Electronic System Design for Communications

Lab 0: Introduction to the design environment

Last Update: September 2018

The aim of this laboratory is to introduce the tools we are going to use along this course: Vivado and SDK. This document describes step-by-step how to implement the design described in the first plenary session. The plenary presentation has been done with the 2018 version of the program, whereas this tutorial is implemented with the 2014 version. Please, keep an open mind! It is important to understand the main path to follow in a design process. The different program versions introduce minor changes in the design flow. However, we are aware that for first time users these minor differences can be a big wall difficult to overcome…so, do not hesitate to ask for help if needed!.

1. Zynq 7000 family chip and Zybo board

Along this course, we are going to implement our designs in one of the Zynq 7000 family chips, manufactured by Xilinx, specifically; we are going to use the ZYNQ XC7Z010-1CLG400C. This integrated circuit is capable to implement by itself complex and high performance systems-on-a-chip. As designers, we have several design choices: on the one hand, we can map all or part of our design in high speed hardware, thanks to the built-in FPGA (Artix 7 family from Xilinx). Additionally, this integrated circuit contains an embedded microcontroller, which is formed by a dual core Cortex A9 RISC microprocessor and other standard hardware blocks, such as Timers, Direct Memory Access (DMAs) and I/O peripherals (which allow communication with several standards, such as USB, Ethernet, I2C, among others). We can use all these blocks to perform a total or partial software implementation of our design. Data can be interchanged between the microcontroller and the FPGA through a built-in bus, called AXI. This bus follows the AXI protocol, which is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip design. Moreover, to enhance design flexibility, both the microprocessor and the FPGA can be connected to several IC pins, which can also be configured to work with several standard logic families.

Due to the inherent complexity of this chip, the design process is as well complex and has several steps. All the design steps are categorized into two groups:

- Hardware design, which may include the following steps:
  - Design of the hardware blocks implemented in the FPGA.
  - Specification of which blocks of the microcontroller are going to be used (for instance, the DMA or the Interrupt Controller can be hardware disabled to save power consumption).

    - Clock design: this implies to specify which is the clock source (can be either off chip or one of the PLL-oscillators on chip) and clock frequency. The design may contain several clock signals.
    - Pinout configuration (i.e. which IC pins is used as input/output of our design and which logic standard levels will follow).
    - Interconnection between the different hardware blocks.
    - Interconnection between the different hardware blocks implemented in the FPGA and the software functions running in the microcontroller using the AXI bus.
    - Physical placement and route of all the hardware blocks within the FPGA.
- Software design: once the hardware has been completely defined, then the software that is going to run in each microprocessor core can be designed. This software can run "stand alone" or running over an operating system, such as Linux.

The designs implemented in the Zynq IC are meant to interact with the outside world. This implies that the design will probably get data from other designs, sensors, or user interfaces and after processing, storing or communicating, to interact again with the outside world or with the user. In this course, we are going to use the Zybo board, designed by Digilent. This board will make easy for us to communicate the Zynq IC with the Personal Computer to either send or receive data to/from the design implemented in the Zynq IC, to perform Ethernet communications between designs, to control a computer screen with the VGA connector and to interact with the user with buttons, leds and switches.

2. Design Specifications

The design specifications are summarized in Fig. 1. The user will enter a 4 bits code using the Zybo board switches, when the code is ready, the user will press any of the switch buttons and the design will show the code entered by the user in the board leds (a led on means a logic '1').

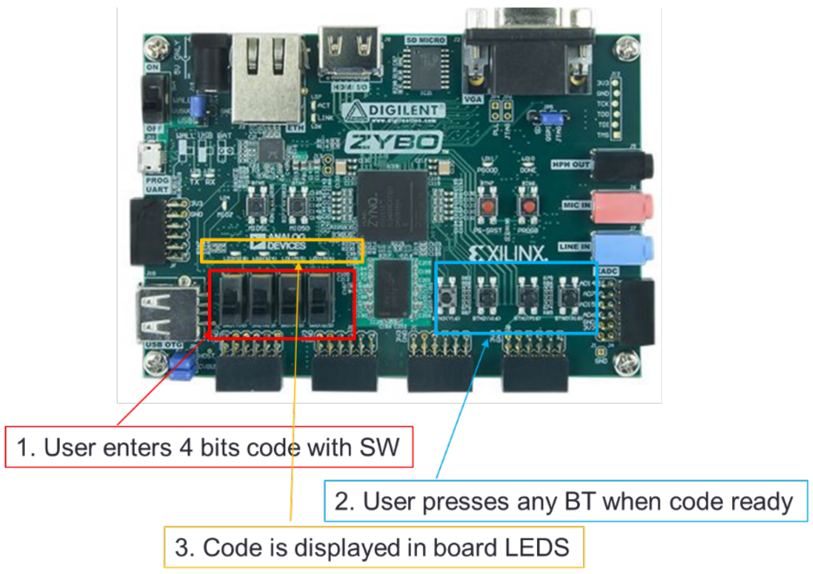

1. User enters 4 bits code with SW
2. User presses any BT when code ready
3. Code is displayed in board LEDS

Figure 1: Design specifications.

Thanks to the flexibility of the Zynq IC, we have several approaches to implement this design: only-hardware design or mixed software/hardware design. Only-software design is not possible with this evaluation board, as the leds, switches and buttons are soldered to pins that are only accessible through the FPGA within the IC. Therefore, minimum hardware design in the FPGA is required to read/write data from the three user interface elements. Fig. 2 shows the block diagram of the hardware design.

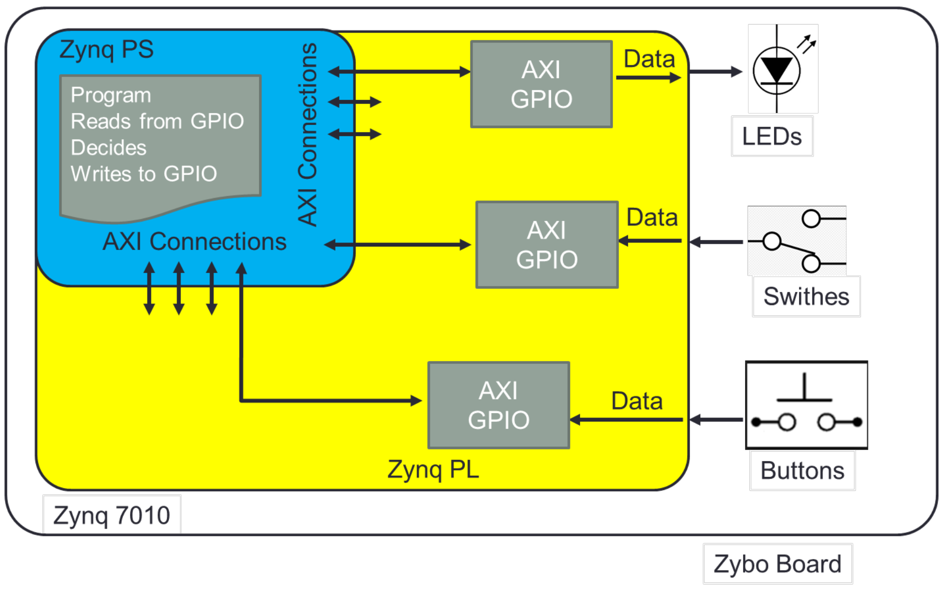


Figure 2: Design block diagram.

The AXI GPIO (General Purpose Input Output) is an IP (Intelectual Property) that is available in the library of the design tools used. On the one hand, it contains registers to store the data that is read/written from/to switches and buttons/leds. On the other hand, it has all the hardware to connect this register to the AXI bus. This allows to the microprocessor to read/write from/to the registers. Once the hardware has been designed, the software should implement the following algorithm: to perform an infinite polling on the buttons state. When a change of state is detected, then to read from the switches and to write the same value to the leds. Once this operation has been performed, the infinite polling resumes.

3. Designing the hardware

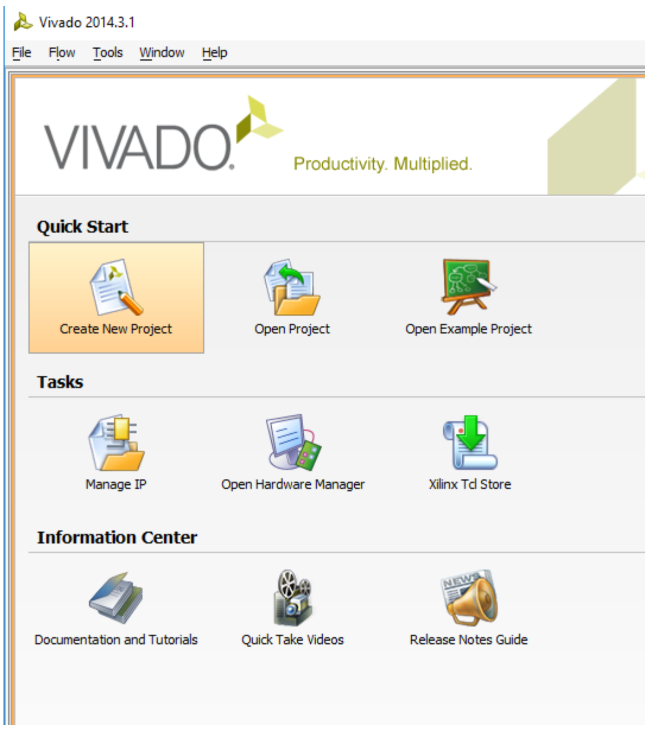
Let’s get down to work! The program that allows us to design the hardware we are going to implement in the Zynq IC is called VIVADO. In this example, we are using the 2014 version of the program.

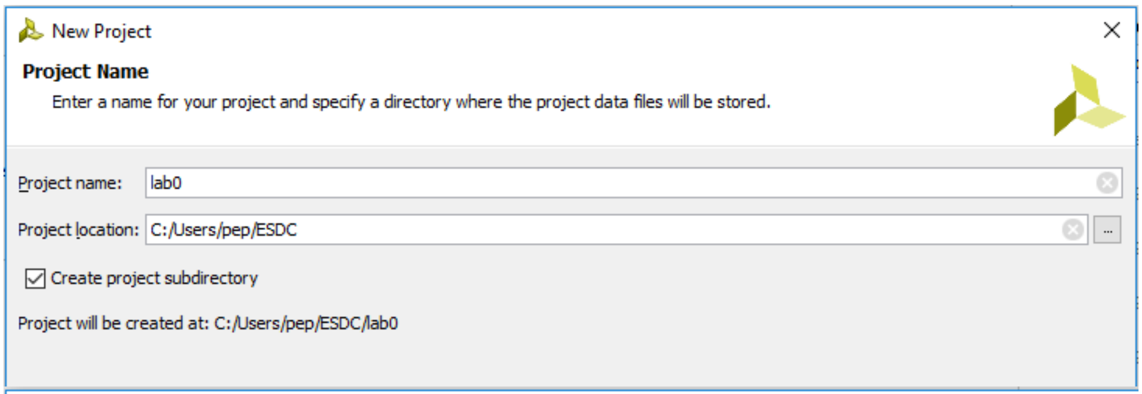
Step1:

Once the VIVADO program has been activated, a new project should be created. To create a new project, a wizard opens up. Depending on the version, the order or the screens may change, but essentially the information that we should give about this new project:

- Project name and project placement. Lab0 sounds a fantastic name… but feel free of using your imagination. Very important: do not place either spaces or fancy characters in this name (use the very old MS DOS rules to name files). The same rules apply to all the folders in the path. Regarding the folder: It is advisable that this project is placed in an independent folder in the local drive storage. Network or Cloud storage are as well OK, but strongly discouraged. From our experience, this slows down the automatic design steps, as multiple drive access are required during these steps.
- Project Type: We are going to implement a RTL Project (Register Transfer Level), as we are going to use IPs, eventually create new ones (not in this lab), and perform a synthesis of all the design. The tag: "Do not specify sources at this time" should be activated.
- Default Part: VIVADO is a generic tool for all the configurable hardware ICs manufactured by Xilinx. It is important that we tell to the program that we are using the Zybo Board by Digilent. With this action, the program would be able, later on, to automatically personalize the different IPs available in the library for this board: in this particular board, led, buttons and switches are already soldered to specific IC pins. In addition to this, the microcontroller has some specific pins soldered to an external memory, where the designed software will be stored.

Figures related to Step 1:





New Project

Project Name

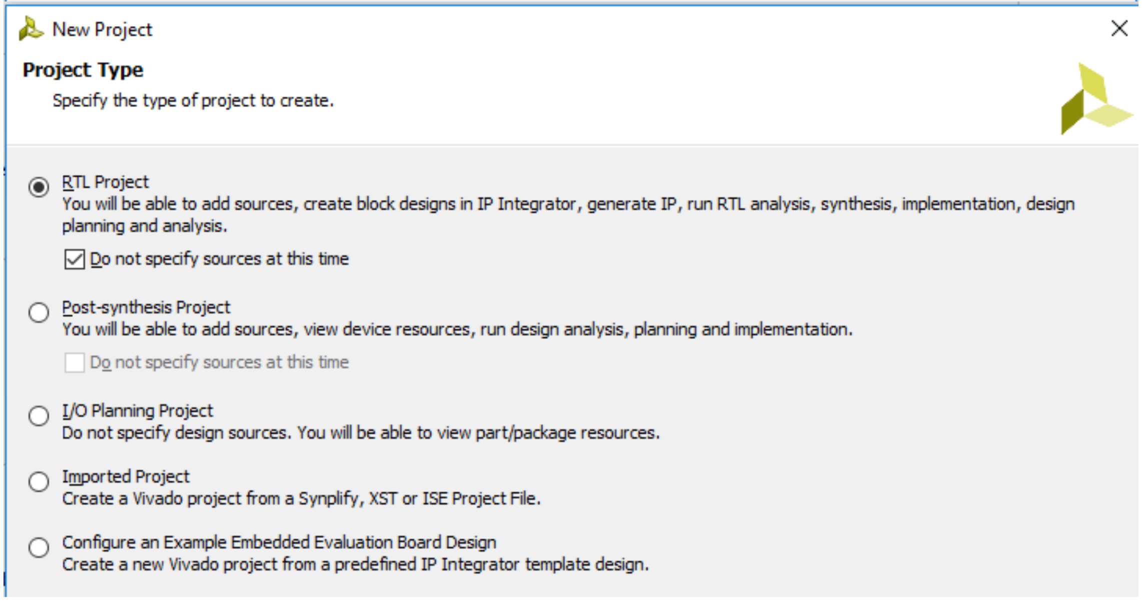
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: lab0

Project location: C:/Users/pep/ESDC

[✓ Create project subdirectory]

Project will be created at: C:/Users/pep/ESDC/lab0



New Project

Project Type

Specify the type of project to create.

[✓ RTL Project]
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

[✓ Do not specify sources at this time]

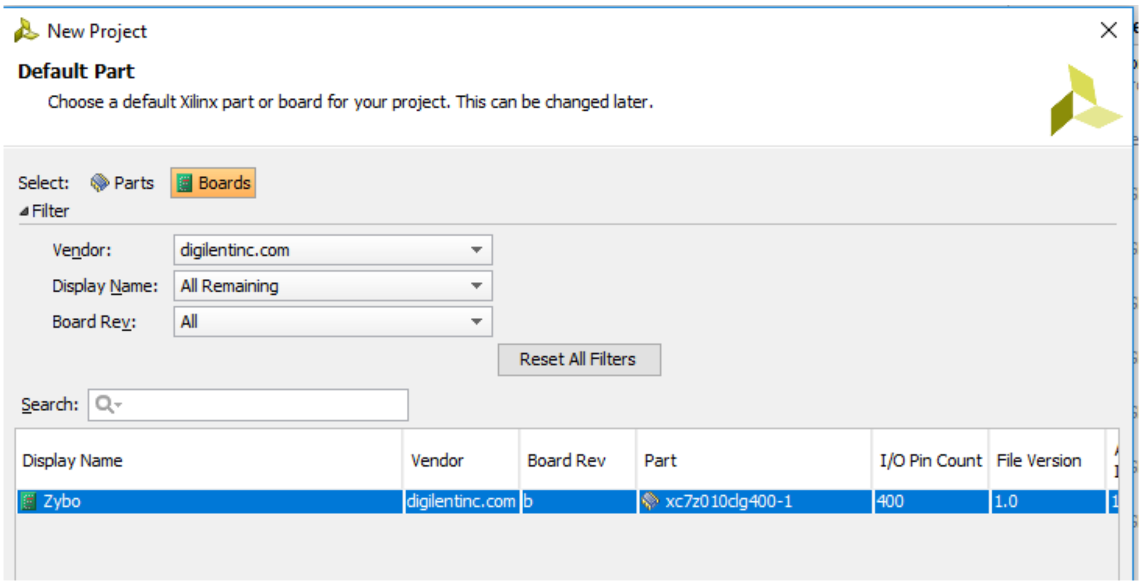
Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.

Do not specify sources at this time

I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

Configure an Example Embedded Evaluation Board Design
Create a new Vivado project from a predefined IP Integrator template design.



