

School on Fully-Programmable Systems-on-Chip for Scientific Instrumentation



LABORATORY

GPIO IP Cores - PS Rd/Wr

Prepared by
C. Sisterna & L. Crespo
ICTP-MLAB

Adding IP cores in Zynq PL

Introduction

This lab guides you through the process of creating a system with two **GPIOs** IP Cores in the **PL** part of the **Zynq**. These **GPIOs** will be controlled by the 'C' application that will run in the **PS**. One of the **GPIOs**, the **GPIO**, configured as an input, will be connected to the switches available in the ZedBoard, and the other **GPIO IP**, will be configured as an output, and will be connected to LEDs. There also will be a reading from an external push-button that is connected to one of the PL I/O pin. This signal will be routed from the **PL** to the **PS** through the **EMIO**. The reading of the switches and the push-button and the writing to the **LEDs** will be done by a 'C' user-written application and the result will be displayed on the Host PC's monitor through a serial communication between the Host and the ZedBoard.

Objectives

After completing this lab, you will be able to:

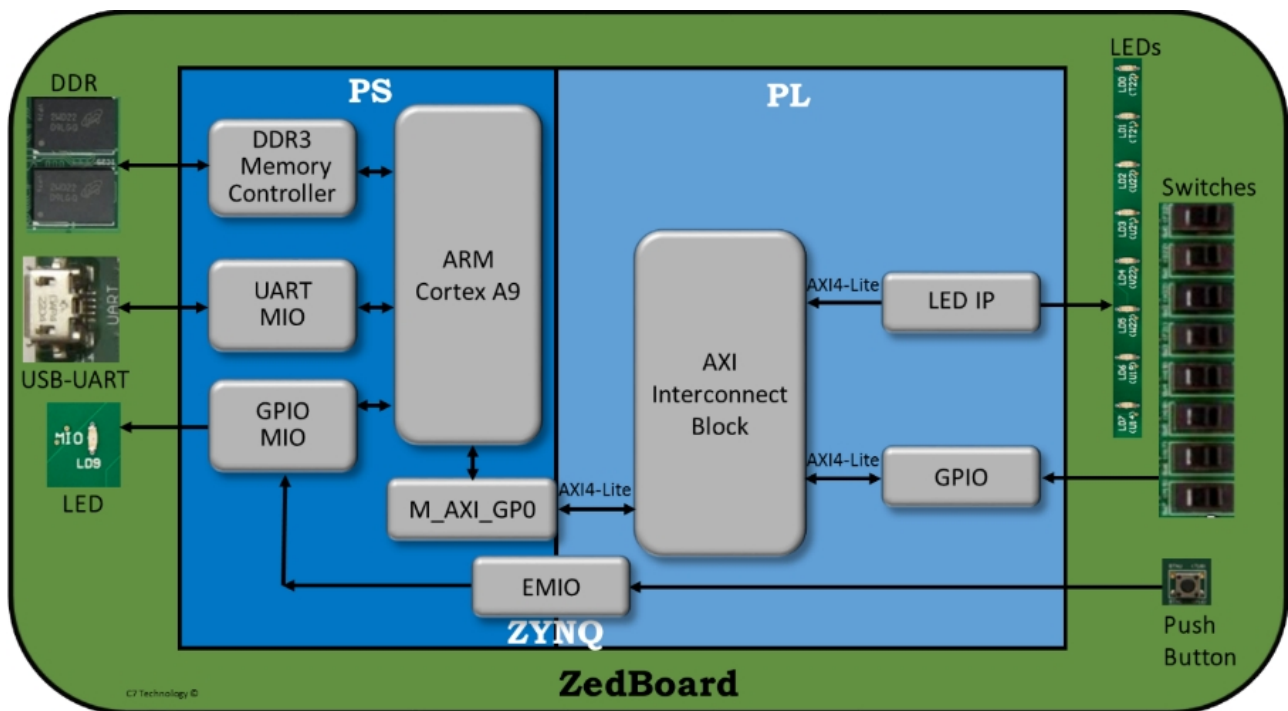
- Configure the *GPIO Master* port
- Add GPIO IP Core in the Programmable Logic (PL) section
- Route the GPIO signal from the PL to the PS by using EMIO
- Write a 'C' application program in the SDK environment
- Configure the FPGA from the SDK software
- Test in hardware the design

Procedure

This lab comprises eight primary steps: You will create a top-level project using Vivado Development Suite, create a processor based system, add two instances of GPIO IPs, validate the design, generate the bitstream, export the design to the SDK, create an application in the SDK and test the design in hardware (ZedBoard).

