

---

## *LABORATORY 2*

# *Vivado Design Flow for a PS Simple Design*

---

*Hello World*  
*Memory Test*

# Vivado Design Flow for PS

## Introduction

This lab guides you through the process of using Vivado IDE to create a simple SoC design targeting 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. You will 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:

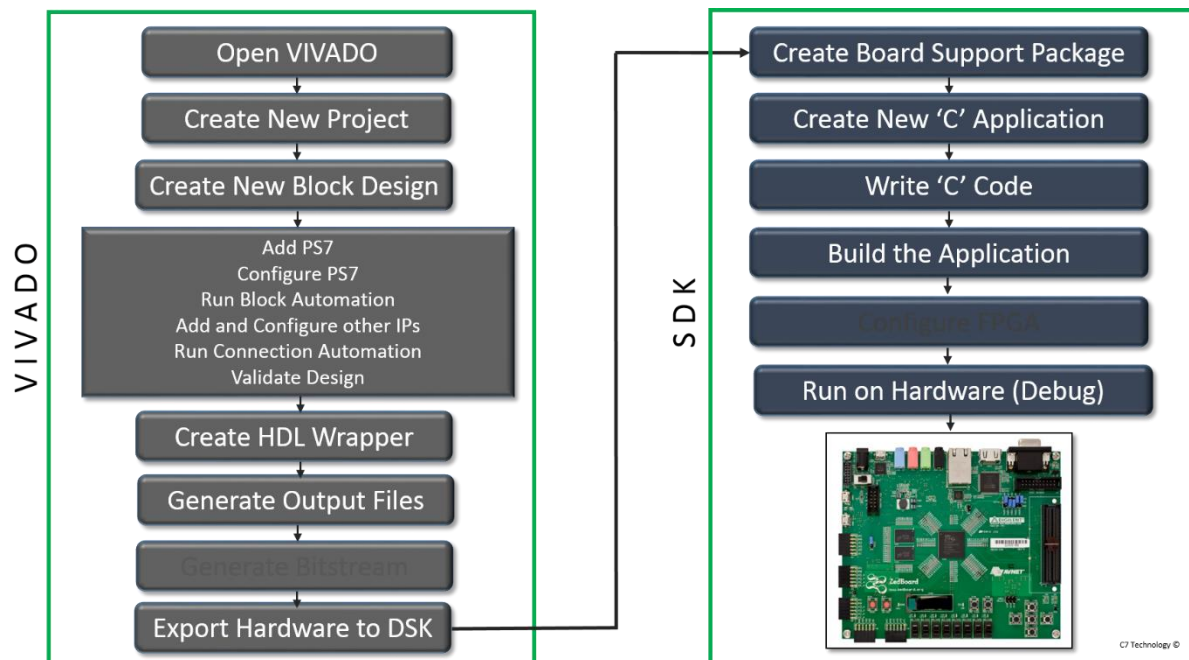
- Create a Vivado project based in the IP Integrator
- Add and configure the PS7
- Generate HDL wrapper
- Export the design to SDK
- Create an application project in SDK, creating a board support package and 'C' code
- Configure the UART to communicate the PC with the ZedBoard
- Program the PS7 using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

## General 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 process in the above flow towards this Lab:

1. The recommended design and implementation flow begins with launching *Vivado*, which is the central cockpit from which design entry through bitstream generation is completed.
2. Open the *Create New Project* 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.

## PART I – Hello World

### Create a Vivado Project using IDE

### Step 1

**1-1. Objective:** Launch Vivado and create a project with an embedded processor system targeting the ZedBoard.

**1-1-1.** Open *Vivado Design Suite*.

**1-1-2.** Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**. Use the information in the table below to configure the different wizard option:

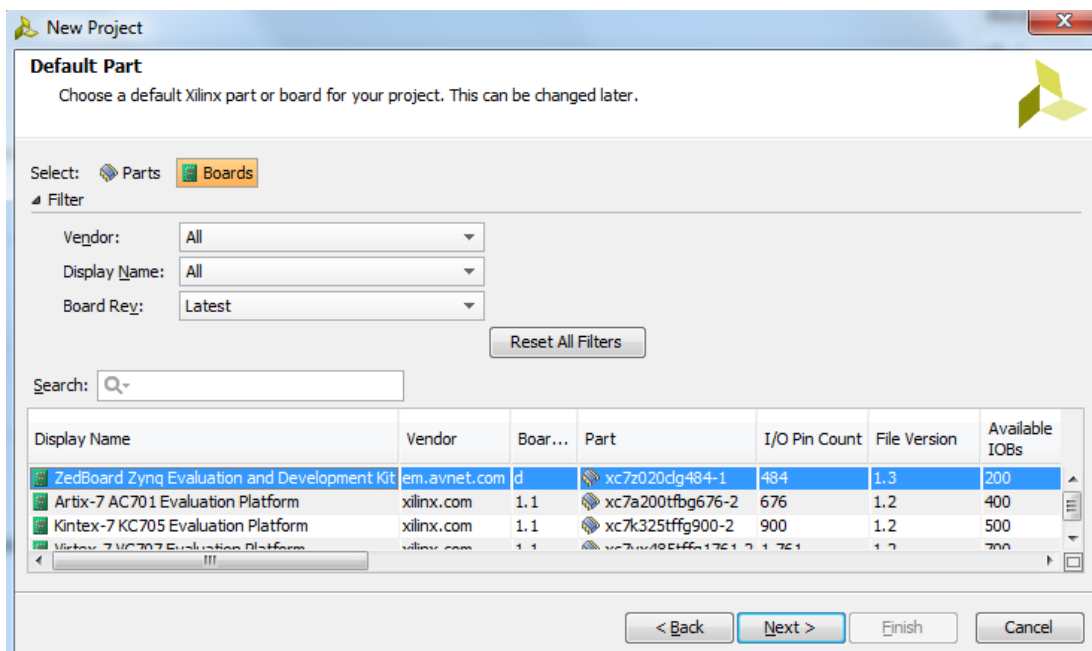
Wizard Option	System Property	Settings
Project Name	Project Name	Lab2
	Project Location	C:\.....\SoC_School\labs\lab2
	Create Project Subdirectory	Check this option.
Click <b>Next</b>		
Project Type	Specify RTL	<b>Select RTL.</b> Keep do not specify sources at this time box unchecked
Click <b>Next</b>		
Add Sources	Do nothing	
Click <b>Next</b>		
Add Existing IP	Do Nothing	
Click <b>Next</b>		
Add Constraints	Do Nothing	
Default Part	Specify	Select <b>Boards</b>
	Board	Select <b>ZedBoard Zynq Evaluation and Development Kit, Rev. D.</b>
Click <b>Next</b>		
New Project Summary	Project Summary	Review the project summary
Click <b>Finish</b>		

