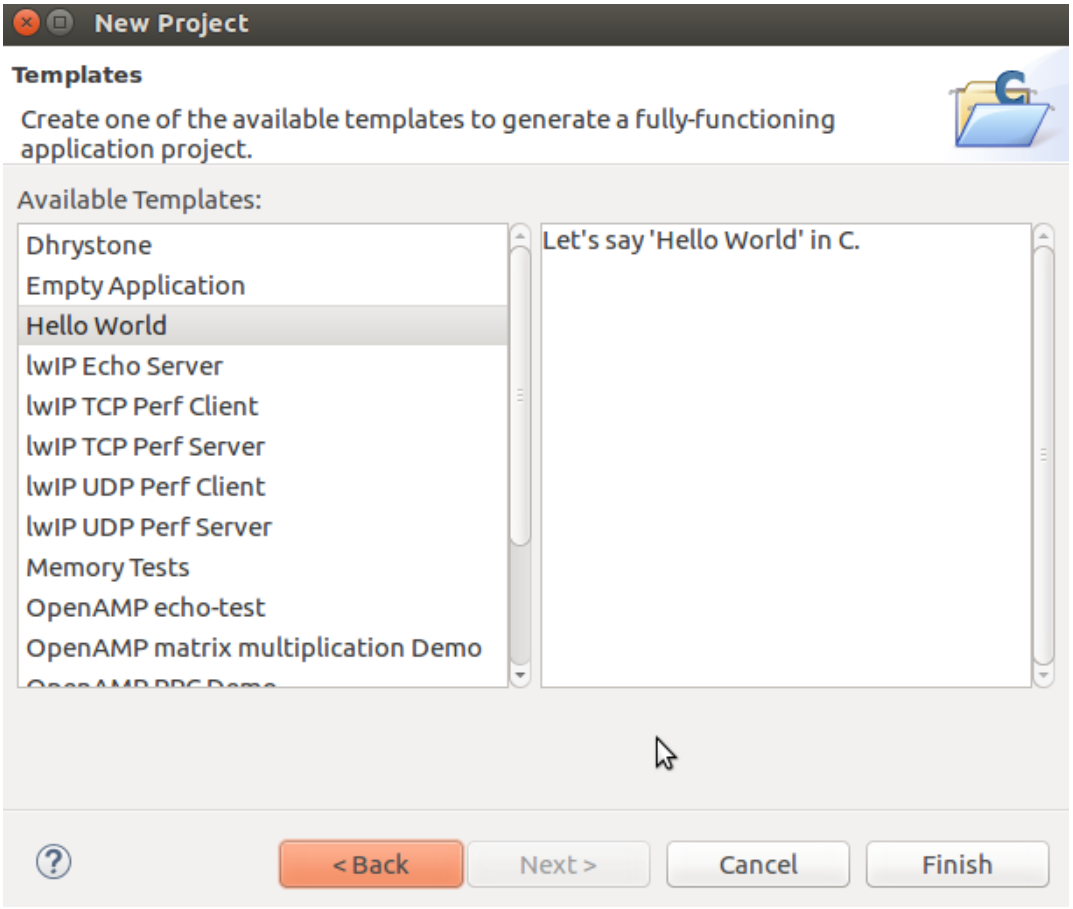


2. Select **File -> New -> Application Project.** To create the new application use the following information:

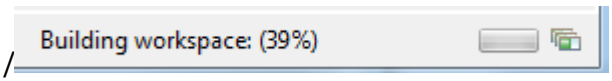
Wizard Screen	System Property	Setting
New Application Project	Project Name	hello_word_sdk
	Used default location	Check this option
	OS Platform	Standalone
	Hardware Platform	system_wrapper_hw_platform_0
	Processor	ps7_cortexa9_0
	Language	C
	Board Support Package	Create New: hello_world_bsp
Click **Next**		
Templates	Available Templates	Hello World