New Project

Default Part

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

Select: Parts [✓ Boards]

Filter

Vendor: digilentinc.com

Display Name: All Remaining

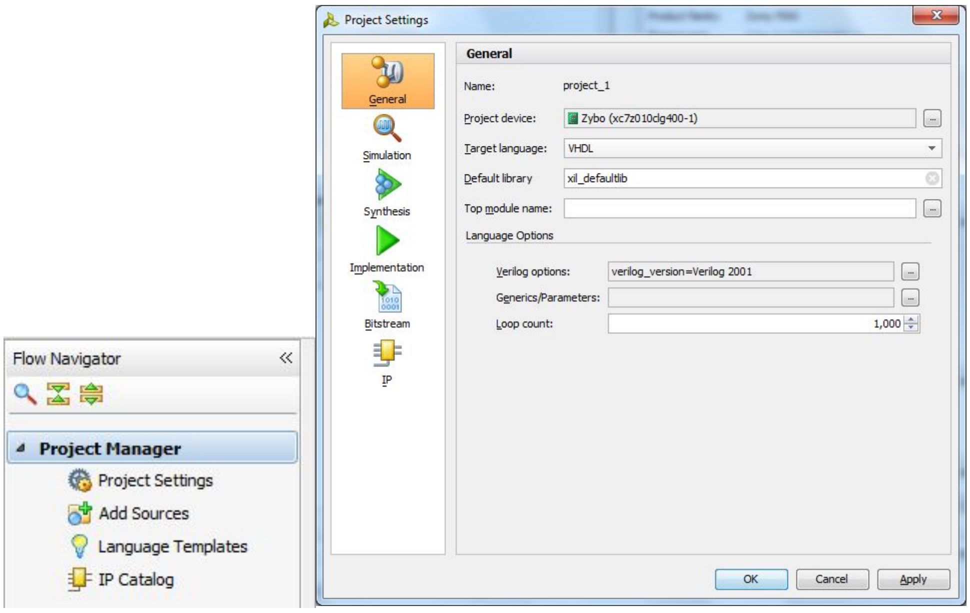
Board Rev: All

Reset All Filters

Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version
Zybo	digilentinc.com	b	xc7z010clg400-1	400	1.0

When the wizard disappears, verify in Project Settings that the target language is VHDL: the window Flow Navigator, click on Project Settings (Project Manager). If not, change from Verilog to VHDL.



General

Name: project_1

Project device: Zybo (xc7z010clg400-1)

Target language: VHDL

Default library xil_defaultlib

Top module name:

Language Options

Verilog options: verilog_version=Verilog 2001

Generics/Parameters:

Loop count: 1,000

Step 2:

At this point, the program knows that we want to perform a design on the Xilinx IC present in the Zybo board. We can now start the hardware design. To do this, we need to create a Block Design, which is identified with a name (e.g.: hardware_lab0). In this Block Design, we will place and interconnect the hardware blocks that we are going to use. In this laboratory, all the hardware blocks are already available in the library of program as IPs. We will need to place three AXI GPIO blocks and the Zynq Microcontroller. Once these blocks have been placed, we need to personalize for our particular purpose, i.e., the microprocessor has to work in the Zybo board (this will assign some microprocessor I/O to specific pins and to size internal resources of the microprocessor). Moreover, we have to indicate that the three GPIOs will be used to read/write data from switches, buttons and leds. This will perform two actions on these hardware blocks: on the one hand, will size its output bus to 4 bits (as in this board, we have 4 leds, switches and buttons). On the other hand, will connect the output bus of the GPIOs to specific pins, i.e., the ones that are soldered to the leds, switches and buttons in the board.

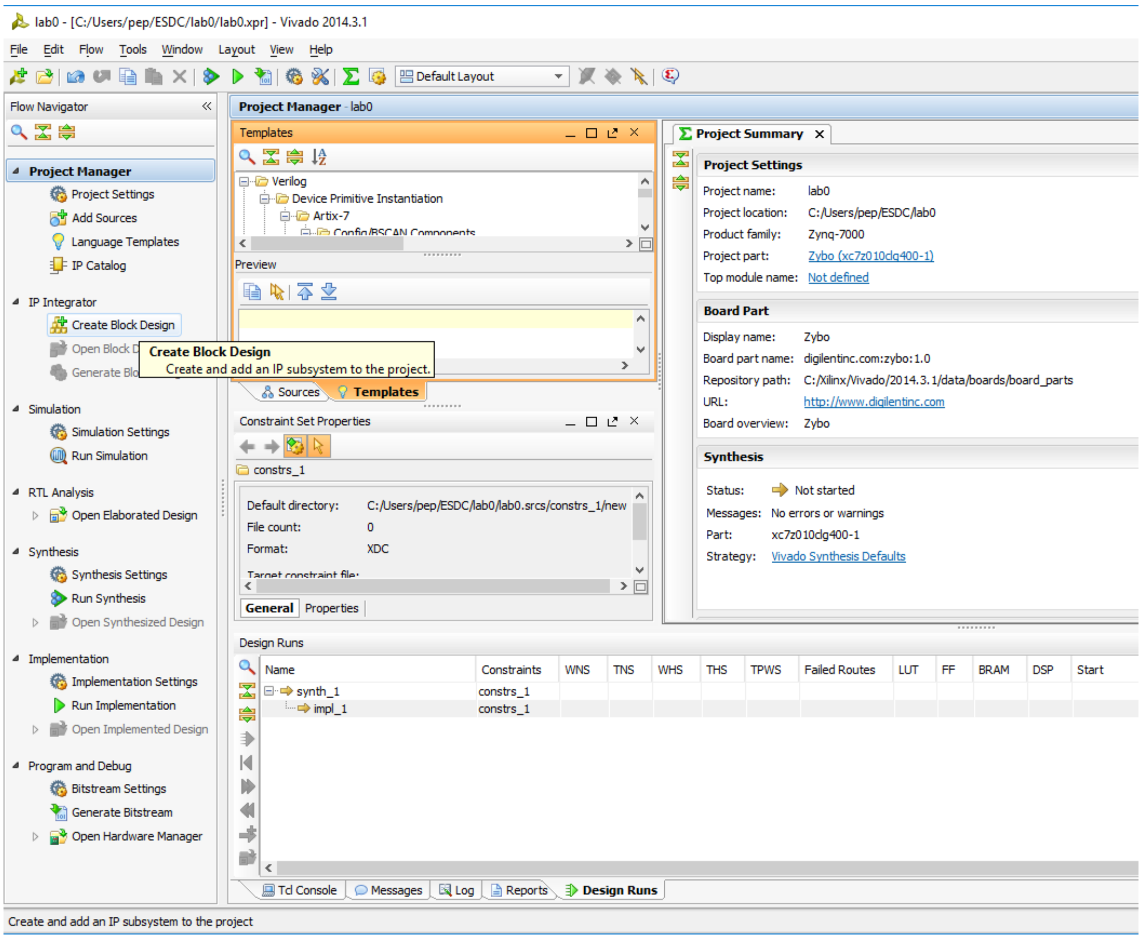
It is as well advisable to change the default name given to the AXI GPIOs blocks. Later on, during the software design, we will use specific functions to write and read data from these GPIOs. If the names of these blocks are related to the function they perform, to recognize their names in the program will be easier for us and the software program will be easier to understand and to maintain.

The following figures show how to introduce the microprocessor, to personalize it to the hardware requirements of the Zybo board, to place the AXI GPIOs and to personalize them.

Other hardware blocks will be as well needed (such as the AXI Bus Arbiter), but we do not need to take care of this, as the program will automatically interconnect the AXI GPIOs and the Microcontroller. This is possible as the IPs have been created with specific Attributes that allow the Vivado program to perform some design steps in automatic way.

Figures related to step 2:

Click on Create Block Design and assign a name.


lab0 - [C:/Users/pep/ESDC/lab0/lab0.xpr] - Vivado 2014.3.1

Create Block Design
Create and add an IP subsystem to the project.

Default directory: C:/Users/pep/ESDC/lab0/lab0.srcs/constrs_1/new
File count: 0
Format: XDC
Target constraint file:

Project Settings

Project name:	lab0
Project location:	C:/Users/pep/ESDC/lab0
Product family:	Zynq-7000
Project part:	Zybo (xc7z010clg400-1)
Top module name:	Not defined

Board Part

Display name:	Zybo
Board part name:	digilentinc.com:zybo:1.0
Repository path:	C:/Xilinx/Vivado/2014.3.1/data/boards/board_parts
URL:	http://www.digilentinc.com
Board overview:	Zybo

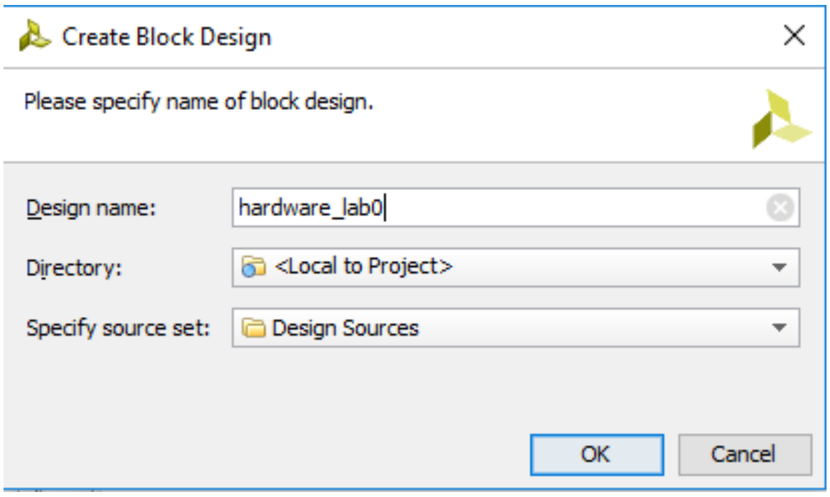
Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z010clg400-1
Strategy:	Vivado Synthesis Defaults

Design Runs

Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP	Start
synth_1	constrs_1											
impl_1	constrs_1											

Create and add an IP subsystem to the project


Create Block Design

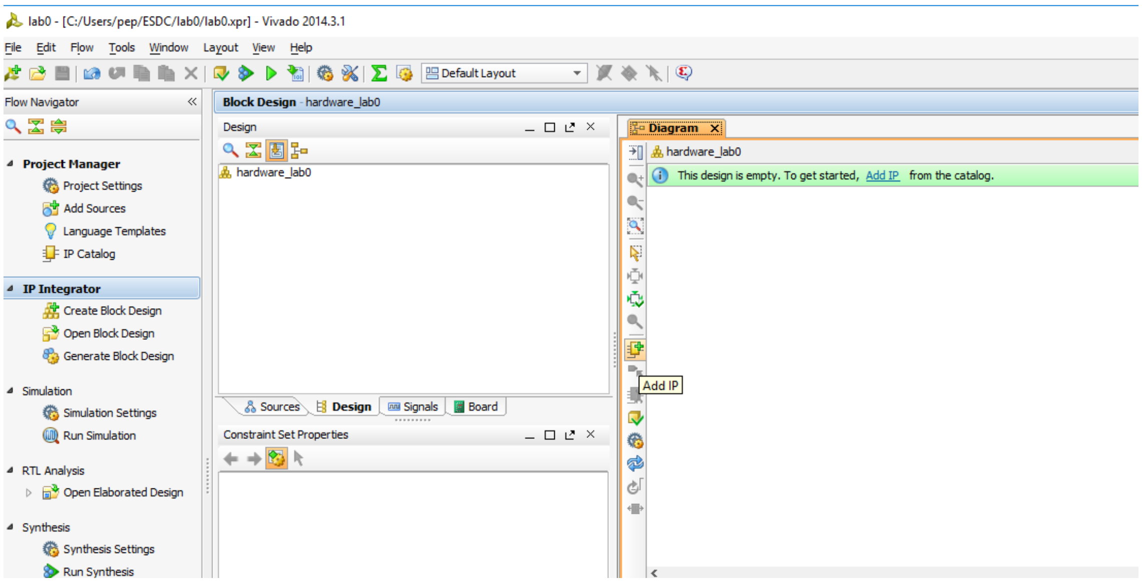
Please specify name of block design.

Design name: hardware_lab0

Directory: <Local to Project>

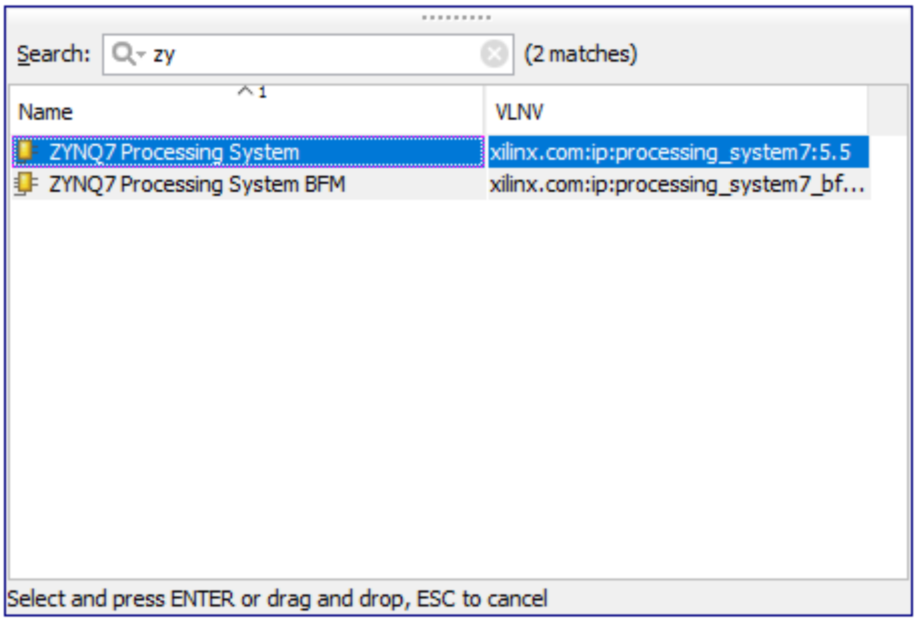
Specify source set: Design Sources

Click on add IP on the Diagram Block window tab. Several ways to perform this action. E.g.: see message on the top of the tap. See highlighted icon in the figure.


lab0 - [C:/Users/pep/ESDC/lab0/lab0.xpr] - Vivado 2014.3.1

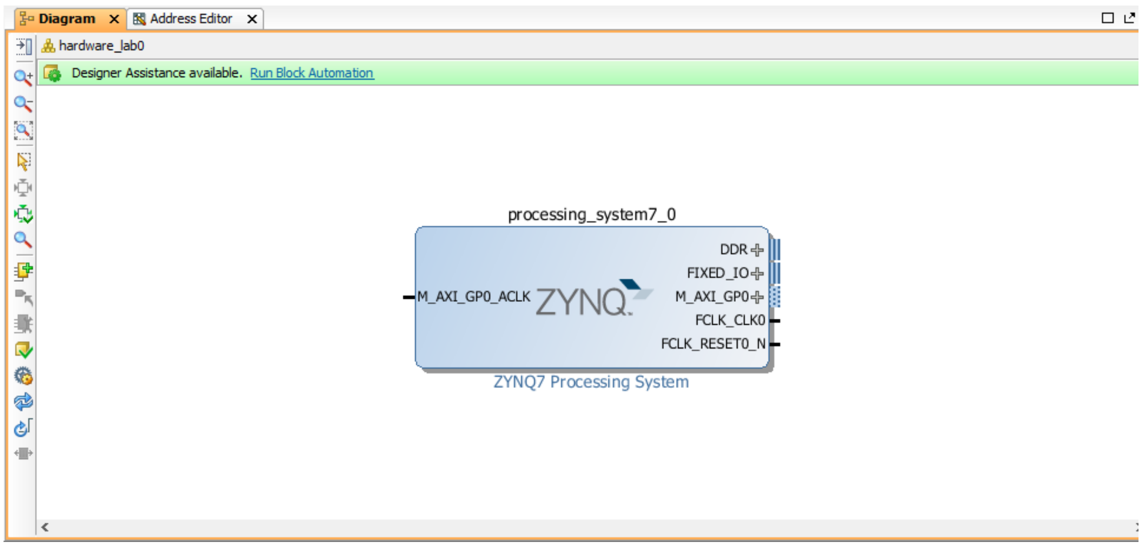
This design is empty. To get started, Add IP from the catalog.