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

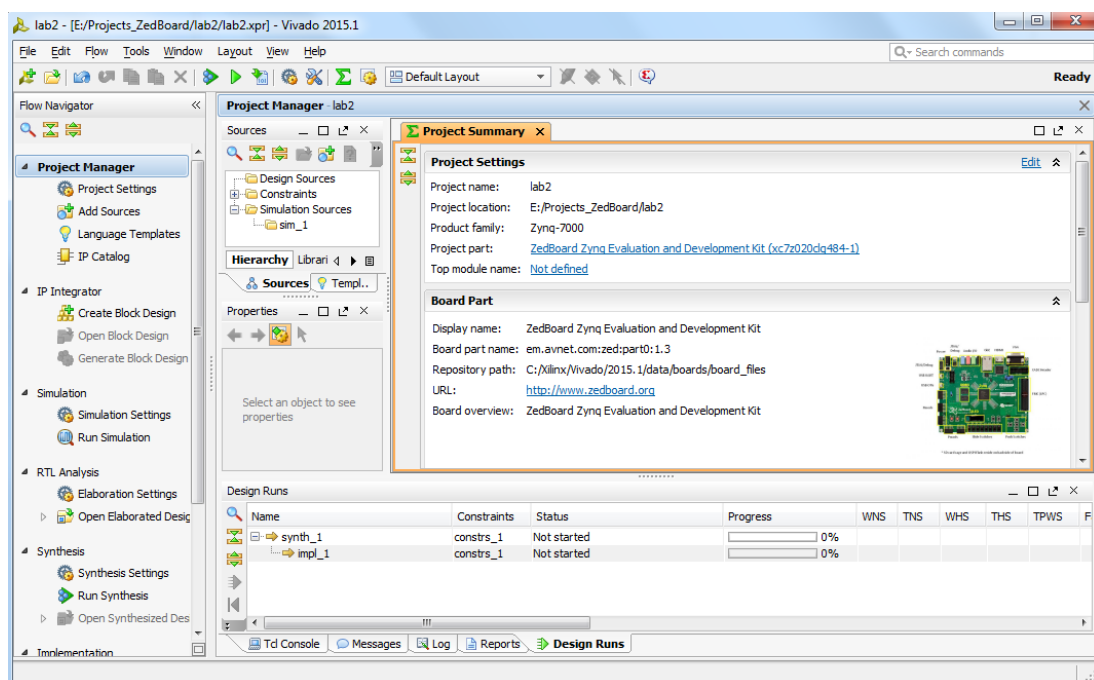
The board selected during the project creation, in this case the ZedBoard, has a direct impact on how the *IP Integrator*, within the Vivado, executes.

*IP Integrator* is board aware and it will automatically assign dedicated PS IO ports to physical pin locations mapped to the specific board peripherals when the *Run Connection* wizard is 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.

The following figure shows the board selection in the wizard:

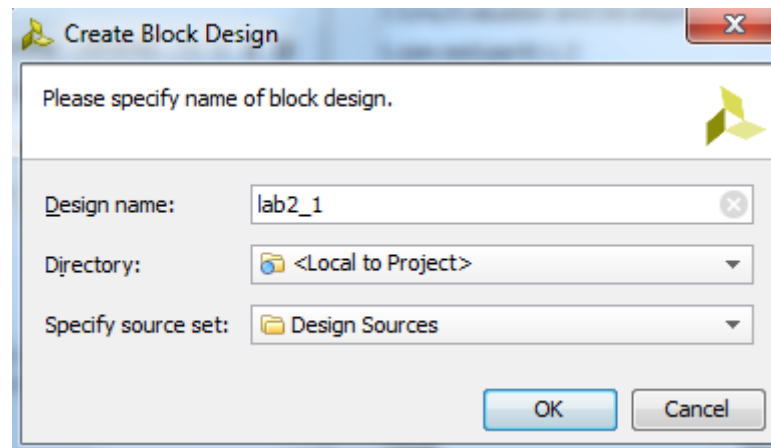


**1-1-3.** The *Vivado Design Suite* main window should look like the following figure:




**1-1-4.** Next step is to use the *IP Integrator* to create an embedded processor project.

**1-1-5.** Click *Create Block Design* in the *Flow Navigator* pane under the *IP Integrator*. Type *lab2* as *Design Name* in the *Create Block Design* window.

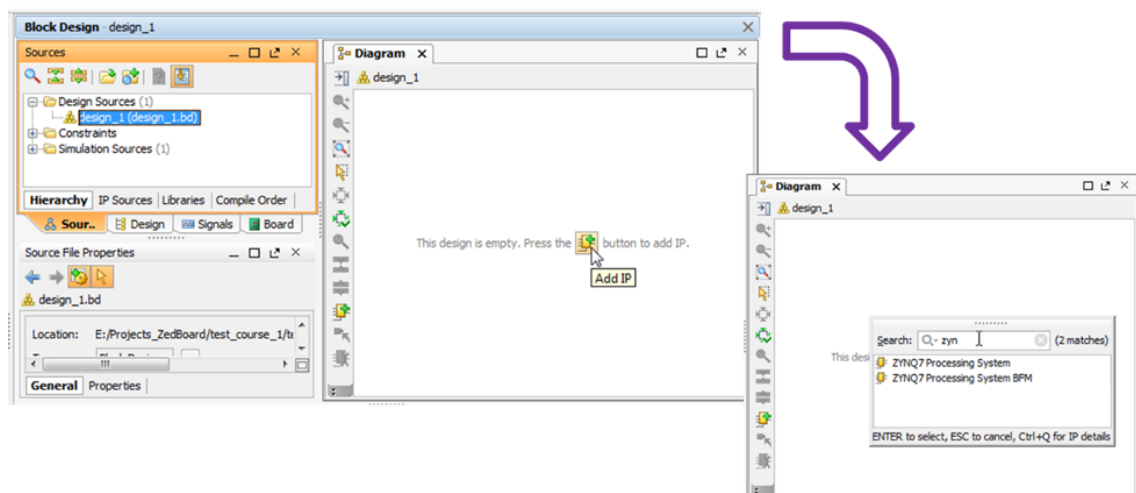


1-1-6. A new blank *Block Diagram* canvas will be presented.