3. The following figures details how to write the respective settings:



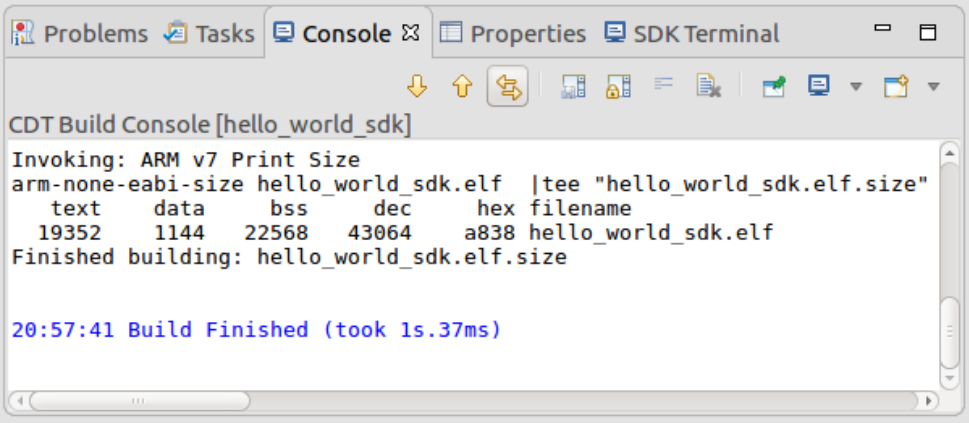


4. After clicking the **Finish** button, **SDK** automatically begins building the application. On the bottom left part of the main window there is a small indicator about this process.

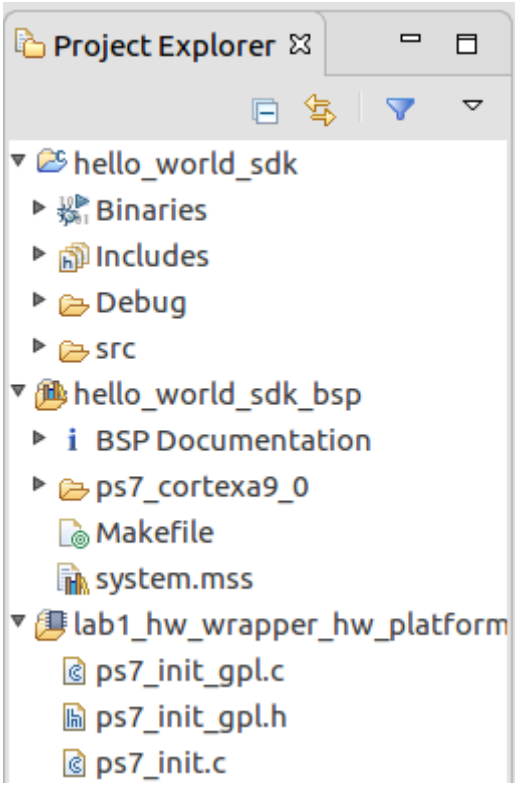


5. In this step the **Board Support Package (BSP)** and the necessary 'C' code for the 'Hello World' has been created. BSPs contain drivers, libraries, and essentially anything else, which will allow the software applications to access features on the hardware.

6. While the application is being built, watch the messages in the **Console** window. When the project is successfully built, the following message should be read "**Build Finished".** Hence, the application and its BSP are both compiled and the *.elf is generated (*.elf is the file that will be downloaded into the PS memory to execute the 'C' file). The information provided by *.size gives an idea of the code size.



7. In the Project Explorer pane, on the left side of the screen, you will now see three folders: hello_world, hello_world_bsp, and the lab2_wrapper_hw_platform. The hello_world folder contains the necessary files for the application itself. For instance, expand the folder, and find the src sub-folder, then double click on the helloworld.c file. Tthe respective 'C' code will be displayed in the edit window. This file contains the main( ) function. Likewise, expand the "hello_world_bsp" folder, and continue to expand the yellow folders until you see the "include" folder. Open this folder and you will see a list of header (.h) files which holds all of the drivers that you will need to develop your software application.