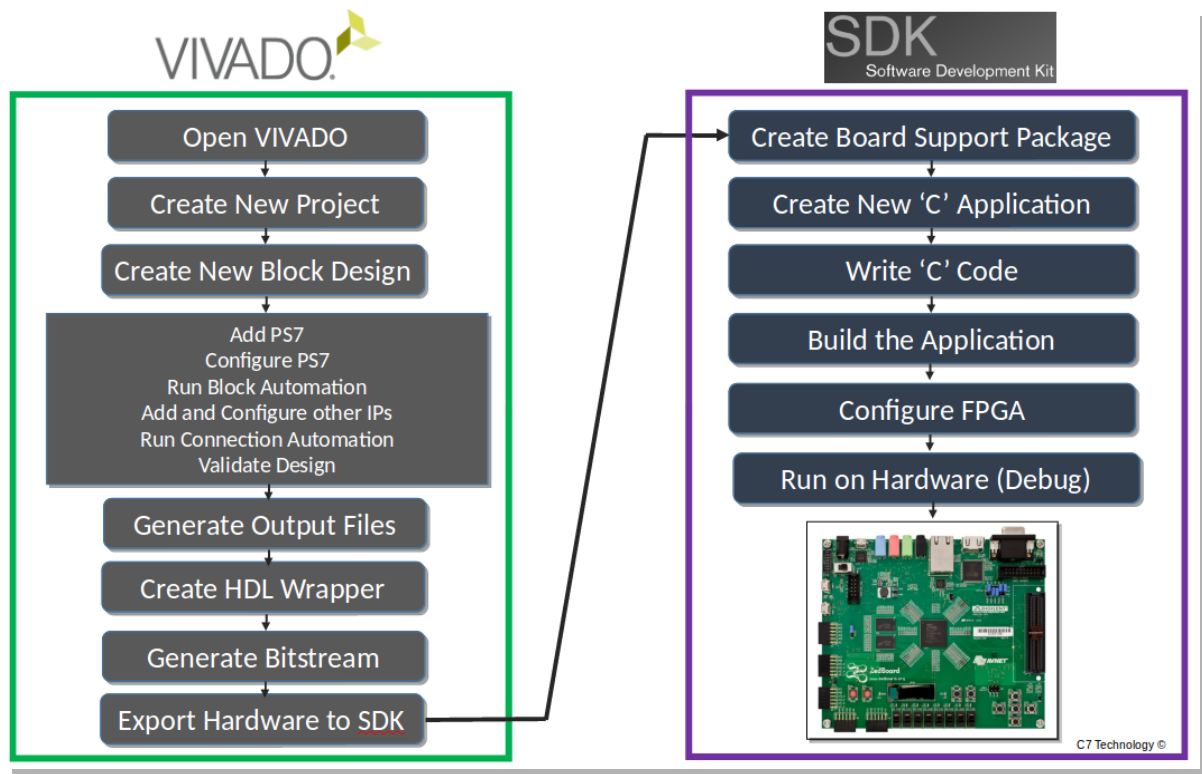
Design Description

The following block diagram represents the design to be created in this laboratory.



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 As -> Launch on Hardware (GDB)** process to program the PS part of the Zynq with the respective *.elf file, and automatically execute the 'C' code in the processor.