Select the IPs to place: Zynq Processing System


Search: zy (2 matches)

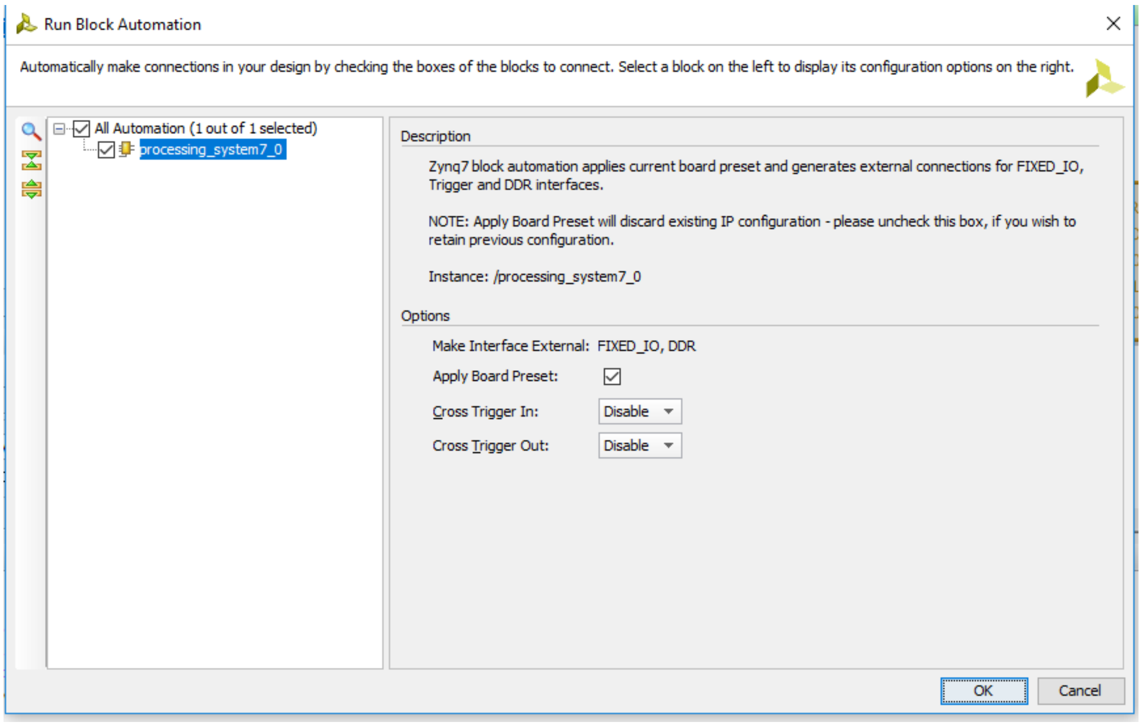
Name	VLNV
ZYNQ7 Processing System	xilinx.com:ip:processing_system7:5.5
ZYNQ7 Processing System BFM	xilinx.com:ip:processing_system7_bf…

Select and press ENTER or drag and drop, ESC to cancel



Designer Assistance available. Run Block Automation

We should Run Block Automation to customize this block for the requirements of the Zybo board:



Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (1 out of 1 selected)
processing_system7_0

Description

Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Options

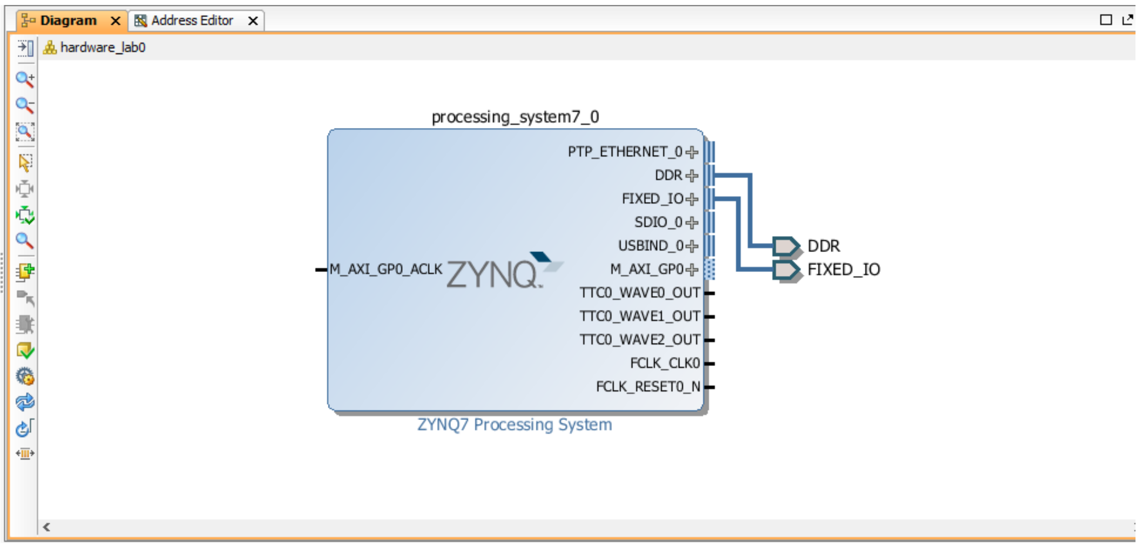
Make Interface External: FIXED_IO, DDR

Apply Board Preset: ✓

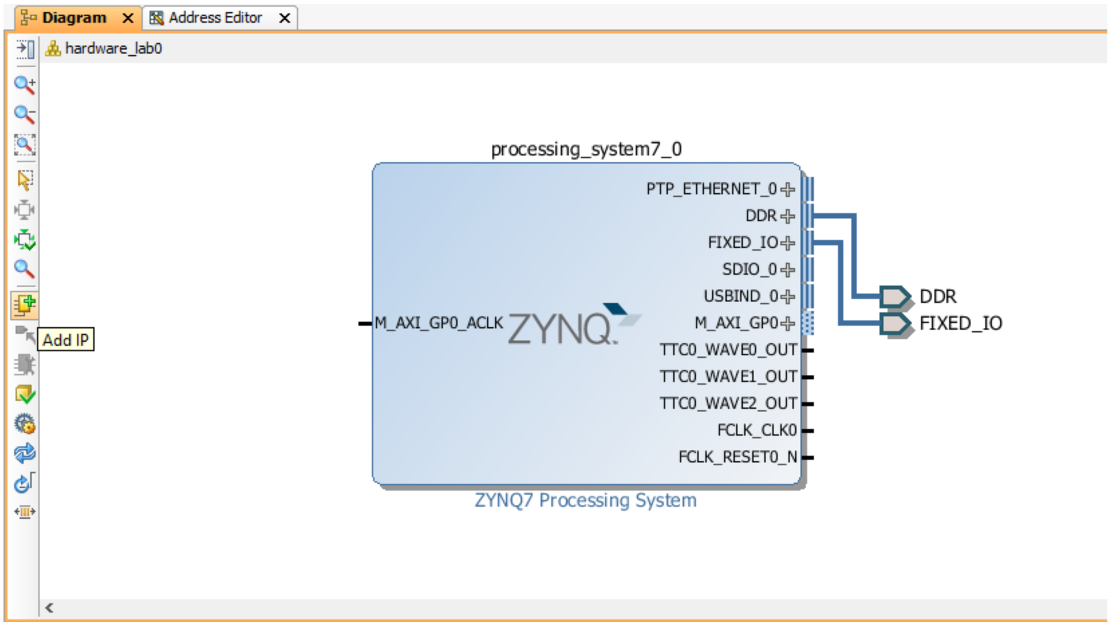
Cross Trigger In: Disable

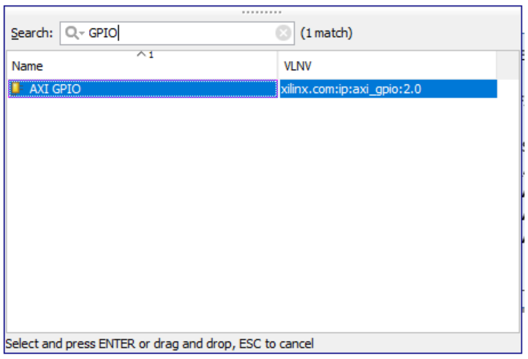
Cross Trigger Out: Disable

Block personalized to be used in our board:



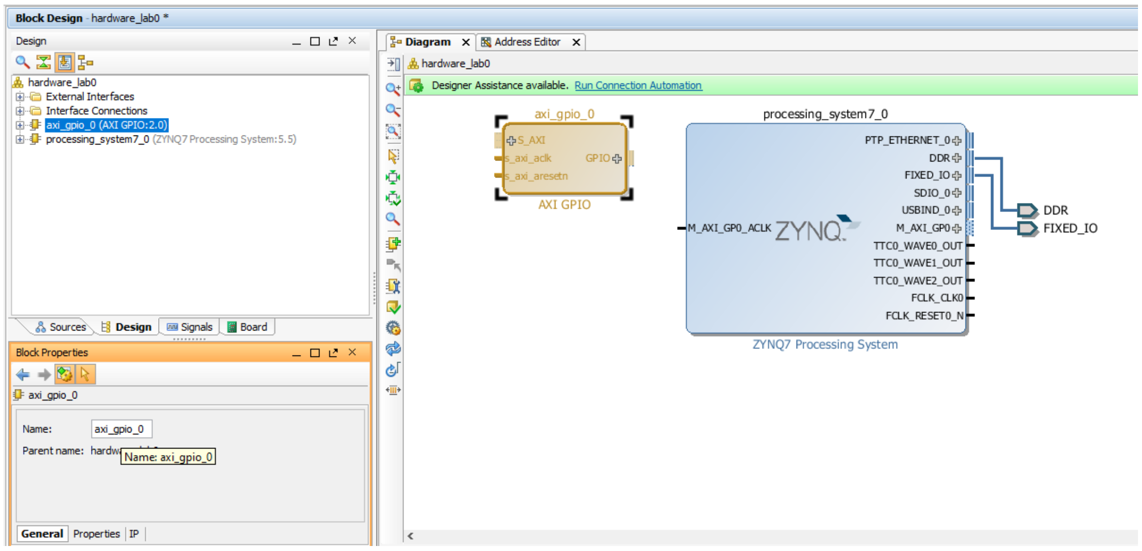
Place the IP AXI GPIO



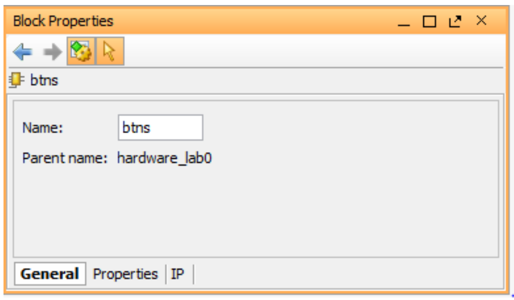


Name	VLNV
AXI GPIO	xilinx.com:ip:axi_gpio:2.0

Single click on the new block. Go to block properties to change the block name. Use, for instance, btns (referring to buttons).



Name: axi_gpio_0
Parent name: hardw
Name: axi_gpio_0



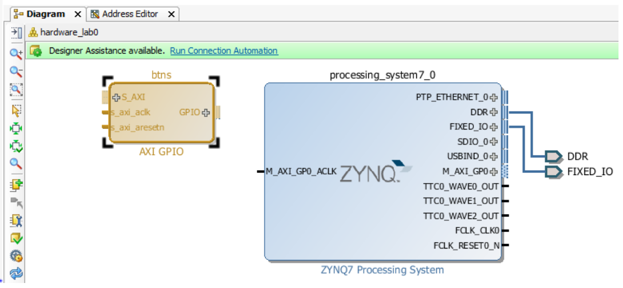
Block Properties

btns

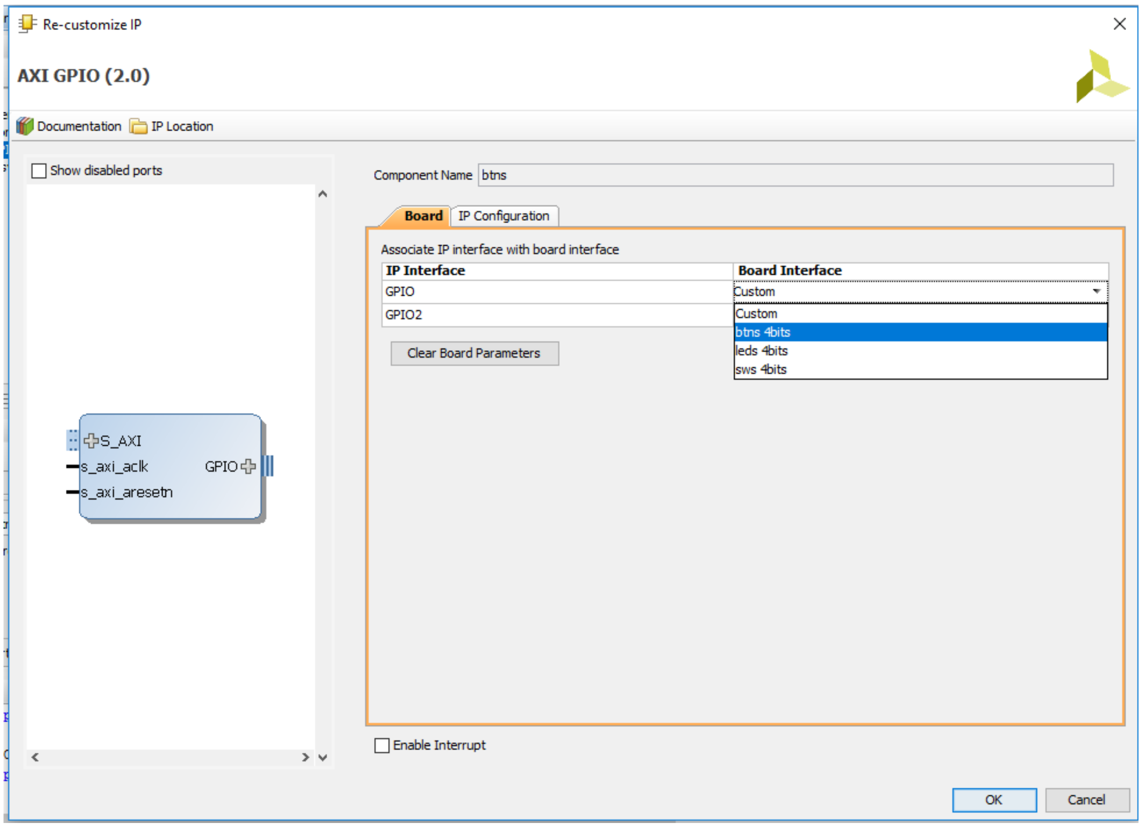
Name: btns

Parent name: hardware_lab0

General | Properties | IP



Double click this new block, in order to customize it to read the Zybo board buttons state. You should modify the Board Interface of GPIO, not GPIO2.



Re-customize IP

AXI GPIO (2.0)

Documentation IP Location

Show disabled ports

Component Name: btns

Board | IP Configuration

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	Custom
GPIO2	

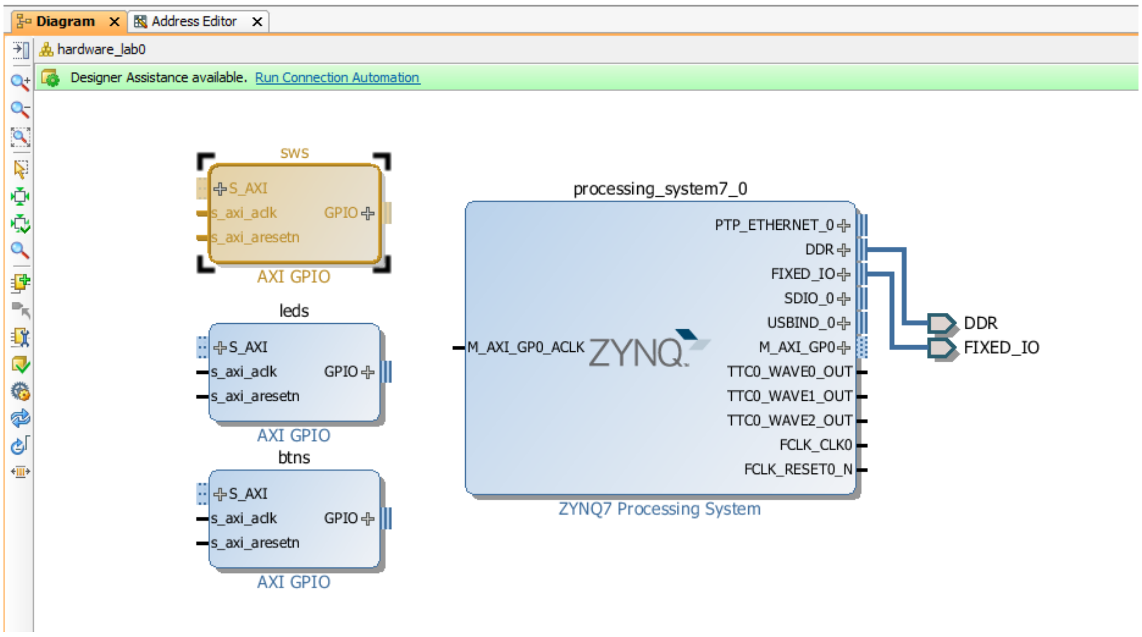
Custom
btns 4bits
leds 4bits
sws 4bits

Clear Board Parameters

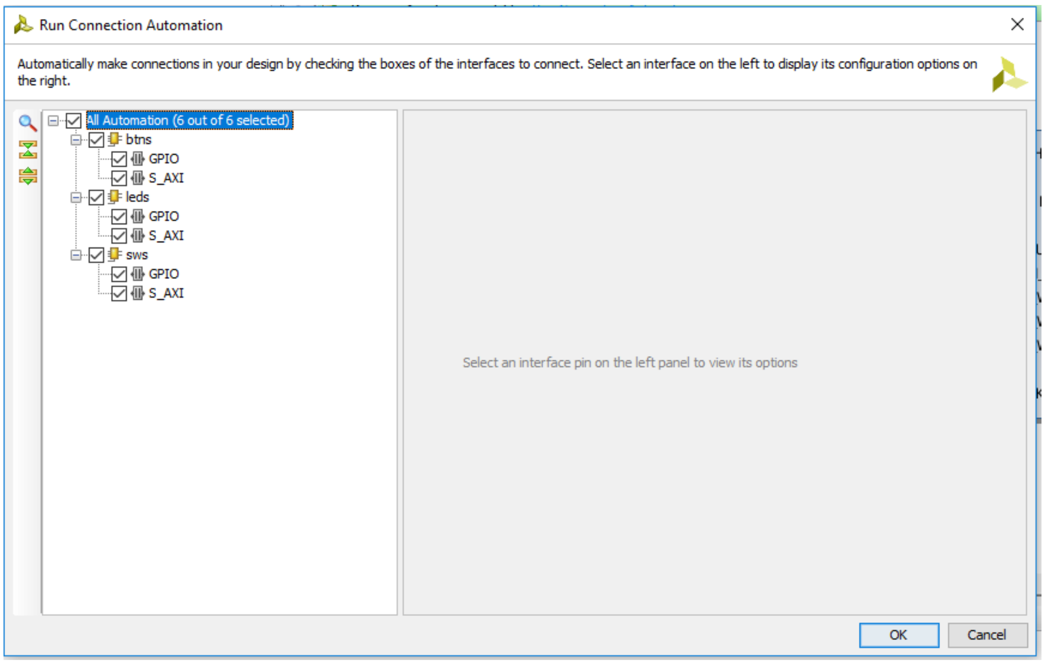
Enable Interrupt

OK Cancel

Repeat the operation twice (to insert GPIO and to customize it): but now with the names leds, sws, and configure these new GPIOs to be connected with the leds and switches.