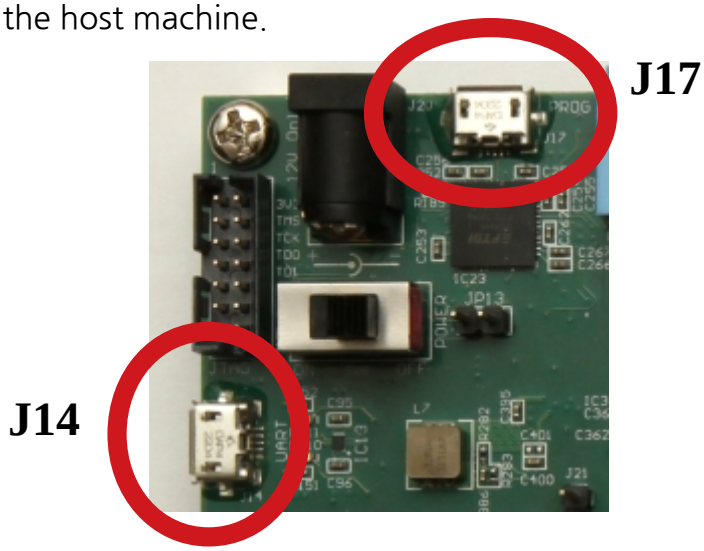


Connecting the ZedBoard

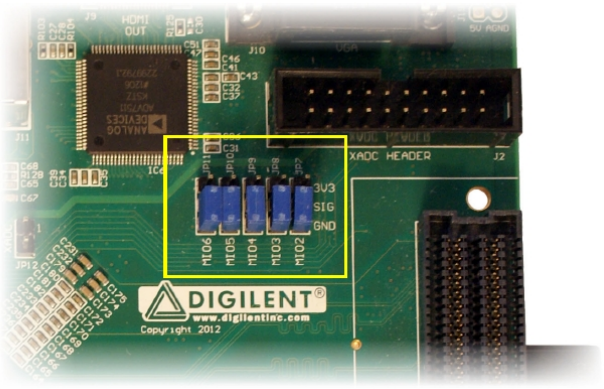
Objective Steps to connect the ZedBoard to run the application.

1. Connect a USB micro cable between the Windows/Linux Host machine and the **ZedBoard JTAG J17** (on the right side of the power connector). This connection will be used to configure the PL (in this particular Lab there is no need for this connection, since there is not PL configuration file, bitstream file, but in all the other Labs this connection will be used).

2. Connect a USB micro cable to the **USB UART** connector (**J14**) on the ZedBoard (on the left side of the power switch), with the Windows/Linux Host machine. This connection will be used to carry out the serial message/data transfer between the ZedBoard and the host machine.

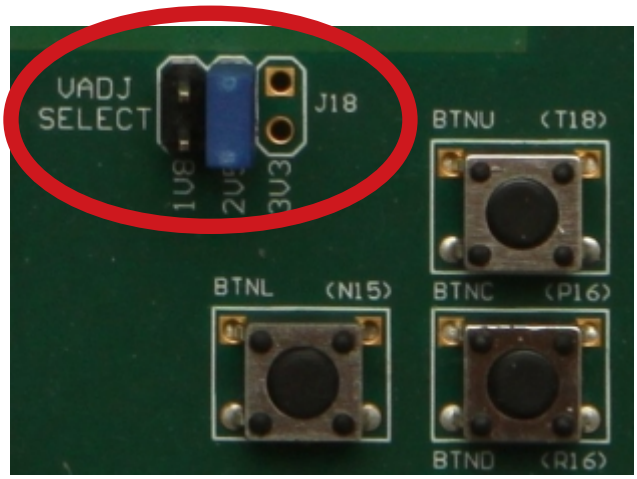


IMPORTANT 1: Ensure that jumpers **JP7** to **JP11** are set as shown in the figure below for the JTAG configuration mode.



IMPORTANT 2: be sure to already have installed the Cypress device driver for the USB-UART chip on the ZedBoard.

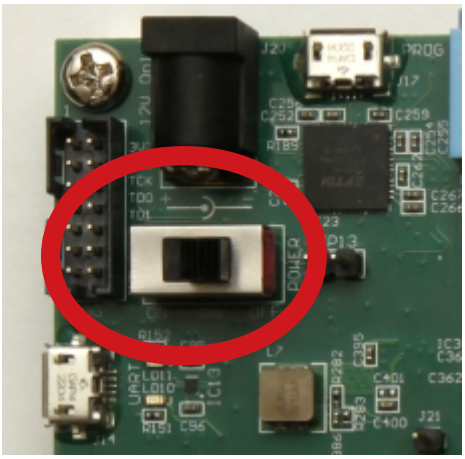
3. Check the jumper setting for **J18** in the bottom right corner of the board. The jumper should be set to 2.5V, which is marked as "2V5" on the board.



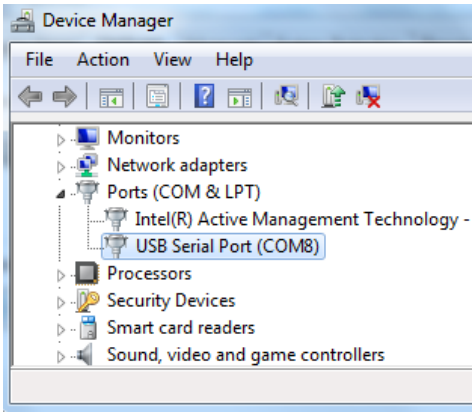
4. Connect the 12V AC/DC converter power cable to the ZedBoard barrel jack.

5. Power-on the board using the **ZedBoard Power** switch. Check that the **Power LED** on the board (green LED) is on. Note, in some instances the board needs to be ON before the SDK launches in order for the SDK to see which COM port is being used by the OS.

Power Switch



6. To find out which COM port has been assigned to the UART connection, use the **Control Panel->Device Manager** Windows utility. An example is shown as follow.