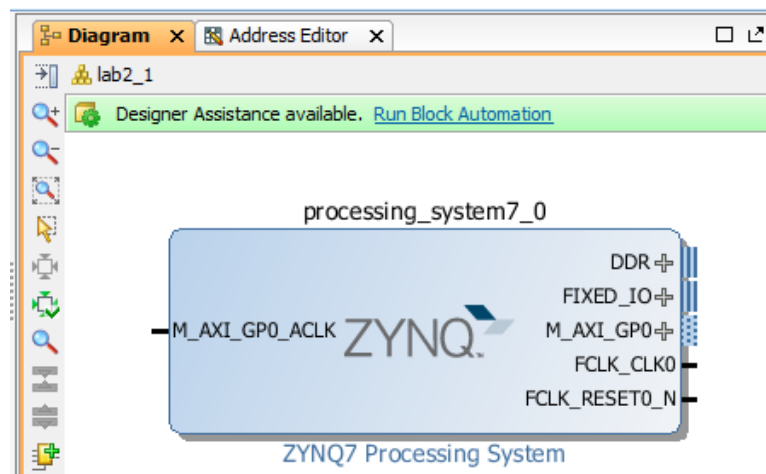
1-2. **Objective: Create an Embedded System via IP Integrator.** You can design a new embedded system in Vivado using IP Integrator by adding a ZYNQ7 Processing System block. By adding this block, you can configure one of the ARM Cortex-A9 processor cores for your application. You can also place additional IP blocks to further the capabilities of the embedded system.

1-2-1. To insert a *Processing System (PS)* block either click the *Add IP* icon  or right click on the canvas blank space and select *Add IP* from the available options.

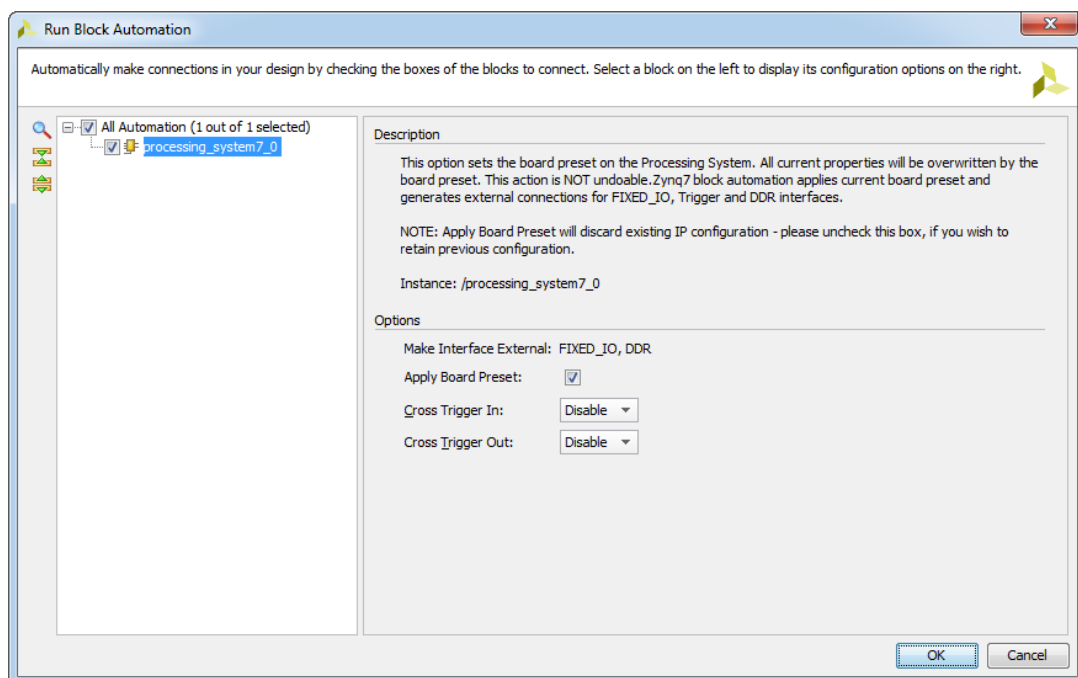
1-2-2. In the upcoming available IPs window, either scroll down to the very bottom of the list or search using the keyword *zynq*, then double click on the *ZYNQ7 Processing System*.



1-2-3. The *Zynq7 PS* block is placed in the block diagram canvas. The I/O ports shown in the block diagram are defined by the default settings for this block as specified by the target development board (in this case the ZedBoard).



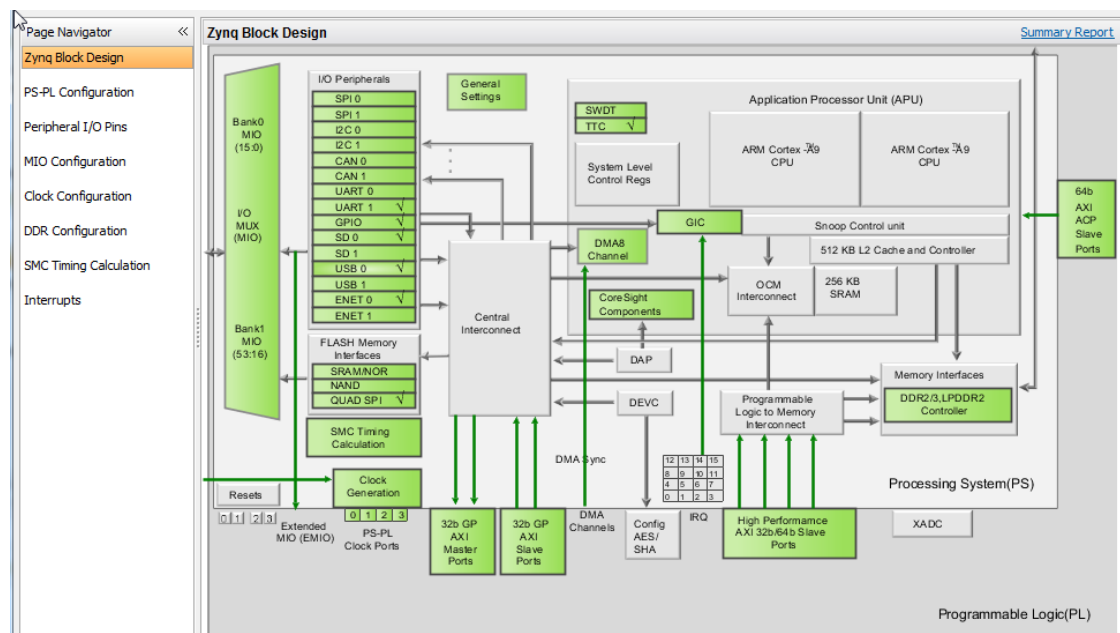
- 1-2-4.** Click **Run Block Automation**, available in the green information bar. Then, select `/processing_system7_0`. Make sure *Apply Board Presets* is checked and select *OK* in the *Run Block Automation* window (leave everything else as default).



- 1-2-5.** 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).