The connection of the GPIO with the suitable IC pins, the connection of the Processing System with the GPIOs through the AXI bus and the connection of the Processing System with the memory board can be done either manually or automatically. If we press Run Connection Automation, the program tells us which of the blocks can be connected in an automatic way. We should select all them (which are, as a matter of a fact, all the blocks placed so far in our design).

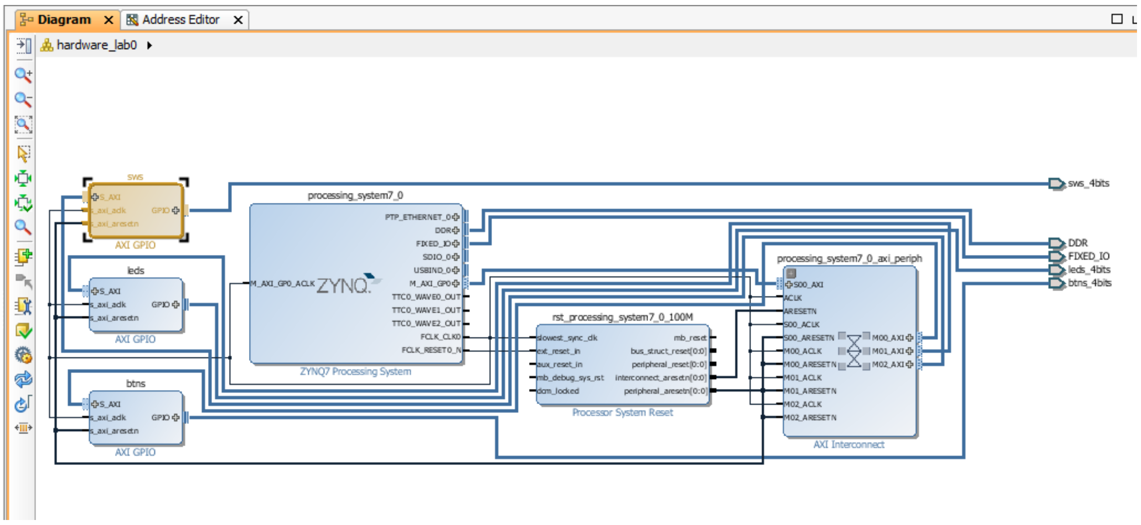

Run Connection Automation

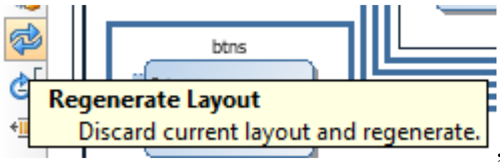
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- ✓ All Automation (6 out of 6 selected)
  - ✓ btns
    - ✓ GPIO
    - ✓ S_AXI
  - ✓ leds
    - ✓ GPIO
    - ✓ S_AXI
  - ✓ sws
    - ✓ GPIO
    - ✓ S_AXI

Select an interface pin on the left panel to view its options

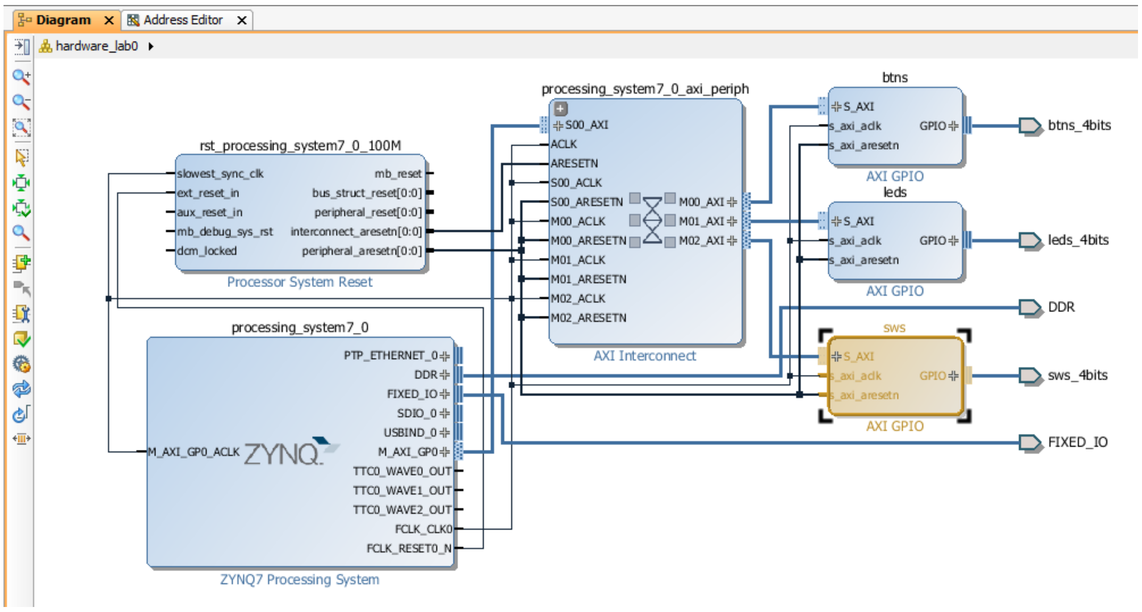
The result is:



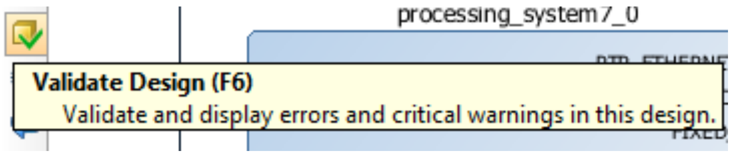


Regenerate Layout
Discard current layout and regenerate.

If we press the button Regenerate Layout, the result is a new block drawing (same design, but different layout): at the left there are the bocks within the microprocessor that are used in our design. At the right, the blocks that should be placed in the IC FPGA. In between: the AXI bus arbiter.

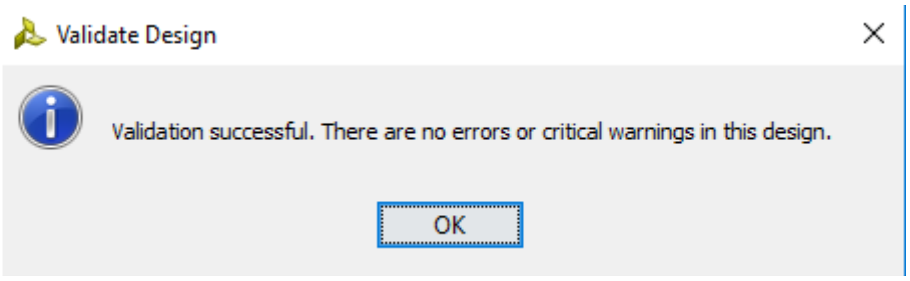


Finally, by pressing F6 or clicking on the symbol:



Validate Design (F6)
Validate and display errors and critical warnings in this design.

A validation of our design is done. In this design, no errors should appear!


Validate Design

Validation successful. There are no errors or critical warnings in this design.

Step 3:

Congratulations! The hardware design is already done! Now we have to perform the complete synthesis process, which will end with the configuration file that should be send to the Zynq chip in order to implement this design. In this design all the steps that we should do are automatic.

First, we need a VHDL description of this design (so called VHDL wrapper). VIVADO can generate a structural VHDL description of this design in an automatic way. Once the VHDL description is available, then the synthesis and the implementation processes should be activated. What these two processes do?

Synthesis

The synthesizer converts HDL (VHDL/Verilog) code into a gate-level netlist (represented in the terms of the UNISIM component library, a Xilinx library containing basic primitives). After a successful synthesis one can run "View RTL Schematic" task (RTL stands for register transfer level) to view a gate-level schematic produced by a synthesizer.

Implementation

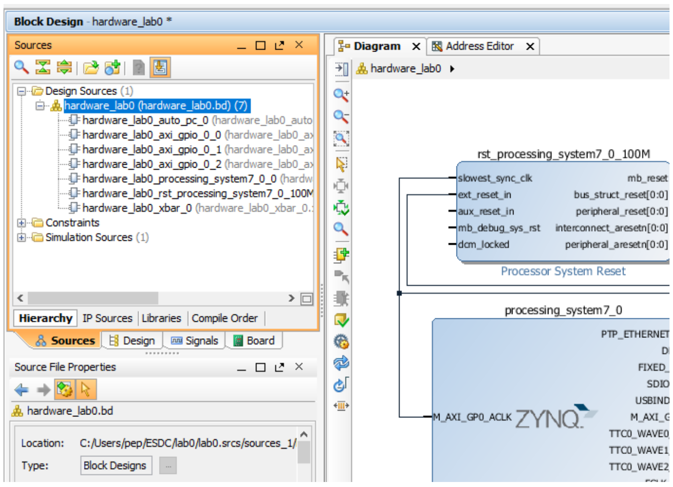
Implementation stage is intended to translate netlist into the placed and routed Zynq design.

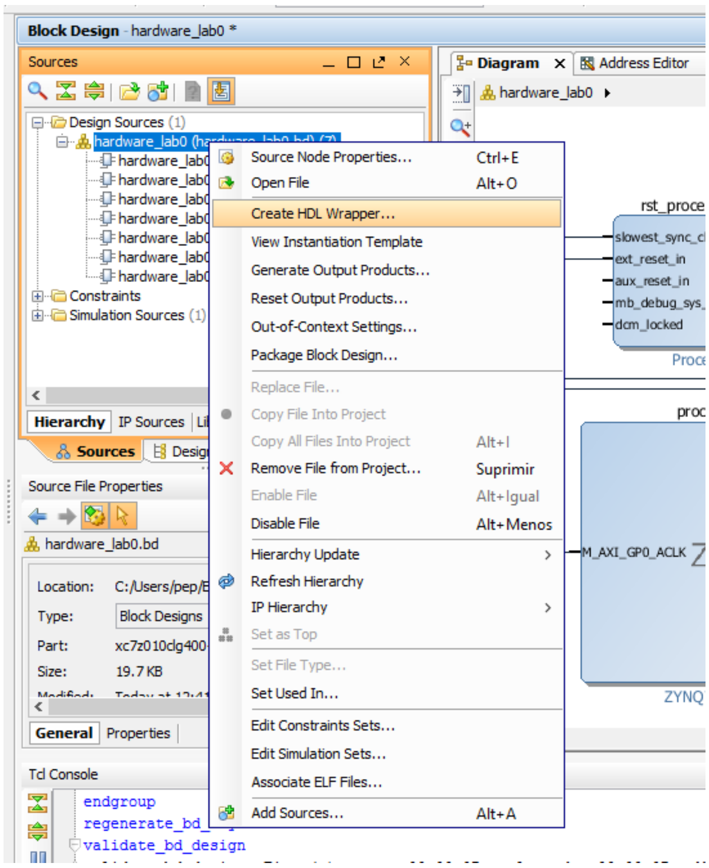
Finally, we should activate the process: Generate bitstream. The final result of this process is the configuration file that should be sent to the Zynq chip in order to be configured to implement our design.

Depending on the computer that you are using, all these automatic processes can take some minutes.

Figures related to Step 3:

Right clicking, the sources tab, the block diagram just designed:





We let Vivado do all the automatic process and auto-update, and generate the VHDL file. This new file appears in the Sources. If you double click its name, you can see the VHDL structural description of your hardware design. It's a very long file!!! So here you have a screen shot of its beginning:

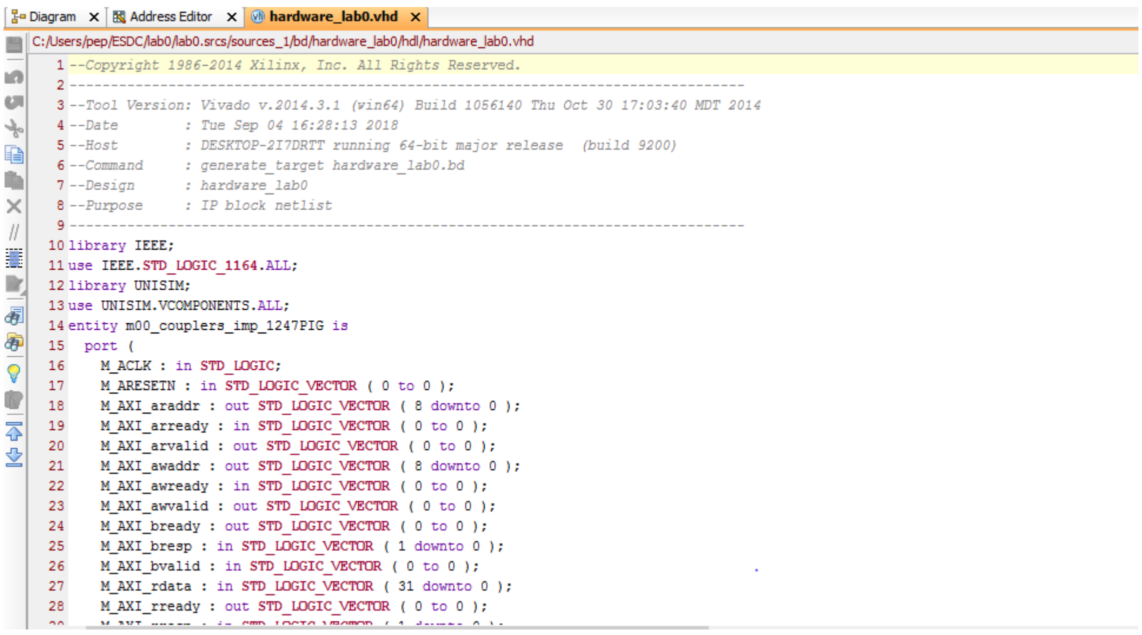


Diagram | Address Editor | hardware_lab0.vhd

C:/Users/pep/ESDC/lab0/lab0.srcs/sources_1/bd/hardware_lab0/hdl/hardware_lab0.vhd

```vhdl
--Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2014.3.1 (win64) Build 1056140 Thu Oct 30 17:03:40 MDT 2014
--Date        : Tue Sep 04 16:28:13 2018
--Host        : DESKTOP-2I7DRTT running 64-bit major release  (build 9200)
--Command     : generate_target hardware_lab0.bd
--Design      : hardware_lab0
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity m00_couplers_imp_1247PIG is
  port (
    M_ACLK : in STD_LOGIC;
    M_ARESETN : in STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_araddr : out STD_LOGIC_VECTOR ( 8 downto 0 );
    M_AXI_arready : in STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_arvalid : out STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_awaddr : out STD_LOGIC_VECTOR ( 8 downto 0 );
    M_AXI_awready : in STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_awvalid : out STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_bready : out STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_bresp : in STD_LOGIC_VECTOR ( 1 downto 0 );
    M_AXI_bvalid : in STD_LOGIC_VECTOR ( 0 to 0 );
    M_AXI_rdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    M_AXI_rready : out STD_LOGIC_VECTOR ( 0 to 0 );
```

And its end:

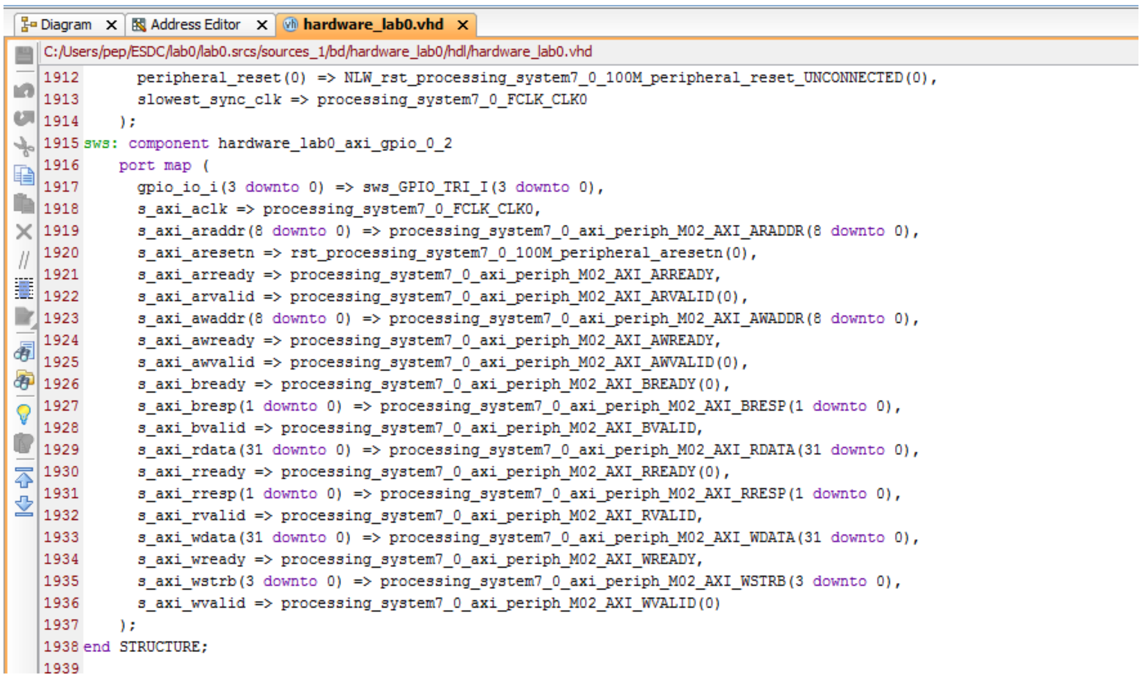


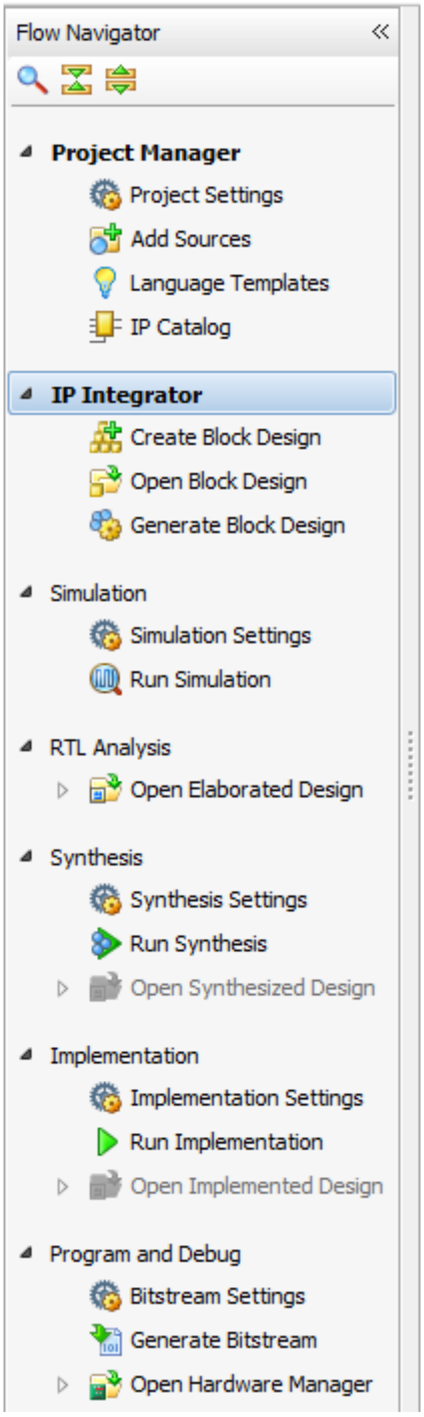
Diagram | Address Editor | hardware_lab0.vhd

C:/Users/pep/ESDC/lab0/lab0.srcs/sources_1/bd/hardware_lab0/hdl/hardware_lab0.vhd

```vhdl
      peripheral_reset(0) => NLW_rst_processing_system7_0_100M_peripheral_reset_UNCONNECTED(0),
      slowest_sync_clk => processing_system7_0_FCLK_CLK0
    );
sws: component hardware_lab0_axi_gpio_0_2
    port map (
      gpio_io_i(3 downto 0) => sws_GPIO_TRI_I(3 downto 0),
      s_axi_aclk => processing_system7_0_FCLK_CLK0,
      s_axi_araddr(8 downto 0) => processing_system7_0_axi_periph_M02_AXI_ARADDR(8 downto 0),
      s_axi_aresetn => rst_processing_system7_0_100M_peripheral_aresetn(0),
      s_axi_arready => processing_system7_0_axi_periph_M02_AXI_ARREADY,
      s_axi_arvalid => processing_system7_0_axi_periph_M02_AXI_ARVALID(0),
      s_axi_awaddr(8 downto 0) => processing_system7_0_axi_periph_M02_AXI_AWADDR(8 downto 0),
      s_axi_awready => processing_system7_0_axi_periph_M02_AXI_AWREADY,
      s_axi_awvalid => processing_system7_0_axi_periph_M02_AXI_AWVALID(0),
      s_axi_bready => processing_system7_0_axi_periph_M02_AXI_BREADY(0),
      s_axi_bresp(1 downto 0) => processing_system7_0_axi_periph_M02_AXI_BRESP(1 downto 0),
      s_axi_bvalid => processing_system7_0_axi_periph_M02_AXI_BVALID,
      s_axi_rdata(31 downto 0) => processing_system7_0_axi_periph_M02_AXI_RDATA(31 downto 0),
      s_axi_rready => processing_system7_0_axi_periph_M02_AXI_RREADY(0),
      s_axi_rresp(1 downto 0) => processing_system7_0_axi_periph_M02_AXI_RRESP(1 downto 0),
      s_axi_rvalid => processing_system7_0_axi_periph_M02_AXI_RVALID,
      s_axi_wdata(31 downto 0) => processing_system7_0_axi_periph_M02_AXI_WDATA(31 downto 0),
      s_axi_wready => processing_system7_0_axi_periph_M02_AXI_WREADY,
      s_axi_wstrb(3 downto 0) => processing_system7_0_axi_periph_M02_AXI_WSTRB(3 downto 0),
      s_axi_wvalid => processing_system7_0_axi_periph_M02_AXI_WVALID(0)
    );
end STRUCTURE;
```

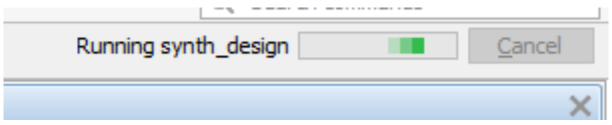
As you can check: almost 2000 lines of VHDL code (for this very simple design!).

At the left of your VIVADO window, you have now the programs that you have to launch: synthesis, implementation and generate bitstream.

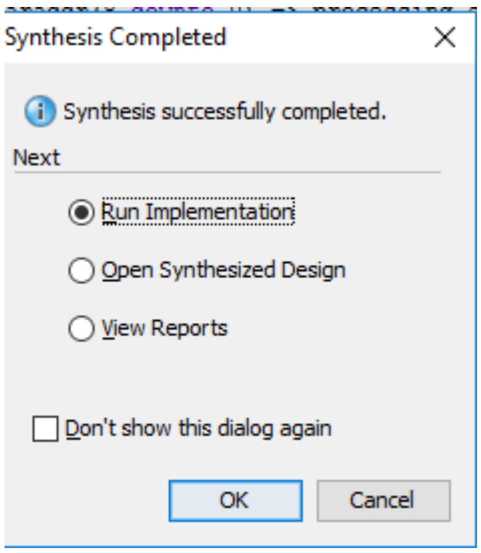


First, click on Run Synthesis.

This goes on as long as you can see this message at the upper right corner of your Vivado windows:



After some minutes, this windows prompts up:


Synthesis Completed

Synthesis successfully completed.

Next

(●) Run Implementation

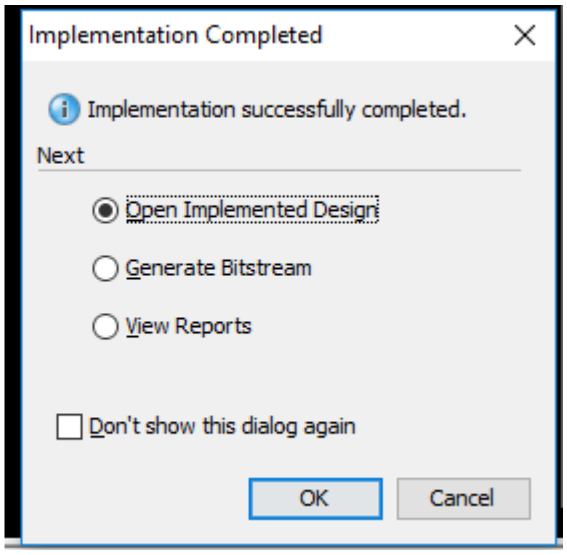
( ) Open Synthesized Design

( ) View Reports

[ ] Don't show this dialog again

OK Cancel

We can now Run Implementation, or, if you are curious, Open the Synthesized Design. When the curiosity is satisfied… you should Run Implementation. This takes a while too… Look at the top right part of the window to know if the Implementation is still running. When it finishes:


Implementation Completed

Implementation successfully completed.

Next

(●) Open Implemented Design

( ) Generate Bitstream

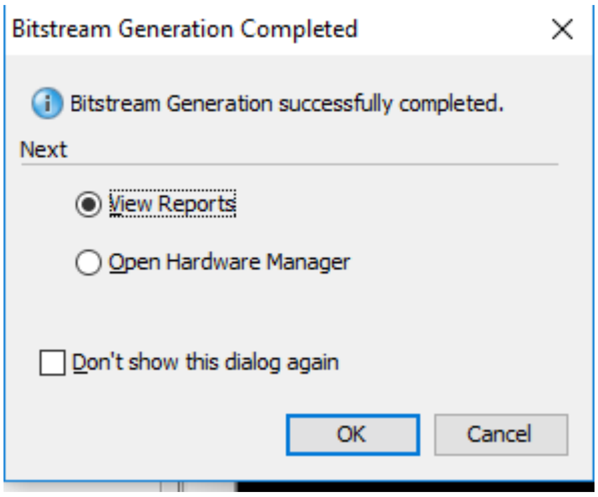
( ) View Reports

[ ] Don't show this dialog again

OK Cancel

By opening the implemented design, you can obtain the following information: the estimated power dissipation (and self heating) for this particular design realization. The estimated static and dynamic power consumption. You can also analyze device resources utilization and perform timing analysis.

You should launch now Generate Bitstream. When finished:


Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

(●) View Reports

( ) Open Hardware Manager

☐ Don't show this dialog again

You can view the reports. We do not open the Hardware Manager. This tool is needed to configure the FPGA when the design does not contain the microprocessor, which is not our current case. In this laboratory we will configure the FPGA after finishing the microprocessor software design, using the SDK (software development kit) tool.

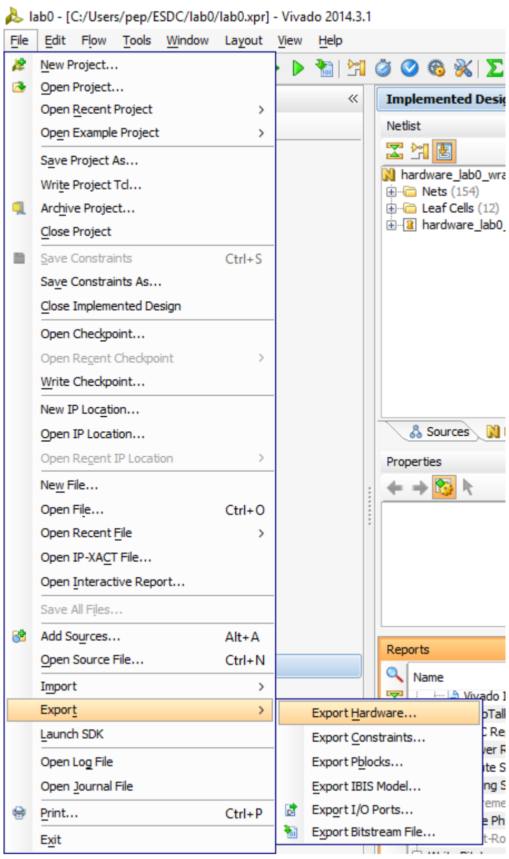
Step 4:

All the hardware design flow is done! Now we have to start the design of the software. We need to launch the tool that will be used to design the software. However, before this, we need to generate the files that inform to this tool which hardware elements we are using in our Zynq IC (e.g.: three AXI GPIO to communicate with the board leds, switches and buttons). By doing this, the system will create a set of C functions to interact, at a reasonable high level, with all these blocks. All these functions will be included in a library within what is called the Board Support Package (BSP).

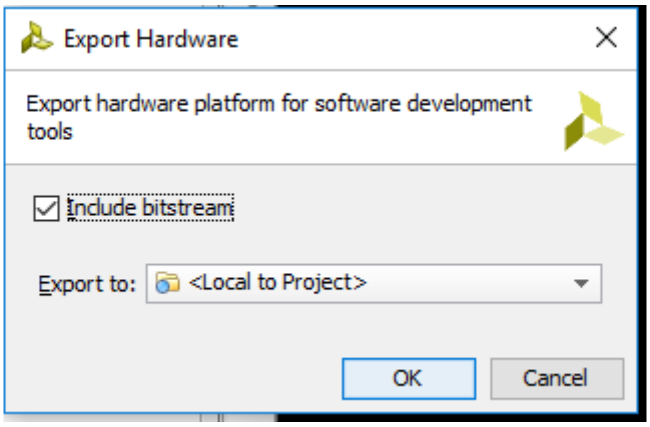
When the files are generated and exported, we can then launch the SDK (Software Development Kit) program, to design the software of our design.

Figures related to Step 4:

Exporting the hardware:



Do not forget to include the bitstream:

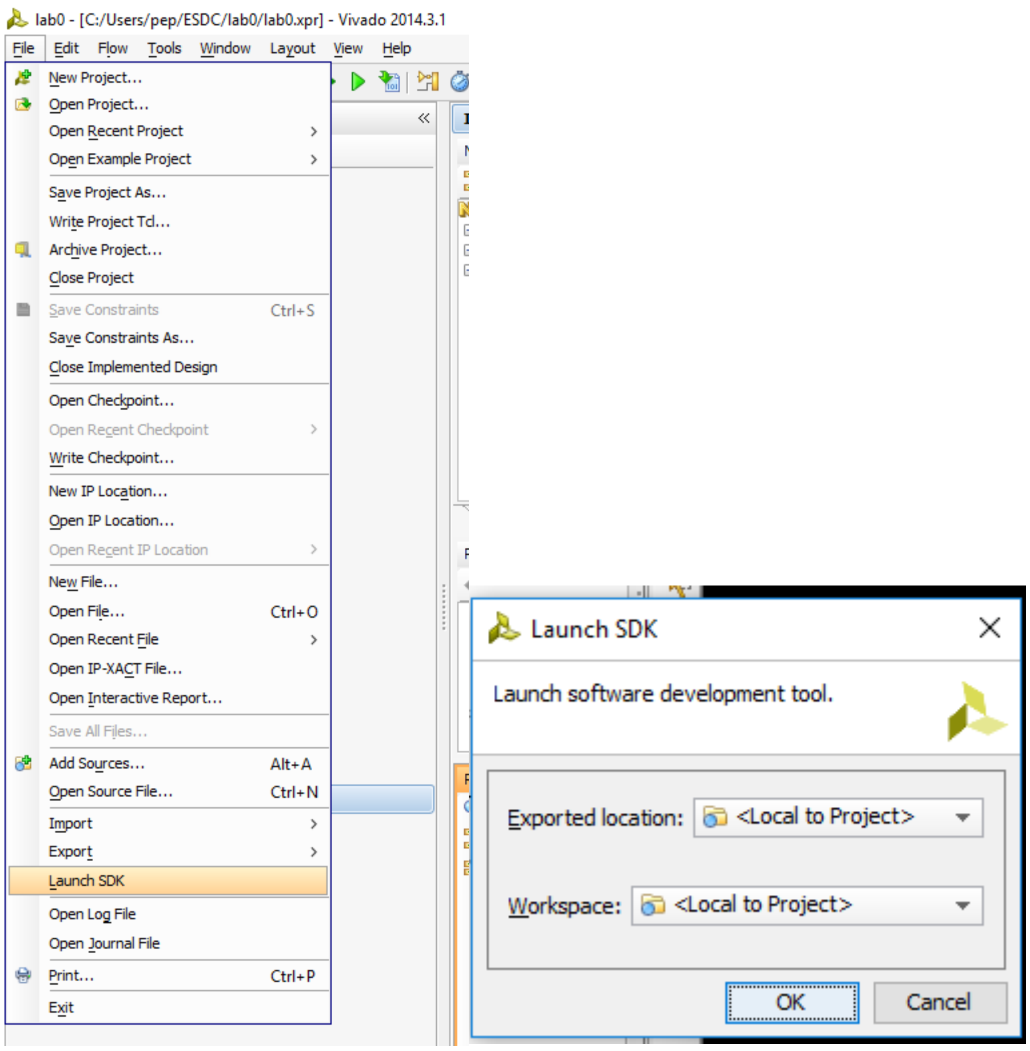


Export hardware platform for software development tools

[✓] Include bitstream

Export to: <Local to Project>

Now you can launch the SDK program:



Launch software development tool.