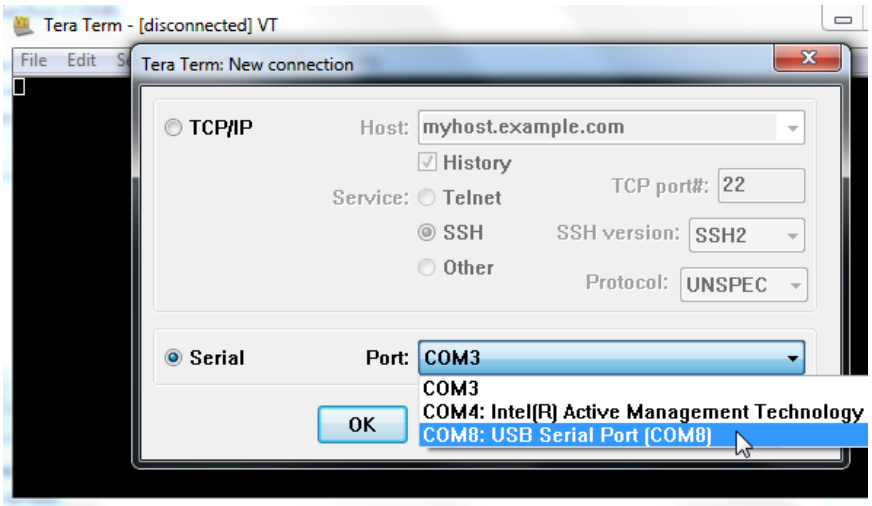


7. Setting up the serial utility software:

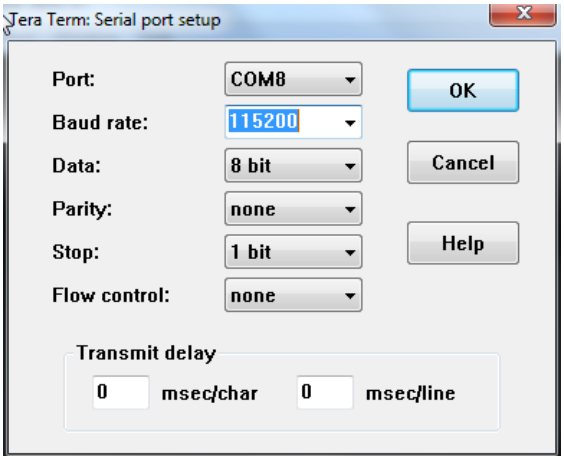
7.1. Use a utility program such as **Tera Term** or **Putty** to setup a serial communication between the Host and the ZedBoard, by using the Host COM port.

For the Tera Term configuration, first select Serial as communication protocol, and from the pull down menu select the Port to be used.

Important: In case that there is no "USB Serial Port ……" option from the Port pull down menu, first check that the board is On, then check whether the serial port is detected by the operative system, e.g. in Windows use the Device Manager utility. In case that no serial port is detected, re-install the Cypress device driver for the USB-UART chip on the ZedBoard.



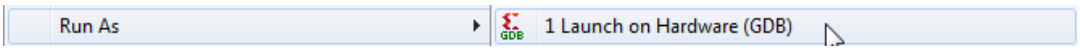
7.2. Configure the Serial port of the Tera Term or the software you are using, with the values detailed in the following figure.


Tera Term: Serial port setup
Port:
COM8
Baud rate:
115200
Data:
8 bit
Parity:
none
Stop:
1 bit
Flow control:
none
OK
Cancel
Help
Transmit delay
0
msec/char
0
msec/line

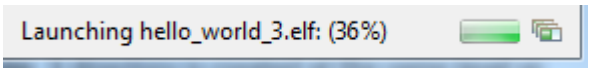
Executing the Application in the ZedBoard

Objective Executing the 'C' application in the Zynq.

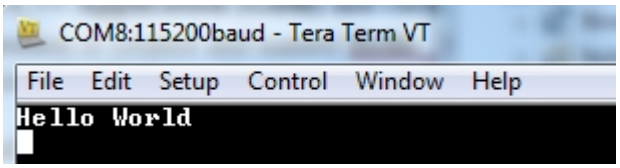
1. On the **Project Explorer** pane, select the **hello_world** application, and then right click mouse and select **Run As -> Launch on Hardware (GDB).**



2. On the bottom right part of the main window there is a bar indicating that the programming file, .elf, is being downloaded into the internal memory.



3. Once it is done, the program will be automatically executed. Hence, in the Tera Term window the "**Hello World**" message should be displayed.