Section I

Create a Vivado Project

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

1. Open Vivado Design Suite.
2. Click **Create 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	gpio_in_out
	Project Location	C:/...../..labs/lab_gpio_inout
	Create Project Subdirectory	Check this option.
Click Next		
Project Type	Specify RTL	Select RTL. 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, Rev. D.
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.

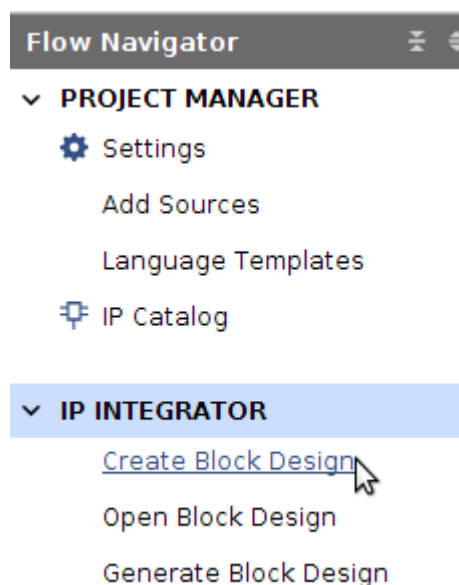
The board selected during the project creation, in this case the **ZedBoard**, has a direct impact on how the **IP Integrator**, within the **Vivado**, executes. **IP Integrator** is board aware and it will automatically assigns dedicated PS IO ports to physical pin locations mapped to the specific board 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.

Adding and Configuring PS7

Objective Create a block design using IP Integrator. Add and configure the PS7 block.

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

3.1. Click **Create Block Design** in the **Flow Navigator** pane under the **IP Integrator**.



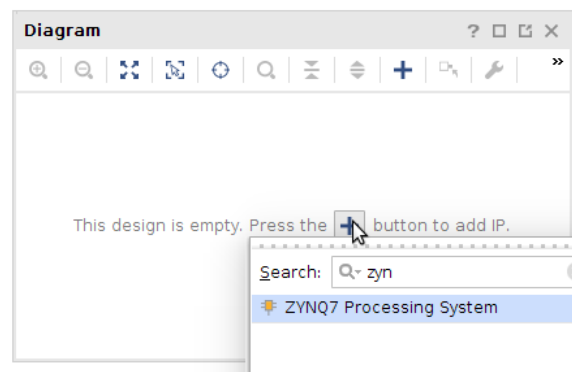
3.2. Type in **lab_gpio_in_out** as **Design Name** in the **Create Block Design** window.

3.3. 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.

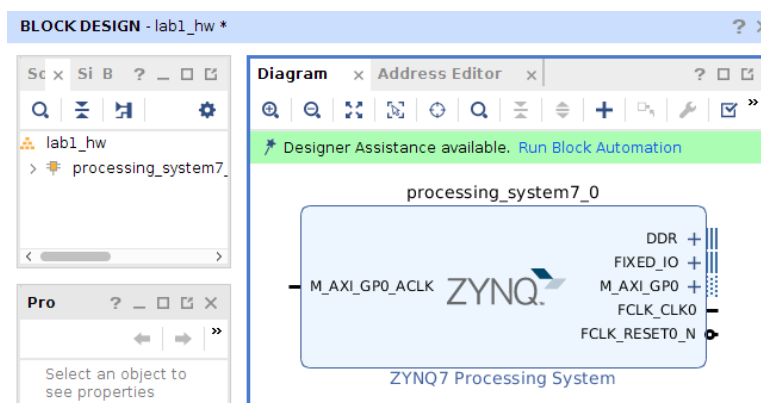
4. Let's begin creating a new embedded system in by first adding a ZYNQ7 Processing System block.

4.1. 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.

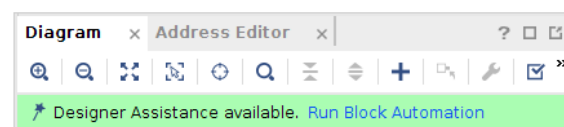
4.2. An small window will come up showing the available **IPs** (they are the Intellectual Property cores that are already available; we will see later, in other lab, how to create and add our own IP). To search for the **PS7 IP** core, we can either scroll down to the very bottom of the IP list or search the **PS7 IP** using the keyword **zynq**. Double click on the **ZYNQ7 Processing System** to select it and add it to the canvas.