Exported location: <Local to Project>

Workspace: <Local to Project>

You can close the Vivado program. Let’s work with the software!

Step 5:

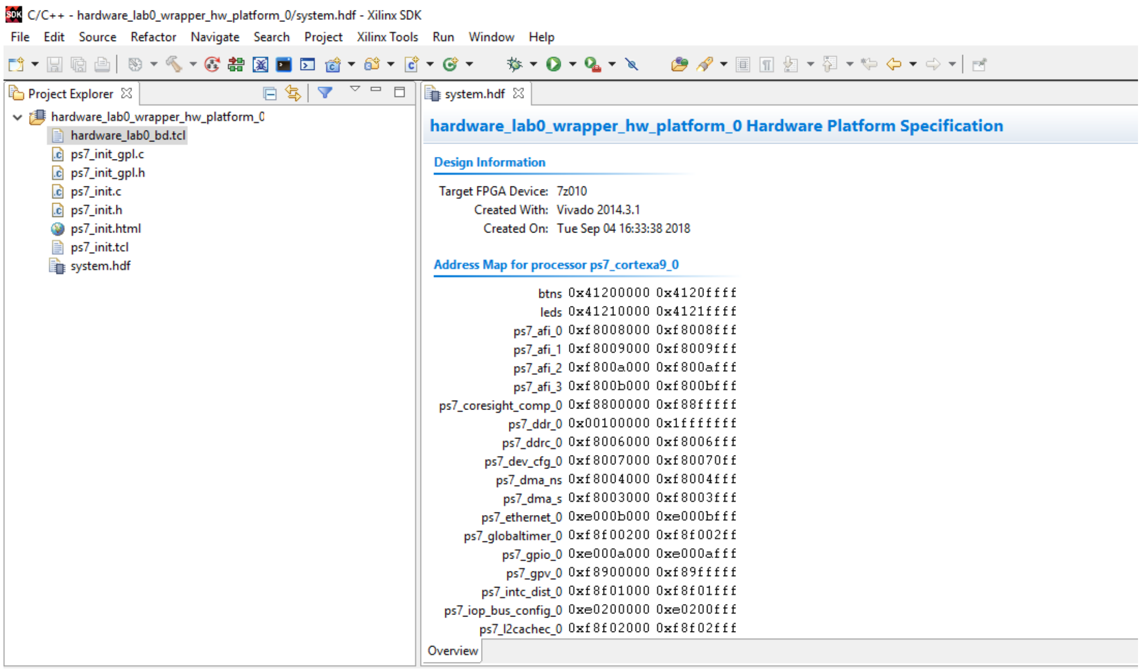
When the Software Development Kit (SDK) has finished all the initialization, you can visualize some files related to the hardware just designed. The file system.hdf contains the address map that allows to the microprocessor to access to the different hardware blocks. There are many hardware blocks that have been placed by default, but among them, it is easy to recognize the access address of the three GPIOs we have just placed in our design. In addition to this file, there are additional .c and .h files with functions declarations, constant definitions, several pre-compiler directives… all these files will be used during the microprocessor start-up. All these values have been defined while we have been designing the hardware, but in our case, they contain the default values given by the Vivado program when the Zybo personalization is used.

The programs we are going to design will be stored within an Application Project. The first step we have to do is to create one. An application project has two folders: the BSP (Board Support Package) and the user functions. The Board Support Package include, within several libraries, all the functions that we can use in our programs to interact with the different hardware blocks that we have enabled/designed our Zynq IC. These functions have been generated by the IP designers and, depending on the IP, there is related documentation that explains what each function do. We can create a BSP that works “stand alone”, which means that after the basic initialization process, all the control of the microprocessor is released to the user program; or it can work in an operating system. In this latter case, after the initialization, the control of the microprocessor is given to an operating system and the user can launch their functions for the microprocessor to execute. Once the functions are finished, control of the microprocessor is taken again by the operating system. We will explore more this second option in future laboratory assignments. In this laboratory 0, we are going to design a program working stand alone. For this reason this program will have an infinite loop (it will never end), and during this loop, we’ll check the status of the buttons. When a button is pressed, the program will read from the switches, write to the leds and resume the infinite observation of the buttons. This programs never ends: it has all the control of the activities performed by the microprocessor (once the initialization programs have been finished).

We will create an Application Project, creating a new BSP. The BSP will be “stand alone”. The tool allows us to create template .c files, that we can use to develop our one software. We are going to create a template file called “hello world”. Once this hello_world.c program has been automatically created, we will replace its code description by a .c program developed by the professors that will perform the desired function.

Figures related to stem 5:

First look at the SDK program: we can see the location of the buttons and leds GPIOs within the microprocessor Address Map.


C/C++ - hardware_lab0_wrapper_hw_platform_0/system.hdf - Xilinx SDK
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help
Project Explorer
hardware_lab0_wrapper_hw_platform_0
hardware_lab0_bd.tcl
ps7_init_gpl.c
ps7_init_gpl.h
ps7_init.c
ps7_init.h
ps7_init.html
ps7_init.tcl
system.hdf
system.hdf
Overview

hardware_lab0_wrapper_hw_platform_0 Hardware Platform Specification

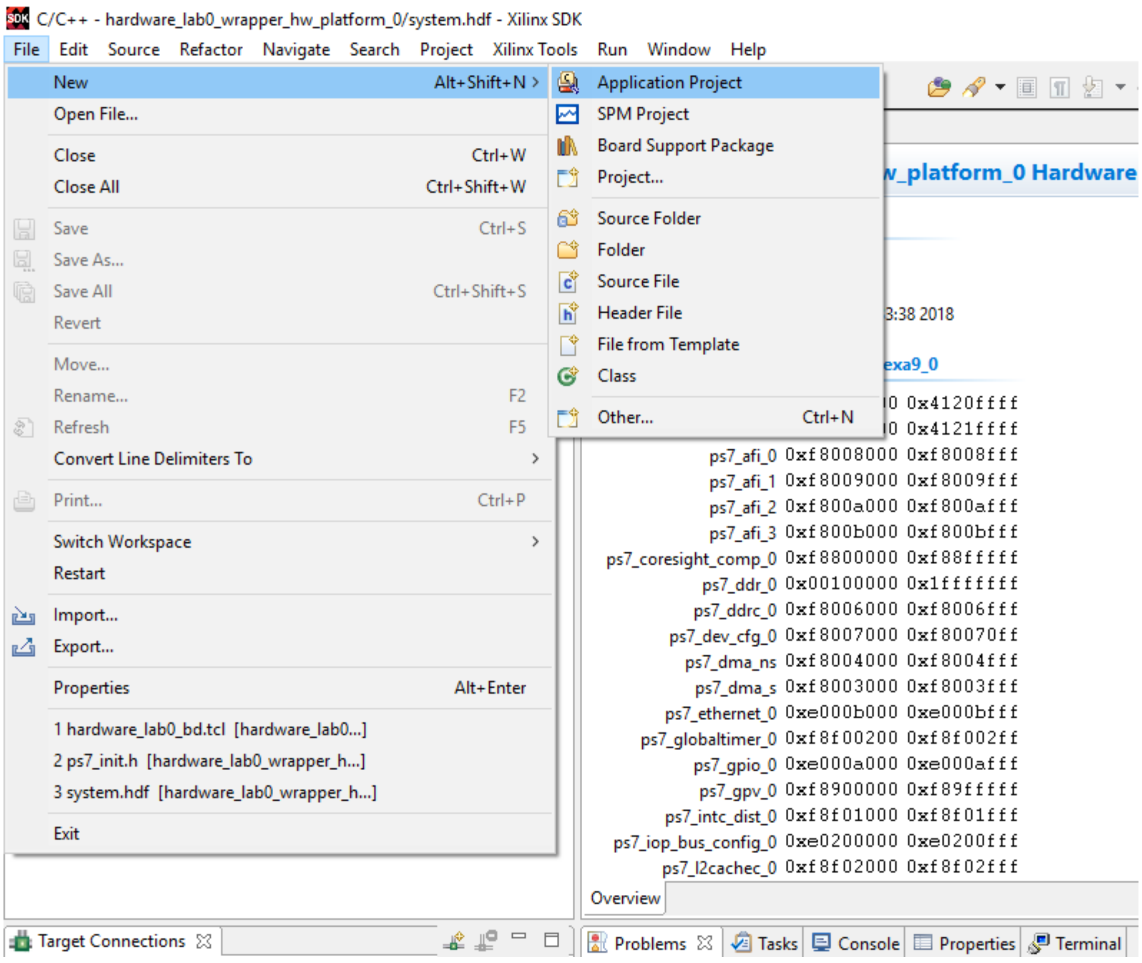
Design Information

```
Target FPGA Device: 7z010
      Created With: Vivado 2014.3.1
        Created On: Tue Sep 04 16:33:38 2018
```

Address Map for processor ps7_cortexa9_0

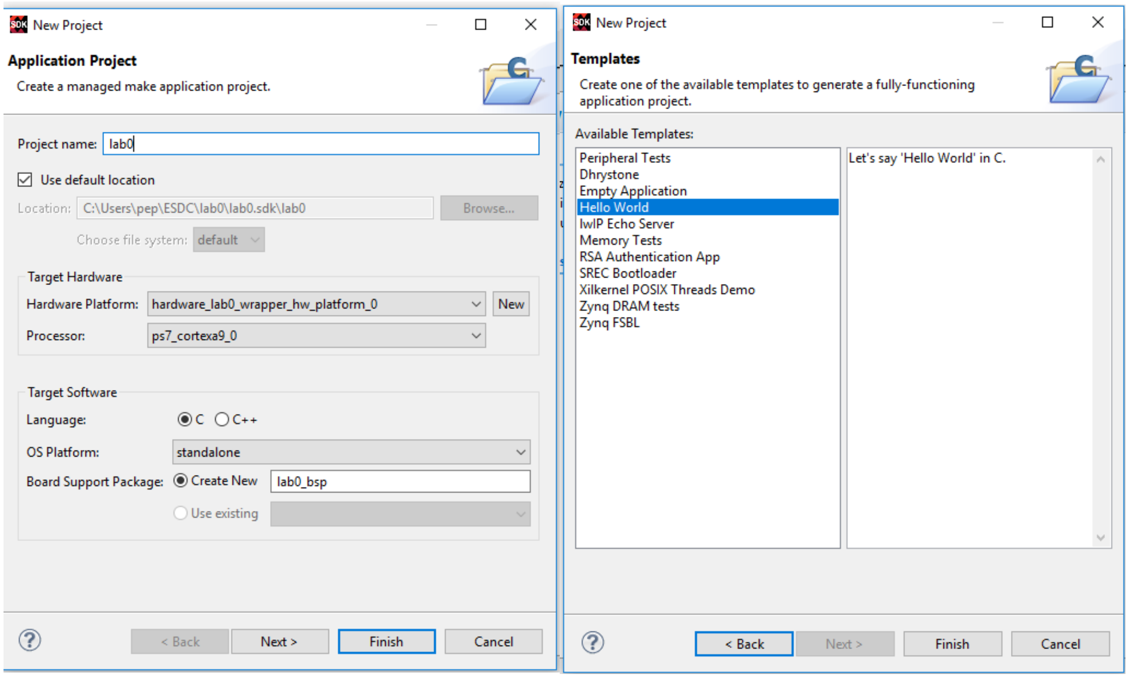
```
                 btns 0x41200000 0x4120ffff
                 leds 0x41210000 0x4121ffff
             ps7_afi_0 0xf8008000 0xf8008fff
             ps7_afi_1 0xf8009000 0xf8009fff
             ps7_afi_2 0xf800a000 0xf800afff
             ps7_afi_3 0xf800b000 0xf800bfff
   ps7_coresight_comp_0 0xf8800000 0xf88fffff
             ps7_ddr_0 0x00100000 0x1fffffff
            ps7_ddrc_0 0xf8006000 0xf8006fff
         ps7_dev_cfg_0 0xf8007000 0xf80070ff
          ps7_dma_ns 0xf8004000 0xf8004fff
           ps7_dma_s 0xf8003000 0xf8003fff
        ps7_ethernet_0 0xe000b000 0xe000bfff
     ps7_globaltimer_0 0xf8f00200 0xf8f002ff
            ps7_gpio_0 0xe000a000 0xe000afff
             ps7_gpv_0 0xf8900000 0xf89fffff
        ps7_intc_dist_0 0xf8f01000 0xf8f01fff
   ps7_iop_bus_config_0 0xe0200000 0xe0200fff
         ps7_l2cachec_0 0xf8f02000 0xf8f02fff
```

Creating New Application Project:



```
                ps7_afi_0 0xf8008000 0xf8008fff
                ps7_afi_1 0xf8009000 0xf8009fff
                ps7_afi_2 0xf800a000 0xf800afff
                ps7_afi_3 0xf800b000 0xf800bfff
     ps7_coresight_comp_0 0xf8800000 0xf88fffff
                ps7_ddr_0 0x00100000 0x1fffffff
               ps7_ddrc_0 0xf8006000 0xf8006fff
            ps7_dev_cfg_0 0xf8007000 0xf80070ff
               ps7_dma_ns 0xf8004000 0xf8004fff
                ps7_dma_s 0xf8003000 0xf8003fff
           ps7_ethernet_0 0xe000b000 0xe000bfff
        ps7_globaltimer_0 0xf8f00200 0xf8f002ff
               ps7_gpio_0 0xe000a000 0xe000afff
                ps7_gpv_0 0xf8900000 0xf89fffff
           ps7_intc_dist_0 0xf8f01000 0xf8f01fff
      ps7_iop_bus_config_0 0xe0200000 0xe0200fff
          ps7_l2cachec_0 0xf8f02000 0xf8f02fff
```

Name of the new Application Project: lab0. Standalone. New BSP is created. Click on NEXT. Select the template Hello World. Finish.