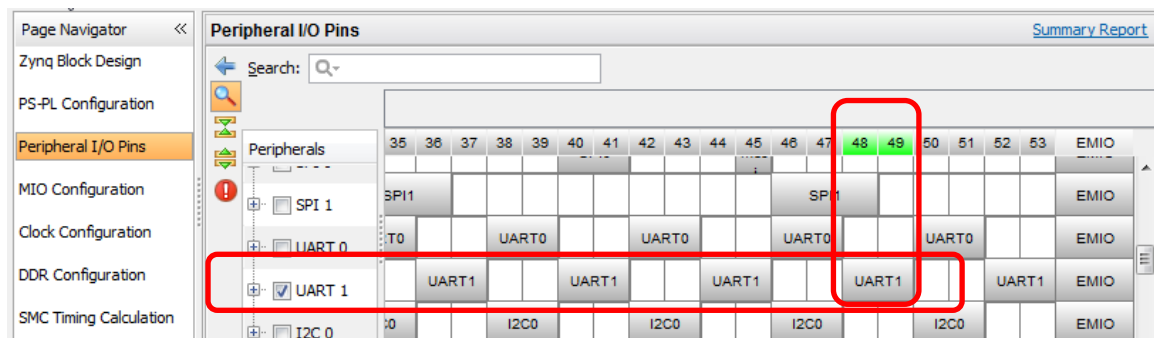




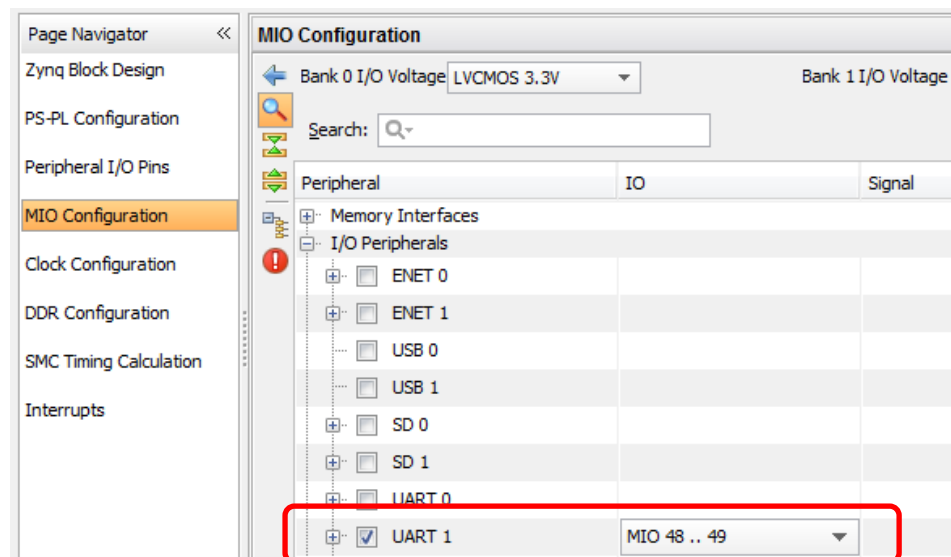
**1-3. Objective:** Customizing the Zynq Processing System settings. For this particular Lab many of the defaults setting of the PS7 will not be necessary, therefore they will be modified in the following steps.

**1-3-1.** Click on the *MIO Configuration* option under the *Page Navigator*. Expand *I/O Peripherals* select *UART1*. The *PS-UART1* will be used to communicate the Zynq device with the PC, by using a serial terminal software as STDIO, like Putty.

There are two ways of selecting the UART1 peripheral. One way is in the *Peripheral I/O Pins* options, as it is shown in the next figure.

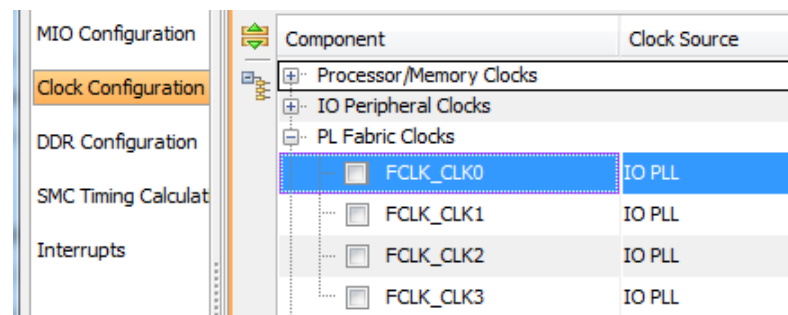


The other way is selecting the *UART1* from the *MIO Configuration*. As it is showed in the next figure.

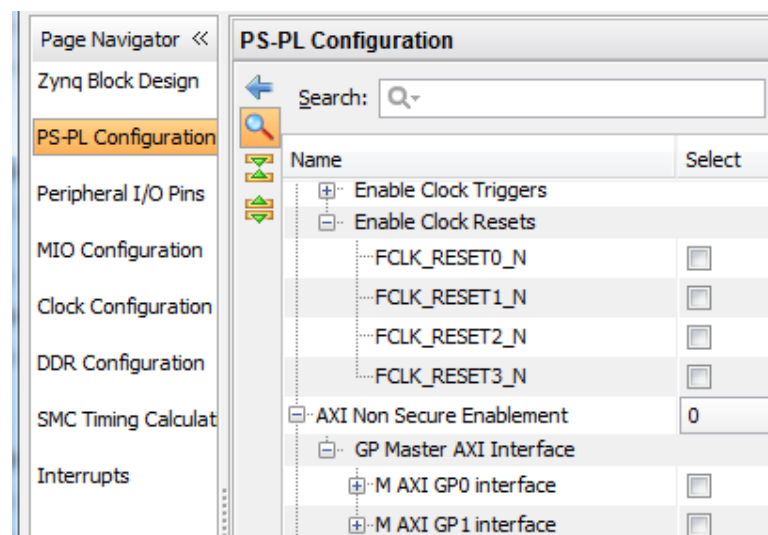


Either way could be used for the purpose of enabling *UART1*.

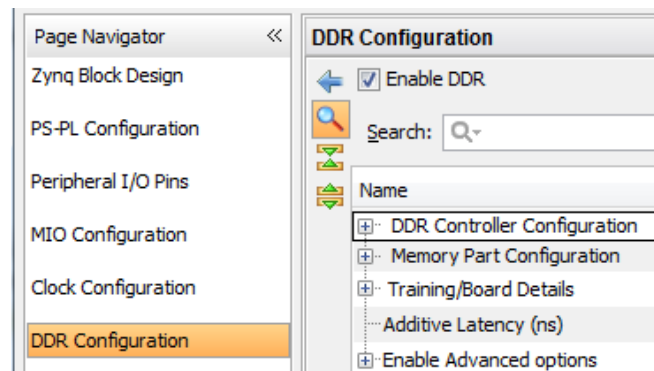
- 1-3-2.** Next, click on the *Clock Configuration* option, expand *PL Fabric Clocks*, and de-select *FCLK\_CLK0*. Since there will not be any logic in the PL part of the Zynq it not necessary to supply any clock to the PL.