Some Points to Keep in Mind

1. For this particular application, since there is no hardware in the PL, there is no need of configuring the PL. Hence, no bitstream download is required.
2. All the initialization routines needed for the ARM Cortex-A9 are automatically created, there is no need for the designer to create them.
3. The Board Support Package (BSP) is the support code for a given hardware platform or board. It initializes the board, ZedBoard, at power up to allow the software applications execute on the platform. It can be specific to some operating systems with bootloader and device drivers.
4. Standalone applications do not utilize an Operating System (OS). They are sometimes also referred to as bare-metal applications. Standalone applications have access to basic processor features such as caches, interrupts, exceptions as well as other simple features specific to the processor. These basic features include standard input/output, profiling, abort, and exit. It is a single threaded semi-hosted environment. Almost all the Labs of this part of the Workshop will be standalone applications.

Challenges

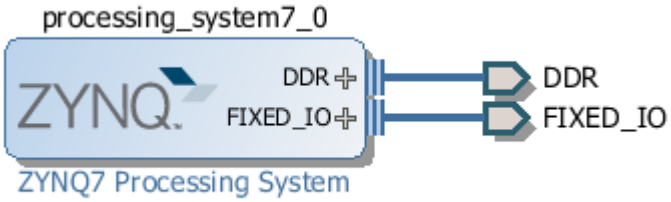
1. Modify the 'C' code of the main function to use the printf function instead of print. Find out how much memory bits are needed in each case.
2. Modify the 'C' code of the main function to use the xil_printf function instead of printf. Find out how much memory is needed for the original code and how much is needed for the code with the xil_printf function.
3. Use the optimization per size option of the C/C++ Build Settings. Repeat the previous two processes and find out the respective sizes.
4. Execute the 'C' code in debug mode. Which are the function invoked by the init_platform function? And which are the function invoked by the cleanup_platform function?

Section II

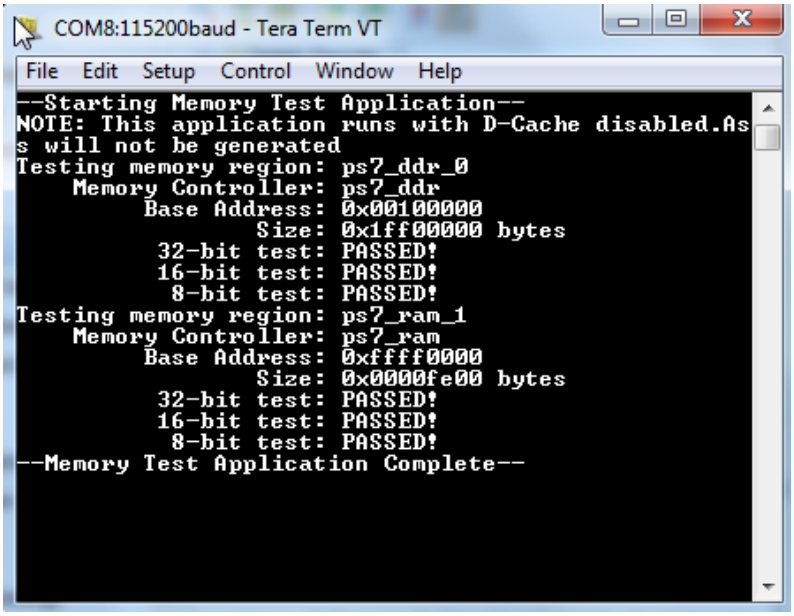
Memory Test Application

Objective Create a new PS7 based design in Vivado, then a 'C' application based in the pre-defined memory test.

1. Follow the steps detailed in the **Section I** of this lab to build a PS7 based block design similar to the one shown in the figure below. Name the design as **"lab1_mem_test.**



2. Create the wrapper.
3. Generate the block design.
4. Export the hardware design.
5. Lunch the SDK.
6. In the **SDK** environment create a new application project with the name **"lab1_memory_test_sdk"**
7. In the templates window select the "**Memory Test**" option.
8. Follow the necessary steps to connect and power the board as it was explained before.
9. Execute the 'C' code (as it was explained before).
10. Open and configure the serial terminal (Putty, TeraTerm, etc).
11. You should obtain a result window similar to this one:



12. Execute the 'C' code step by step. Go into the for-loop to try to understand the different functions executed inside the loop.

13. Insert a **print** or **xil_printf** to print out your name and lab number in the result window.

14. Try to find out the size of the 'C' code.

Conclusion

First steps to create a simple design of the Zynq. Only the PS part is used in this lab. The PS was added and configured according to the requirements. An SDK based application project was created and a simple 'C' code executed to display the 'Hello World' phrase on the UART terminal software.