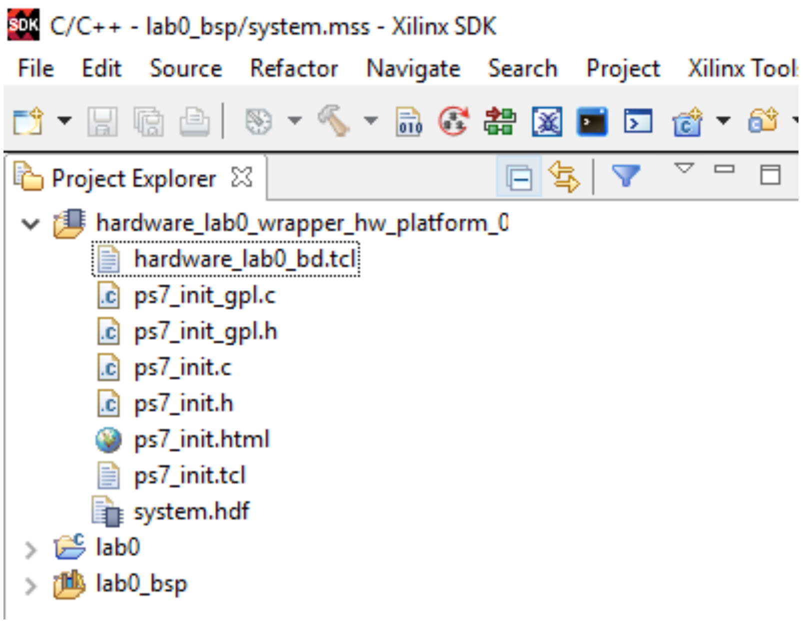

New Project
Application Project
Create a managed make application project.
< Back
Next >
Finish
Cancel
New Project
Templates
Create one of the available templates to generate a fully-functioning application project.
< Back
Next >
Finish
Cancel

Project name: lab0
☑ Use default location
Location: C:\Users\pep\ESDC\lab0\lab0.sdk\lab0
Choose file system: default
Target Hardware
Hardware Platform: hardware_lab0_wrapper_hw_platform_0
Processor: ps7_cortexa9_0
Target Software
Language: [✓ C] C++
OS Platform: standalone
Board Support Package: [✓ Create New] lab0_bsp
Use existing

Available Templates:
Peripheral Tests
Dhrystone
Empty Application
[✓ Hello World]
lwIP Echo Server
Memory Tests
RSA Authentication App
SREC Bootloader
Xilkernel POSIX Threads Demo
Zynq DRAM tests
Zynq FSBL

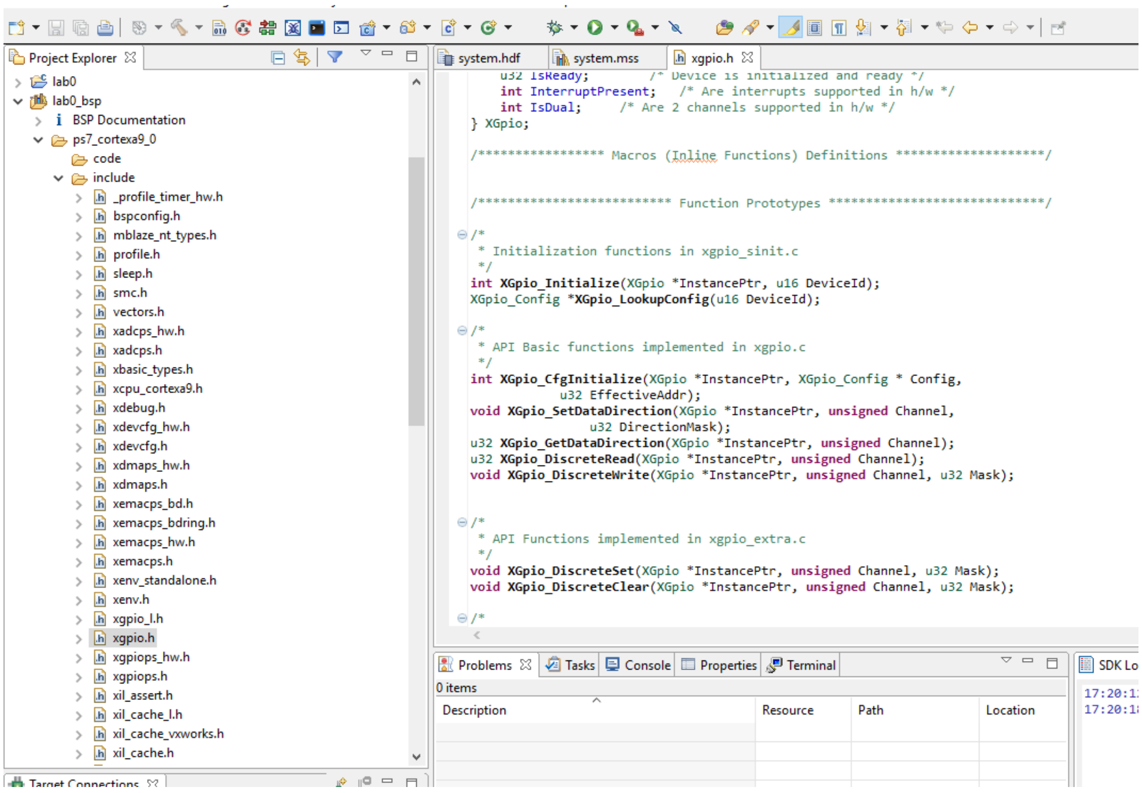
Let's say 'Hello World' in C.

Two new folders created:


C/C++ - lab0_bsp/system.mss - Xilinx SDK

```
hardware_lab0_wrapper_hw_platform_0
    hardware_lab0_bd.tcl
    ps7_init_gpl.c
    ps7_init_gpl.h
    ps7_init.c
    ps7_init.h
    ps7_init.html
    ps7_init.tcl
    system.hdf
lab0
lab0_bsp
```

If we explore the BSP folder, we see functions to interact with hardware blocks. We see some of the functions that we can use to interact with the AXI GPIOs:



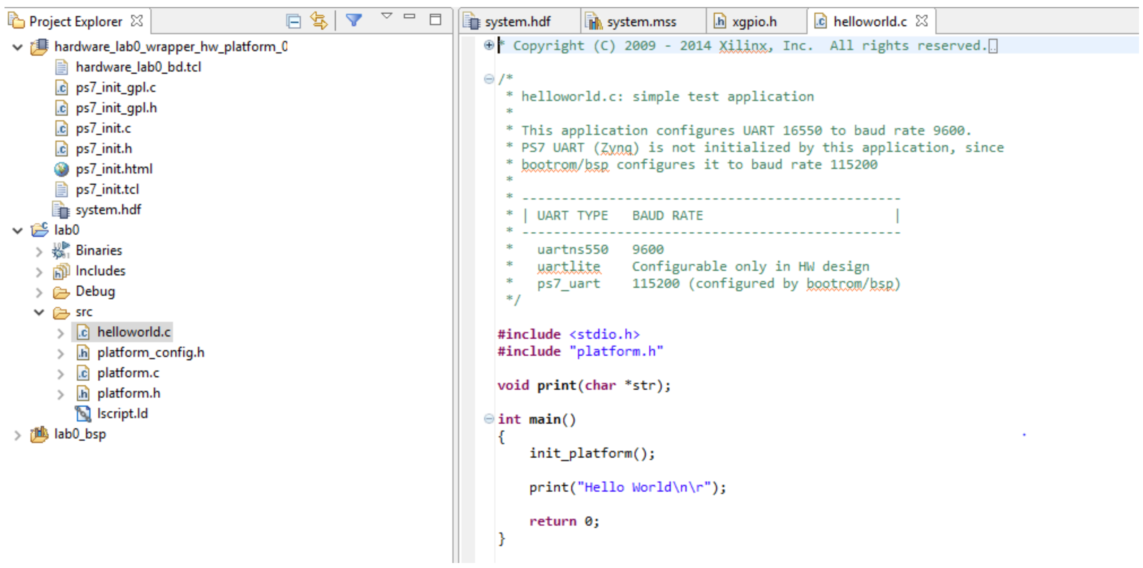
```
lab0
lab0_bsp
    BSP Documentation
    ps7_cortexa9_0
        code
        include
            _profile_timer_hw.h
            bspconfig.h
            mblaze_nt_types.h
            profile.h
            sleep.h
            smc.h
            vectors.h
            xadcps_hw.h
            xadcps.h
            xbasic_types.h
            xcpu_cortexa9.h
            xdebug.h
            xdevcfg_hw.h
            xdevcfg.h
            xdmaps_hw.h
            xdmaps.h
            xemacps_bd.h
            xemacps_bdring.h
            xemacps_hw.h
            xemacps.h
            xenv_standalone.h
            xenv.h
            xgpio_l.h
            xgpio.h
            xgpiops_hw.h
            xgpiops.h
            xil_assert.h
            xil_cache_l.h
            xil_cache_vxworks.h
            xil_cache.h
```

```
    u32 IsReady;          /* Device is initialized and ready */
    int InterruptPresent;   /* Are interrupts supported in h/w */
    int IsDual;      /* Are 2 channels supported in h/w */
} XGpio;

/*************** Macros (Inline Functions) Definitions *******************/


/************************* Function Prototypes ****************************/

/*
 * Initialization functions in xgpio_sinit.c
 */
int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
XGpio_Config *XGpio_LookupConfig(u16 DeviceId);

/*
 * API Basic functions implemented in xgpio.c
 */
int XGpio_CfgInitialize(XGpio *InstancePtr, XGpio_Config * Config,
            u32 EffectiveAddr);
void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel,
                u32 DirectionMask);
u32 XGpio_GetDataDirection(XGpio *InstancePtr, unsigned Channel);
u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
void XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Mask);


/*
 * API Functions implemented in xgpio_extra.c
 */
void XGpio_DiscreteSet(XGpio *InstancePtr, unsigned Channel, u32 Mask);
void XGpio_DiscreteClear(XGpio *InstancePtr, unsigned Channel, u32 Mask);

/*
```

```
0 items
Description    Resource    Path    Location
```

```
17:20:1
17:20:1
```

If we explore the folder with the user application programs: some programs automatically generated by the tool. Among then, the helloworld.c, which only prints a message to the standard output console.



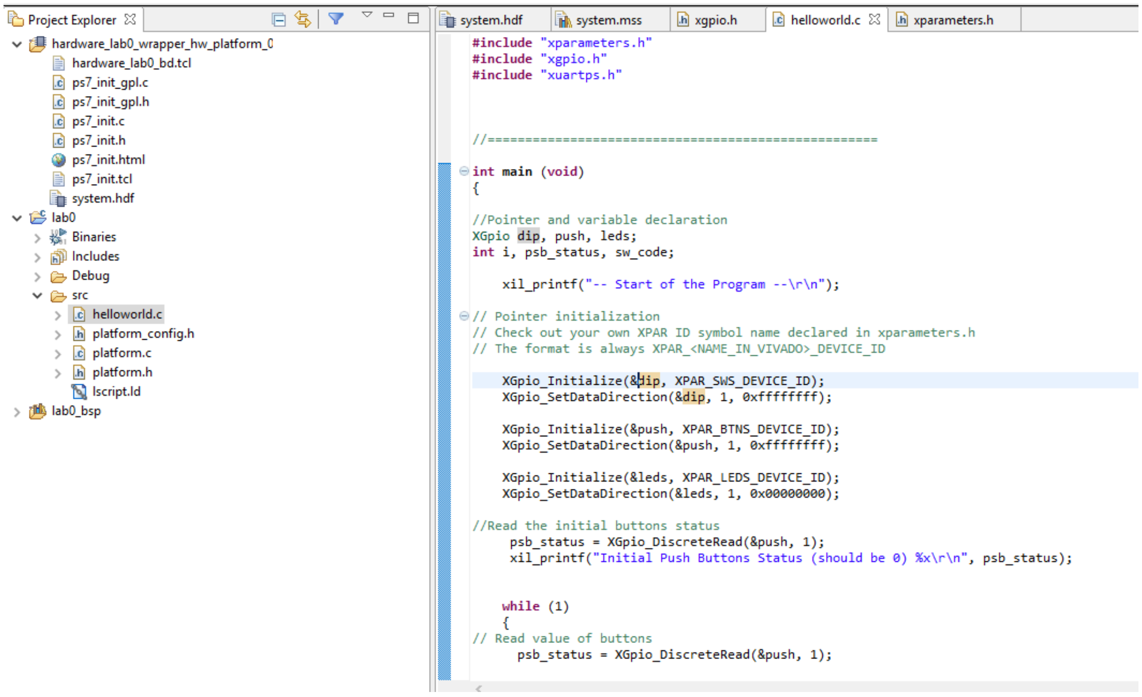
```
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"

void print(char *str);

int main()
{
    init_platform();

    print("Hello World\n\r");

    return 0;
}
```

You should replace this .c program by this code:

```
#include "xparameters.h"
#include "xgpio.h"
#include "xuartps.h"



//====================================================

int main (void)
{

//Pointer and variable declaration
XGpio dip, push, leds;
int i, psb_status, sw_code;

    xil_printf("-- Start of the Program --\r\n");

// Pointer initialization
// Check out your own XPAR ID symbol name declared in xparameters.h
// The format is always XPAR_<NAME_IN_VIVADO>_DEVICE_ID

       XGpio_Initialize(&dip, XPAR_SWS_DEVICE_ID);
       XGpio_SetDataDirection(&dip, 1, 0xffffffff);

       XGpio_Initialize(&push, XPAR_BTNS_DEVICE_ID);
       XGpio_SetDataDirection(&push, 1, 0xffffffff);

       XGpio_Initialize(&leds, XPAR_LEDS_DEVICE_ID);
       XGpio_SetDataDirection(&leds, 1, 0x00000000);

//Read the initial buttons status
        psb_status = XGpio_DiscreteRead(&push, 1);
        xil_printf("Initial Push Buttons Status (should be 0) %x\r\n", psb_status);


       while (1)
       {
// Read value of buttons
         psb_status = XGpio_DiscreteRead(&push, 1);

         if(psb_status)
         {
                xil_printf("A button has been pressed: %x\r\n", psb_status);
                // Read value of switches
                sw_code = XGpio_DiscreteRead(&dip, 1);

                xil_printf("The code detected is: %x\r\n", sw_code);

                // Write code of switches to leds (example)
                 XGpio_DiscreteWrite(&leds, 1, sw_code);
         }
         // Waiting cycles
         for (i=0; i<9999999; i++);
       }

}
```

You need to understand this program. The functions that are used are defined in the BSP libraries. You should explore the .h files defined at the beginning. Specially the xparameters.h where the constants `XPAR_SWS_DEVICE_ID`, `XPAR_LEDS_DEVICE_ID` and `XPAR_BTNS_DEVICE_ID` are defined. Thanks to the names used to identify the AXI GPIOs, we can now recognize the constant definition that refer to each hardware GPIO. Description of what the different functions do can be found either in the .h files (e.g. all the gpio related functions are described in the `xgpio.h` file or you can google it. Do not hesitate to ask for help. Take your time to understand what this program do. The function xil_printf will be used to promp messages on the PC console. Later on we will explain how to configure this console in order to receive the messages from the Zynq standard output through a virtual serial connection.

When this new .c code is saved, it is automatically complied and linked.



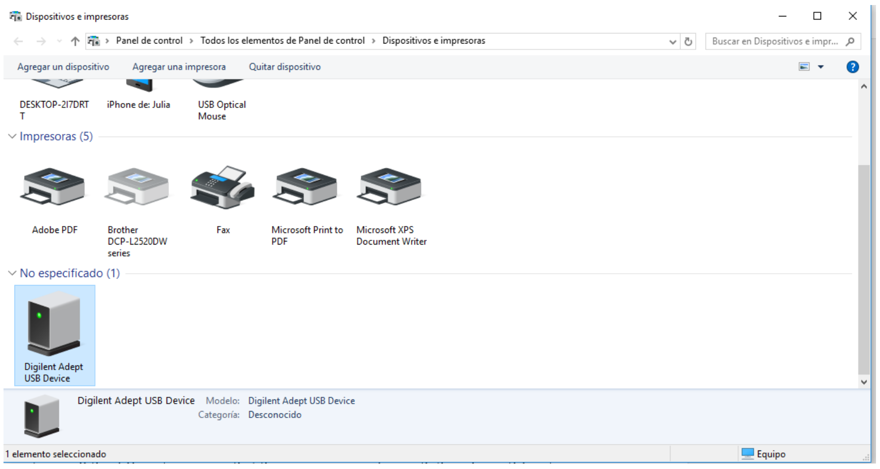
```
#include "xparameters.h"
#include "xgpio.h"
#include "xuartps.h"



//====================================================

int main (void)
{

//Pointer and variable declaration
XGpio dip, push, leds;
int i, psb_status, sw_code;

    xil_printf("-- Start of the Program --\r\n");

// Pointer initialization
// Check out your own XPAR ID symbol name declared in xparameters.h
// The format is always XPAR_<NAME_IN_VIVADO>_DEVICE_ID

    XGpio_Initialize(&dip, XPAR_SWS_DEVICE_ID);
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_BTNS_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xffffffff);

    XGpio_Initialize(&leds, XPAR_LEDS_DEVICE_ID);
    XGpio_SetDataDirection(&leds, 1, 0x00000000);

//Read the initial buttons status
     psb_status = XGpio_DiscreteRead(&push, 1);
     xil_printf("Initial Push Buttons Status (should be 0) %x\r\n", psb_status);


    while (1)
    {
// Read value of buttons
      psb_status = XGpio_DiscreteRead(&push, 1);
```

Step 6:

Congratulations! Both hardware and software have been designed… now it is time to play and have fun with the design. We are going to connect the Zybo board to the PC through the USB cable provided. Make sure that in the Zybo board, the jumper JP7 – Top left corner- is in USB mode and the JP5 – top right corner- is in JTAG mode (they should already be in that position. Refer to either the board manual or the documentation available in Atenea for further information). If it is the first time that the board is connected to the computer, Windows will install a driver… which may take some time. Going to the device and printers set up, you can see when the board is recognized and ready!. Once connected, first we will transfer to the board the hardware configuration file. This will configure the FPGA to implement the hardware blocks as specified in Vivado and will interconnect the microprocessor with these blocks as specified. Second, we will transfer the software program. We have as well to configure a console, in order to be able to see all the different messages that the program is sending with the xil_printf function.