- 1-3-3.** Next, click on the *PS-PL Configuration* option, expand *General*, then *Enable Clock Resets*, then de-select *FCLK\_RESET0\_N*. Also, expand *AXI Non Secure Enablement*, then expand *GP Master AXI Interface* and de-select *M AXI GP0 Interface*. Since there is no interface between the PS-PL, it is not necessary any AXI interface.

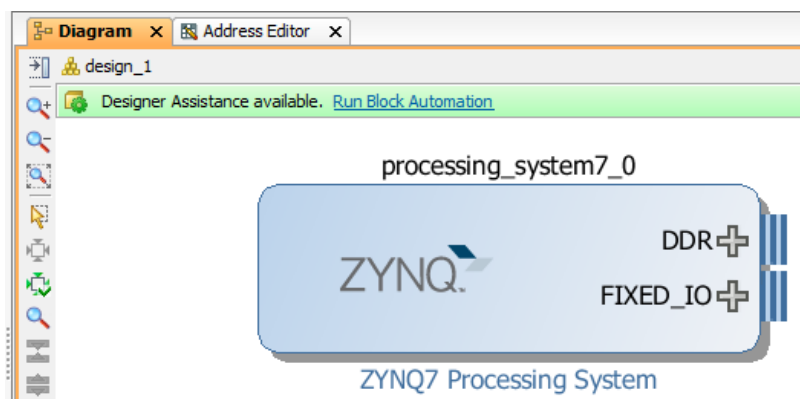


- 1-3-4.** In the *DDR Configuration* option, be sure that the *Enable DDR* configuration is selected.

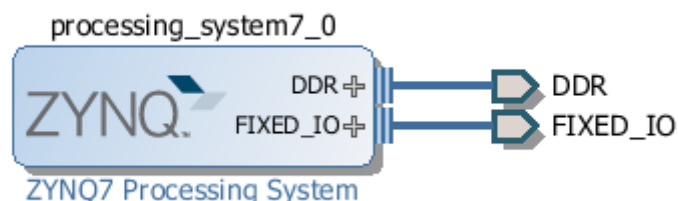


**1-3-5.** Click *OK* to exit the *PS* customization window.


**1-3-6.** 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.



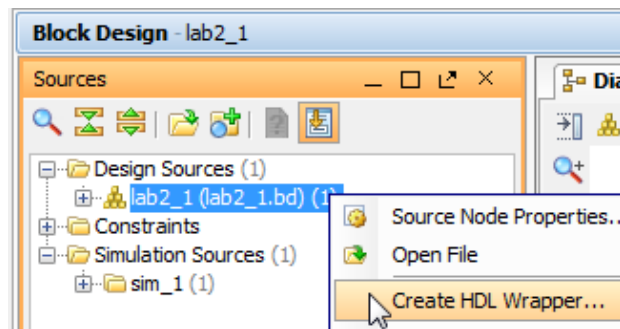
**1-3-7.** Notice that the *FIXED\_IO* and the *DDR* are now connected to output ports (they are connected to the peripherals and DDR memory). This is because this project was configured on the ZedBoard, therefore Vivado knows which are the Zynq I/Os connected to peripherals.



**1-3-8.** Click the *Save* button to save the current block diagram.

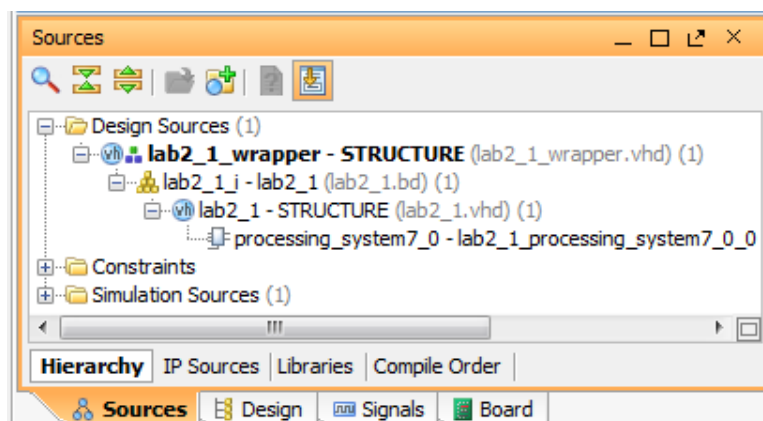
**1-3-9.** Select *Tools-> Validate Design*, or click in the  icon. There should be no warnings or errors.

**1-3-10.** In the *Sources* pane, click on the *Sources* tab. The *lab2\_1.bd* (board design) file contains all the settings and configuration of the block diagram created in the block diagram editor window. Right click on *lab2\_1.bd* and select *Create HDL Wrapper* to create the top level VHDL/Verilog file from the block diagram.

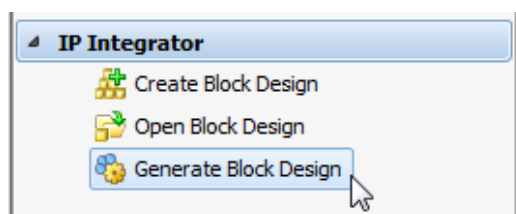


In the *Create HDL Wrapper* window make sure you select the “*Let Vivado manage wrapper and auto-update*” option to generate the VHDL/Verilog file. This will let Vivado update the HDL wrapper when you change the block diagram design. Click *OK* to generate the wrapper.

- 1-3-11.** Notice that *lab2\_1\_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.

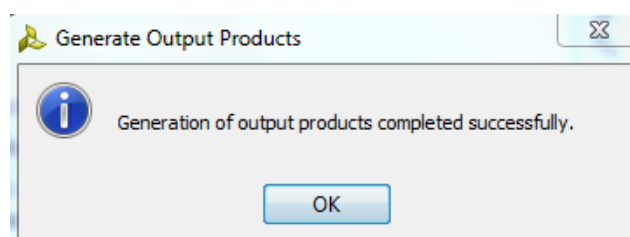


- 1-3-12.** Click on *Generate Block Design* in the *Flow Navigator* to generate the block design.

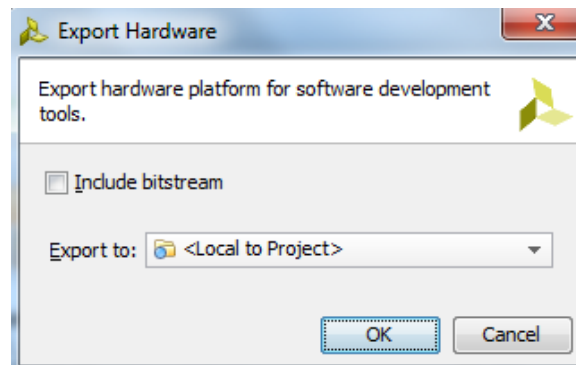


- 1-3-13.** In the *Generate Output Products* window, by default the *Synthesis Option Global* should be selected. Click *Generate*.

- 1-3-14.** The message “*Generation of output products completed successfully*” should be displayed in the *Generate Output Products* window. 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.



- 1-3-15.** 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 to* option with the default directory that is showed. Click *OK* to continue.

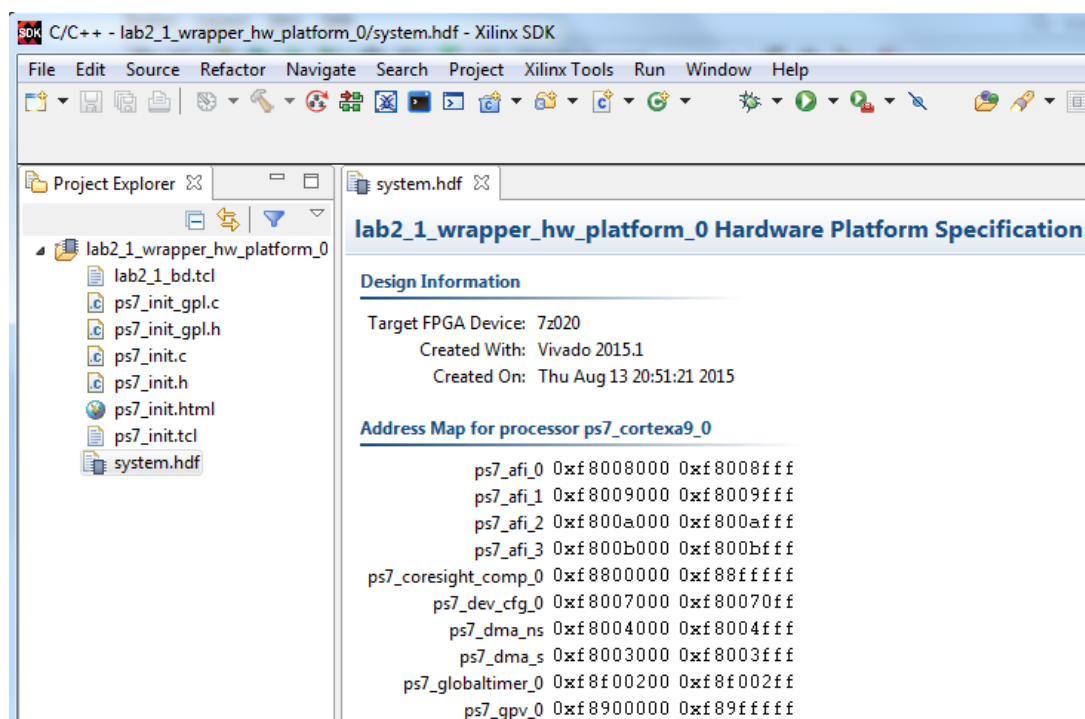


- 1-3-16.** Click *File > Launch SDK*. The *Launch SDK* dialog box opens. Click *OK* to continue (leave the default values). If prompted, click *Save* to save the project. *Important:* keep open the block design while launching SDK.

### What Just Happened?

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*.

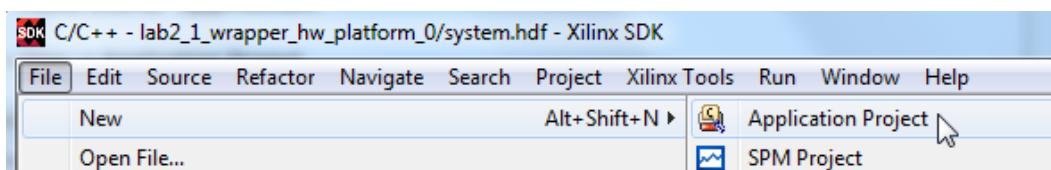
The *system.hdf* file opens by default when SDK is launched. The address map of your system read from this file.



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.

## 2-1. **Objective:** Creating a “Hello World” application.

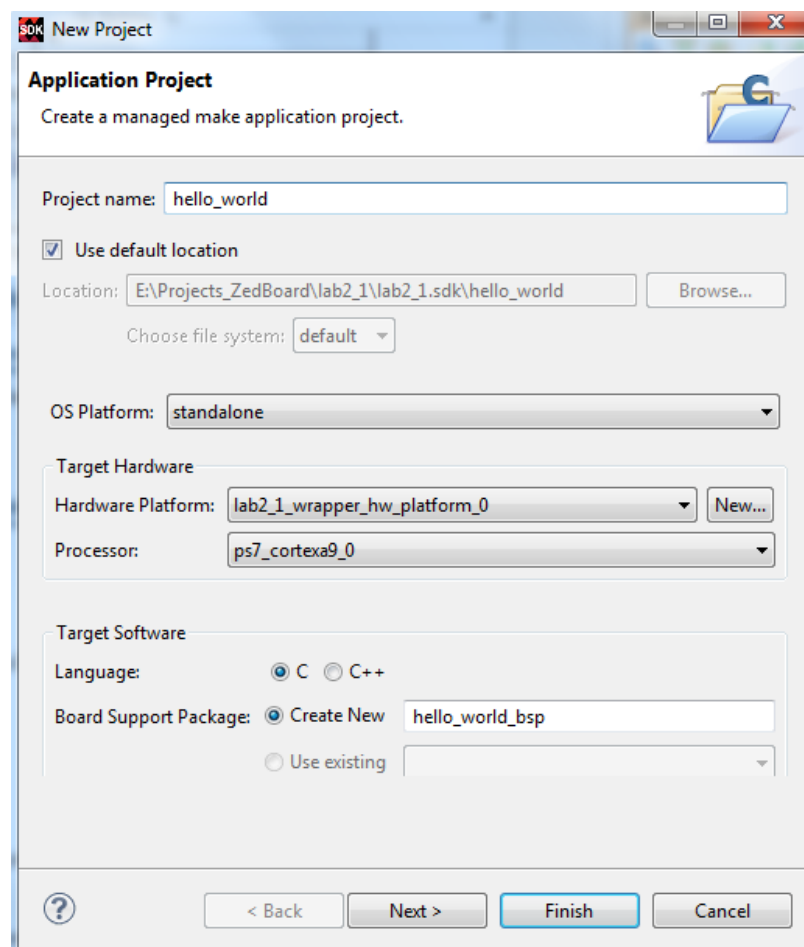
### 2-1-1. Select File -> New -> Application Project.