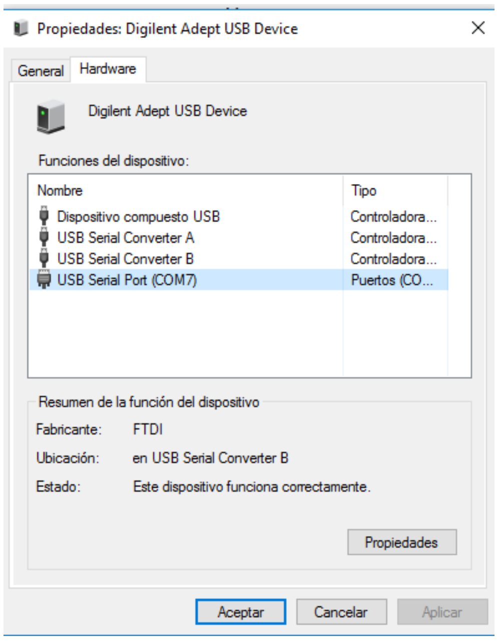
Figures associated to step 6:

When the board driver is installed, you'll see:


Dispositivos e impresoras
Panel de control › Todos los elementos de Panel de control › Dispositivos e impresoras
Buscar en Dispositivos e impr...
Agregar un dispositivo
Agregar una impresora
Quitar dispositivo
DESKTOP-2I7DRTT
iPhone de: Julia
USB Optical Mouse
Impresoras (5)
Adobe PDF
Brother DCP-L2520DW series
Fax
Microsoft Print to PDF
Microsoft XPS Document Writer
No especificado (1)
Digilent Adept USB Device
Equipo

Digilent Adept USB Device Modelo: Digilent Adept USB Device
Categoría: Desconocido

1 elemento seleccionado


Propiedades: Digilent Adept USB Device

General | Hardware

Digilent Adept USB Device

Funciones del dispositivo:

Nombre	Tipo
Dispositivo compuesto USB	Controladora...
USB Serial Converter A	Controladora...
USB Serial Converter B	Controladora...
USB Serial Port (COM7)	Puertos (CO...

Resumen de la función del dispositivo

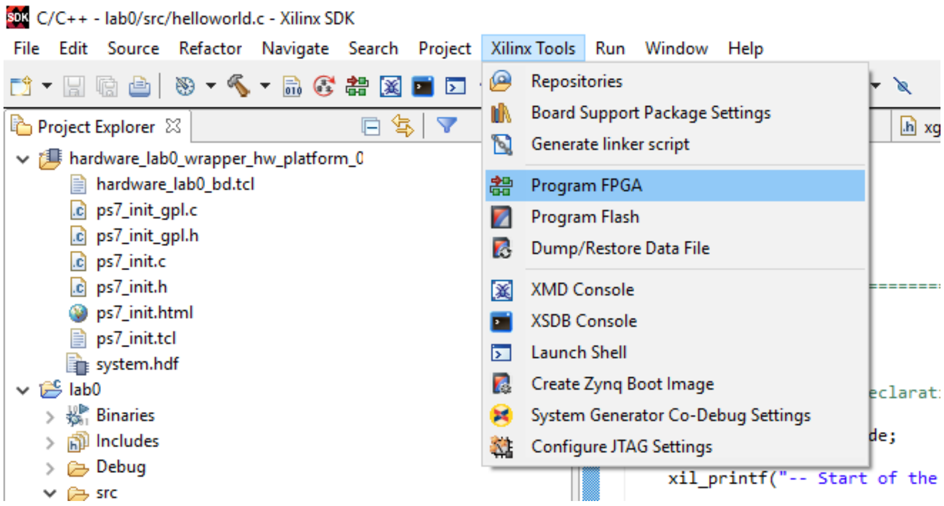
Fabricante:	FTDI
Ubicación:	en USB Serial Converter B
Estado:	Este dispositivo funciona correctamente.

Propiedades

Aceptar | Cancelar | Aplicar

Always choose a COM port other than 1 and 3, which are already assigned to keyboard and mouse, respectively. In my computer, COM7 was the assigned serial port… but in your case, you may have COM10 or COM13… or other.

Transfering the hardware configuration file (bitstream) for the FPGA:

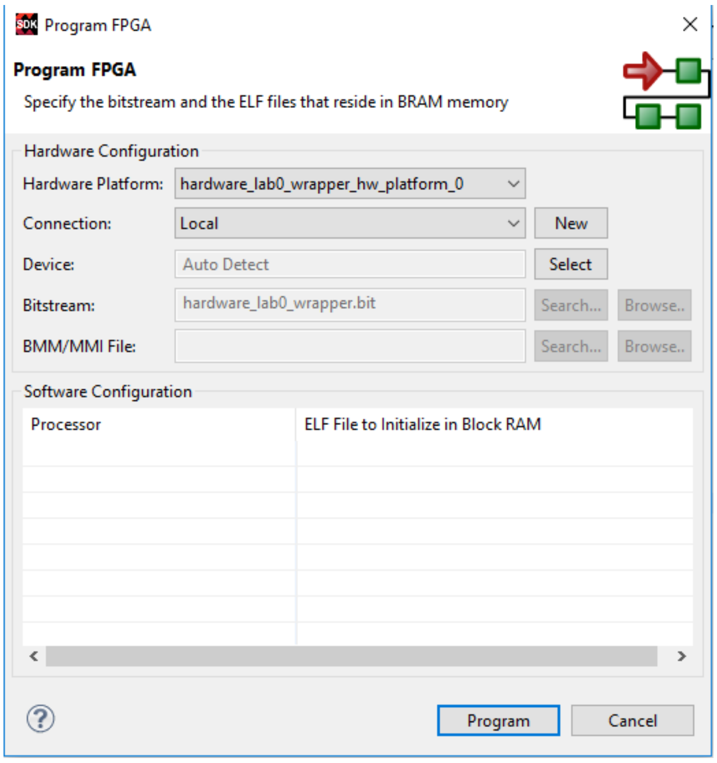

C/C++ - lab0/src/helloworld.c - Xilinx SDK

File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help

Project Explorer
- hardware_lab0_wrapper_hw_platform_0
  - hardware_lab0_bd.tcl
  - ps7_init_gpl.c
  - ps7_init_gpl.h
  - ps7_init.c
  - ps7_init.h
  - ps7_init.html
  - ps7_init.tcl
  - system.hdf
- lab0
  - Binaries
  - Includes
  - Debug
  - src

Xilinx Tools menu:
- Repositories
- Board Support Package Settings
- Generate linker script
- Program FPGA
- Program Flash
- Dump/Restore Data File
- XMD Console
- XSDB Console
- Launch Shell
- Create Zynq Boot Image
- System Generator Co-Debug Settings
- Configure JTAG Settings

```
xil_printf("-- Start of the
```



Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

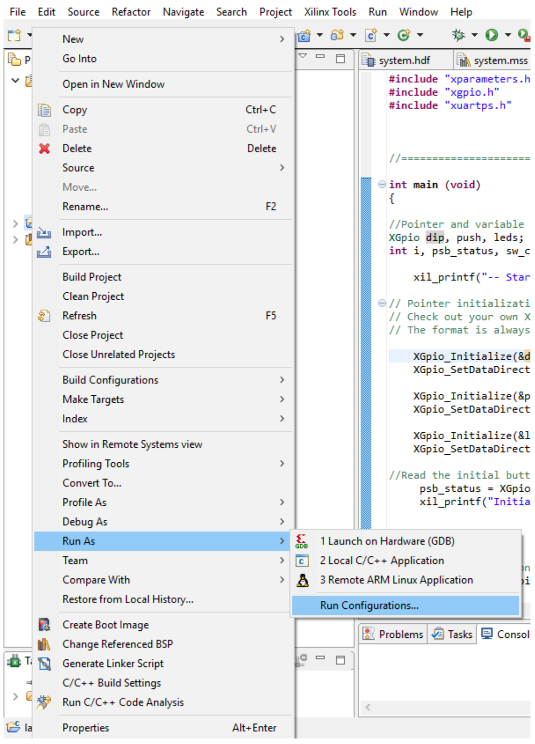
Hardware Platform:	hardware_lab0_wrapper_hw_platform_0
Connection:	Local
Device:	Auto Detect
Bitstream:	hardware_lab0_wrapper.bit
BMM/MMI File:	

Software Configuration

Processor	ELF File to Initialize in Block RAM

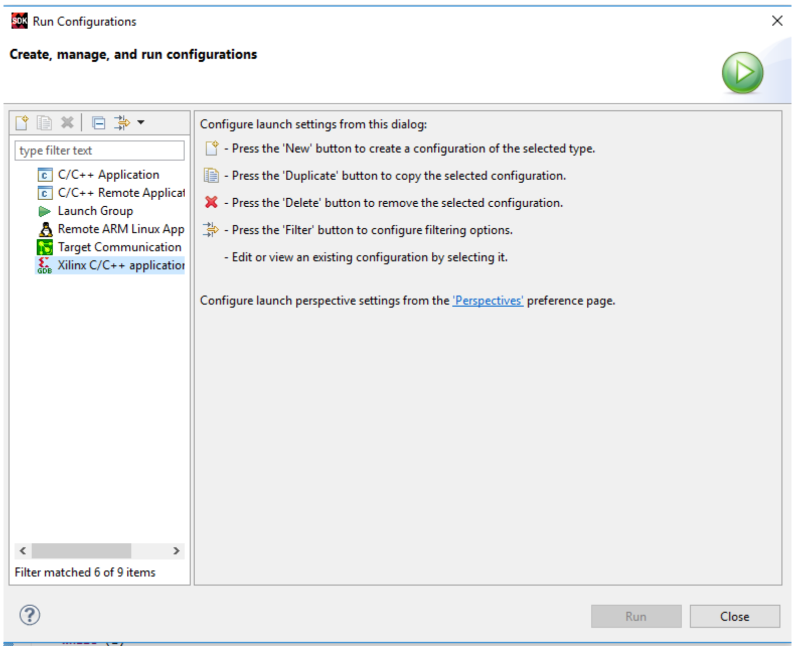
If successful, the Green LED DONE should light in the Zybo board.

To transfer the software program. He have to create a new run configuration. Right click on lab0 (NOT lab0_BSP!!!!).



```
#include "xparameters.h
#include "xgpio.h"
#include "xuartps.h"


//=====================

int main (void)
{

//Pointer and variable
XGpio dip, push, leds;
int i, psb_status, sw_c

   xil_printf("-- Star

// Pointer initializati
// Check out your own X
// The format is always

   XGpio_Initialize(&d
   XGpio_SetDataDirect

   XGpio_Initialize(&p
   XGpio_SetDataDirect

   XGpio_Initialize(&l
   XGpio_SetDataDirect

//Read the initial butt
    psb_status = XGpio
    xil_printf("Initia
```

New window pops up.



Run Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Applicat
- Launch Group
- Remote ARM Linux App
- Target Communication
- Xilinx C/C++ applicatior

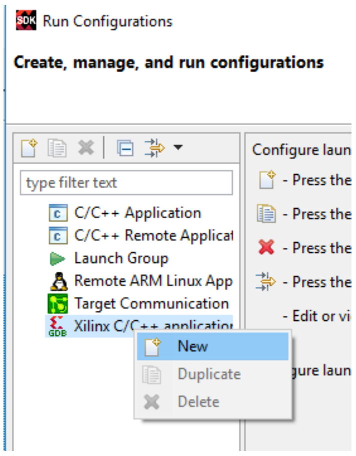
Filter matched 6 of 9 items

Configure launch settings from this dialog:

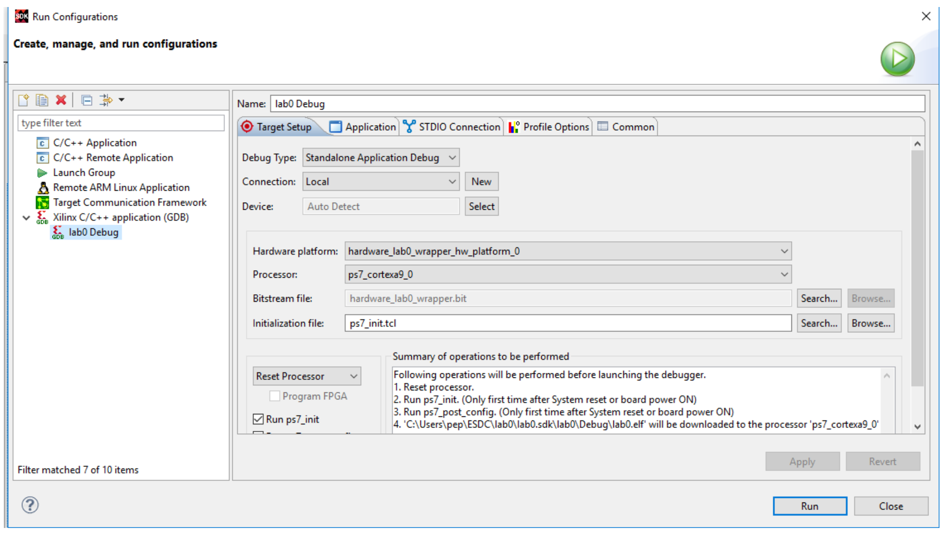
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Right click Xilinx C/C++ applications:



Lab0 debug is created:



Run Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- Launch Group
- Remote ARM Linux Application
- Target Communication Framework
- Xilinx C/C++ application (GDB)
  - lab0 Debug

Filter matched 7 of 10 items

Name: lab0 Debug

Target Setup | Application | STDIO Connection | Profile Options | Common

Debug Type: Standalone Application Debug

Connection: Local

Device: Auto Detect

Hardware platform: hardware_lab0_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Bitstream file: hardware_lab0_wrapper.bit

Initialization file: ps7_init.tcl

Reset Processor

☐ Program FPGA

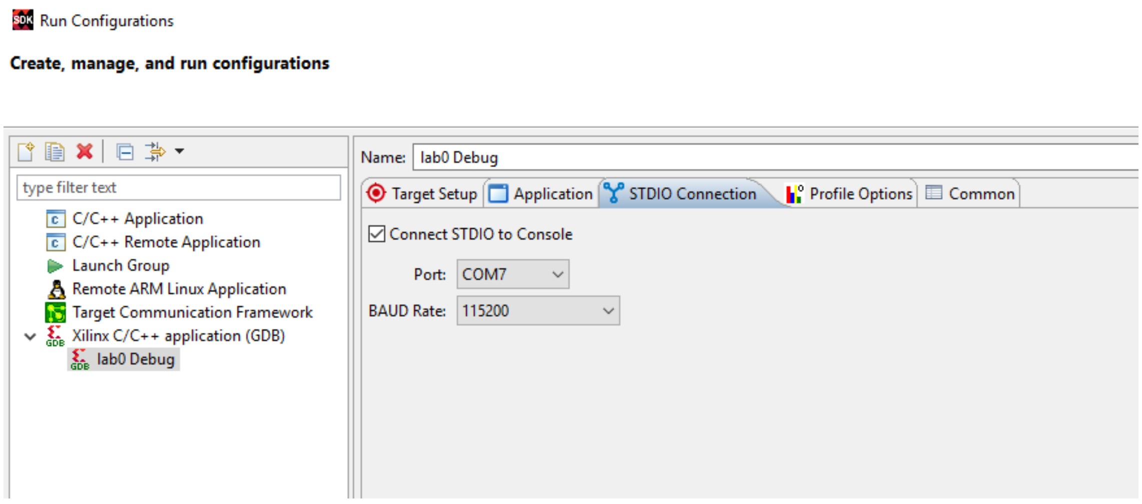
☑ Run ps7_init

Summary of operations to be performed

```
Following operations will be performed before launching the debugger.
1. Reset processor.
2. Run ps7_init. (Only first time after System reset or board power ON)
3. Run ps7_post_config. (Only first time after System reset or board power ON)
4. 'C:\Users\pep\ESDC\lab0\lab0.sdk\lab0\Debug\lab0.elf' will be downloaded to the processor 'ps7_cortexa9_0'
```

Tab application: we see the current project name.

Important: Click on tab STDIO to configure virtual serial connection. Should be as:


Run Configurations

Create, manage, and run configurations

Name: lab0 Debug

Target Setup | Application | STDIO Connection | Profile Options | Common

☑ Connect STDIO to Console

Port: COM7

BAUD Rate: 115200

type filter text
C/C++ Application
C/C++ Remote Application
Launch Group
Remote ARM Linux Application
Target Communication Framework
Xilinx C/C++ application (GDB)
lab0 Debug

(You should first check which COM# you have).

Click RUN! Check the console TAB. Change the code in the switches. Press buttons. Observe LEDs and console messages. This laboratory is finished. Your professor should verify that the design is working.

MARK

6/10: Realization of this tutorial the assigned laboratory day or the consecutive one.

5/10: Realization of this tutorial 2 sessions after the assigned day.

Extra marks (optional work). You have until the last laboratory day to present any of these improvements.

+1 point: If you keep pressed a button while the program is working, the system continuously reads the value introduced in the switches. Modify the program in such a way that no matter for how long the button is pressed, there is only one read of the buttons.

+3 points: Implement this design only using hardware. +1 Compare the performances of both realizations in terms of power consumption and device occupancy (Vivado implementation report).

+3 points: Implement the following game: While button 0 is pressed, a ramdom 4 bits number is generated and stored. The user should guess this random number stored. The guess number is introduced through the switches and the user should press any other button when ready. Then, the system compares the number introduced with the one generated randomly and informs, with the leds, if the number generated is greater than the introduced (right led on), smaller (left led on) or equal (two central leds on). The user wins when the random number is detected. The number of possible iterations can be limited. +1 point: modify the game for two players.

Fell free of suggesting other design improvements/variations to your laboratory professor.