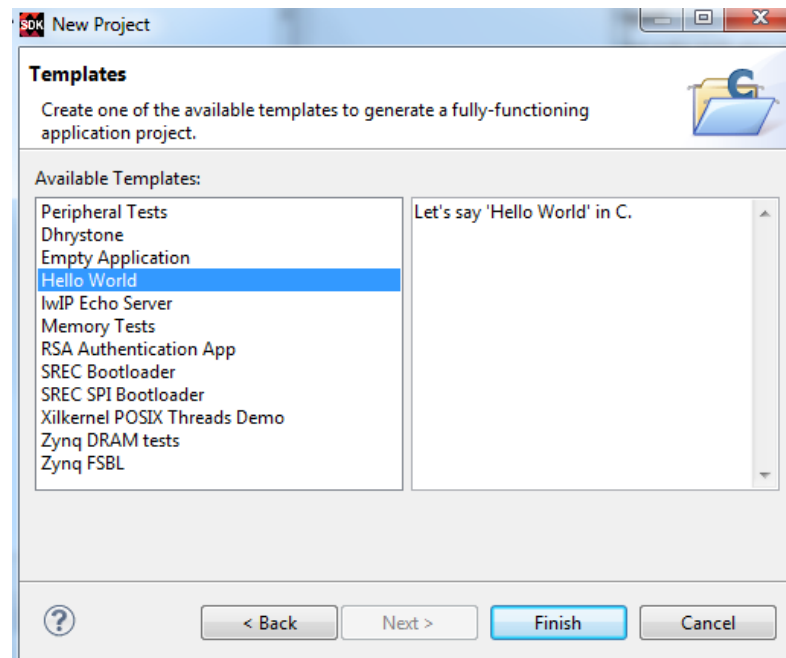


To create the new application use the following information:

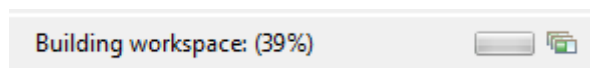
Wizard Screen	System Property	Setting
New Application Project	Project Name	hello_word
	Used default location	Check this option
	OS Platform	Standalone
	Hardware Platform	system_wrapper_hw_platform_0
	Processor	ps7_cortexa9_0
	Language	C
	Board Support Package	Create New: hello_world_bsp
Click <b>Next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

The following figures shows the results of the explained procedures.





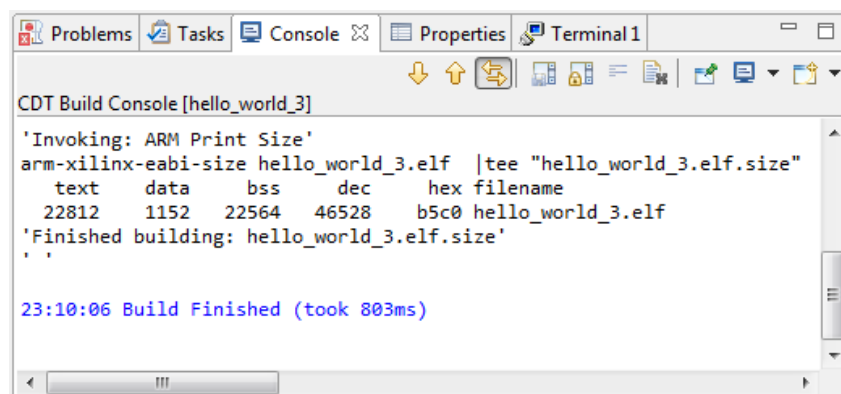
- 2-1-2.** After clicking the *Finish* button, SDK automatically begins building the application. On the bottom left part of the main window there is a small indicator about this process.



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.

- 2-1-3.** 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.



- 2-1-4.** 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. The 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.



---

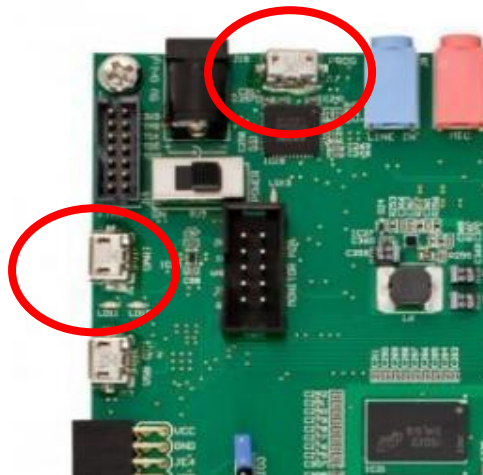
## Connecting the ZedBoard

---

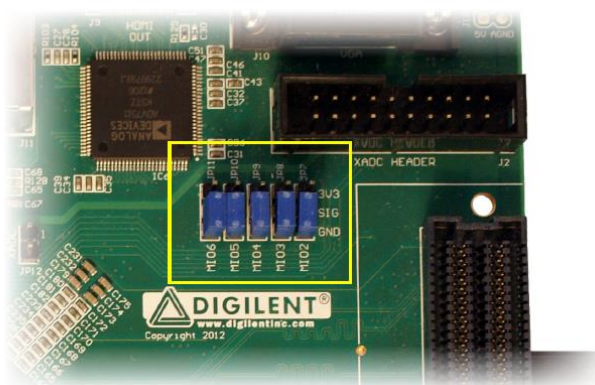
## Step 3

### 3-1. **Objective:** Steps to connect the ZedBoard to run the application.

- 3-1-1.** 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).
- 3-1-2.** 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.



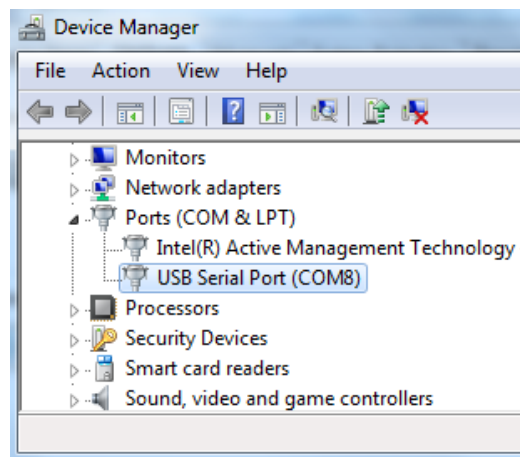
- 3-1-3. IMPORTANT 1:** Ensure that jumpers JP7 to JP11 are set as shown in the figure below for the JTAG configuration mode.



- 3-1-4. IMPORTANT 2:** be sure to have installed the Cypress device driver for the USB-UART chip on the ZedBoard.
- 3-1-5.** 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.



- 3-1-6.** Connect the 12V AC/DC converter power cable to the ZedBoard barrel jack.
- 3-1-7.** 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.
- 3-1-8.** 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.

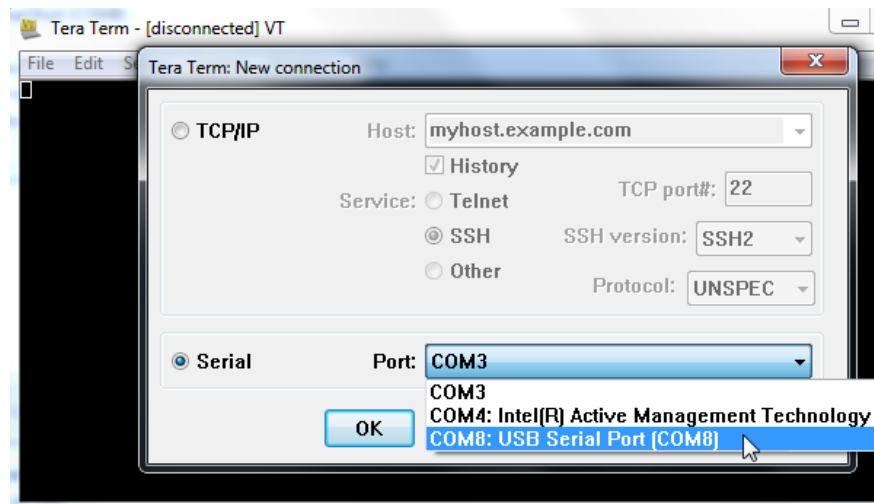


### 3-2. **Objective:** Setting up the serial COM port

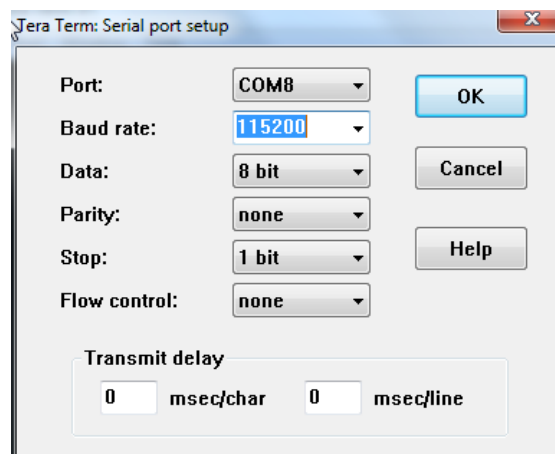
- 3-2-1.** Use a utility program such as *Tera Term* or *Putty* to setup a serial communication between the Host and the ZedBoard, by using the Host COM port.

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.



**3-2-2.** Configure the Serial port of *the Tera Term* or the software you are using, with the values detailed in the following figure.



---

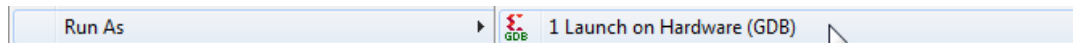
## Running the application

---

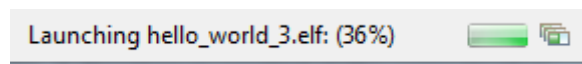
## Step 4

### 4-1. **Objective:** Steps to follow to execute the application written in 'C'

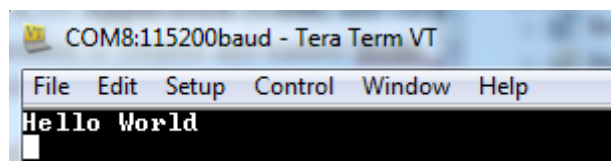
- 4-1-1. On the Project Explorer pane, select the *hello\_world* application, and then right click mouse and select *Run As -> Launch on Hardware (GDB)*



- 4-1-2. 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.



- 4-1-3. Once it is done, the program will be automatically executed. Hence, in the Tera Term window the "Hello World" message should be displayed.



## *Additional Information*

---

- 5-1. Note 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.
- 5-2. Note 2:** All the initialization routines needed for the ARM Cortex-A9 are automatically created, there is no need for the designer to create them.
- 5-3. Note 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.
- 5-4. Note 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*

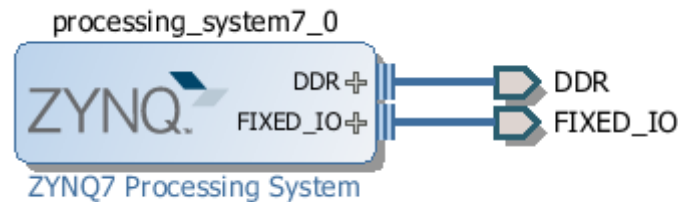
---

- 6-1.** 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.
- 6-2.** 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.
- 6-3.** Use the optimization per size option of the C/C++ Build Settings. Repeat the previous two processes and find out the respective sizes.
- 6-4.** 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?

## PART II – Memory Test

### Description

- 7-1. Follow the steps detailed in the Part I of this lab to build a block design, name the design as “lab2\_2”.



Note: be aware that you need to enable DDR memory controller from the Zynq customization window.

- 7-2. Create the wrapper.  
 7-3. Generate the block design.  
 7-4. Export the hardware design.  
 7-5. Launch the SDK.  
 7-6. In the SDK environment create a new application project with the name “lab2\_2\_memory\_test”.  
 7-7. In the templates window select the “Memory Test” option.  
 7-8. Follow the necessary steps to connect and power the board as it was explained before.  
 7-9. Execute the ‘C’ code.  
 7-10. You should obtain a result window similar to this one:

```
COM8:115200baud - Tera Term VT
File Edit Setup Control Window Help
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As
s will not be generated
Testing memory region: ps7_ddr_0
Memory Controller: ps7_ddr
Base Address: 0x00100000
Size: 0x1fff0000 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
Testing memory region: ps7_ram_1
Memory Controller: ps7_ram
Base Address: 0xffff0000
Size: 0x0000fe00 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
--Memory Test Application Complete--
```

## Challenges

---

- 8-1.** Execute the 'C' code step by step. Go into the for-loop to try to understand the different functions executed inside the loop.
- 8-2.** Insert a *print* or *xil\_printf* to print out your name and lab number in the result window.
- 8-3.** Try to find out the size of the 'C' code.

## Note

*This practical exercise is based in a Laboratory offered by the Xilinx University Program (XUP). It has been modified and updated by Cristian Sisterna.*