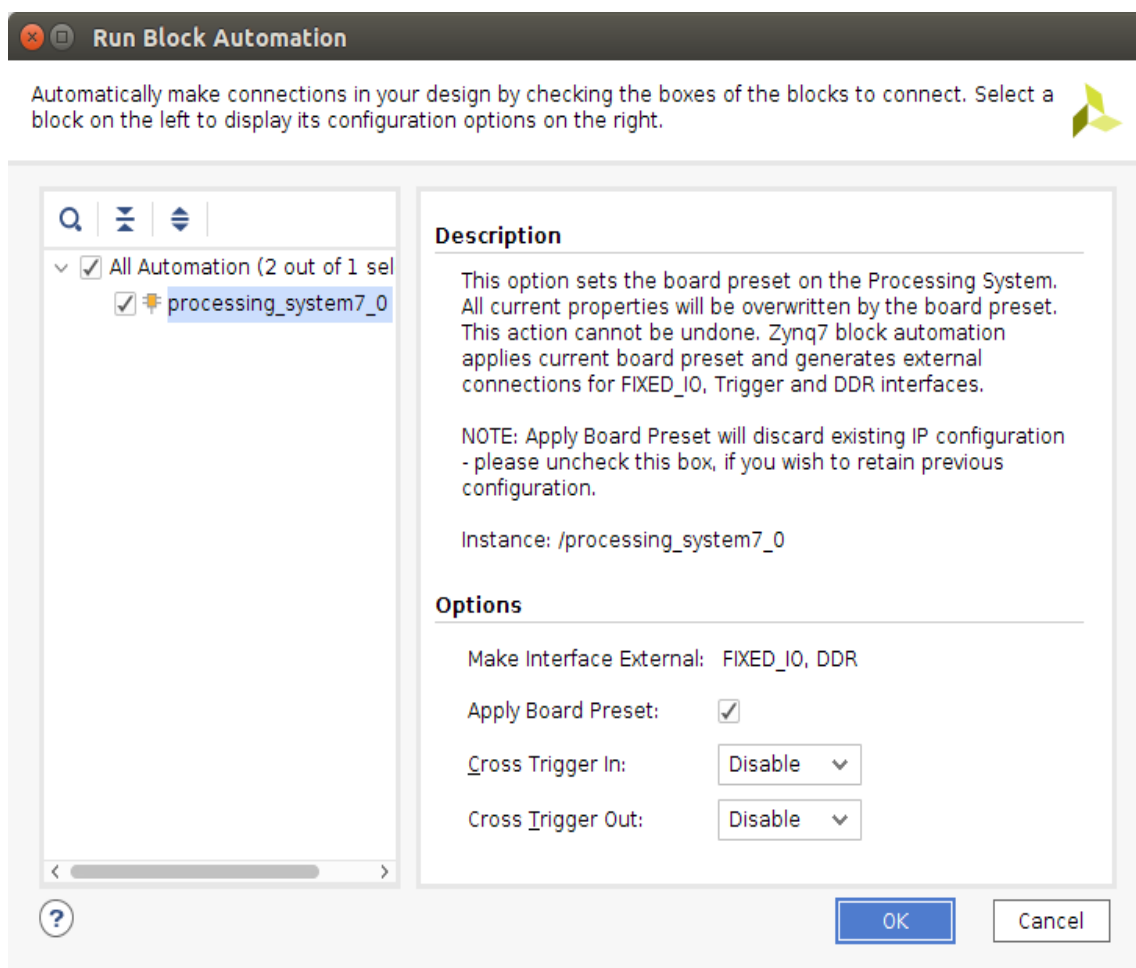
4.3. Then the **Zynq7 PS IP** block is placed in the block diagram canvas. The I/O ports shown in the Zynq block are defined by the default settings for this block.



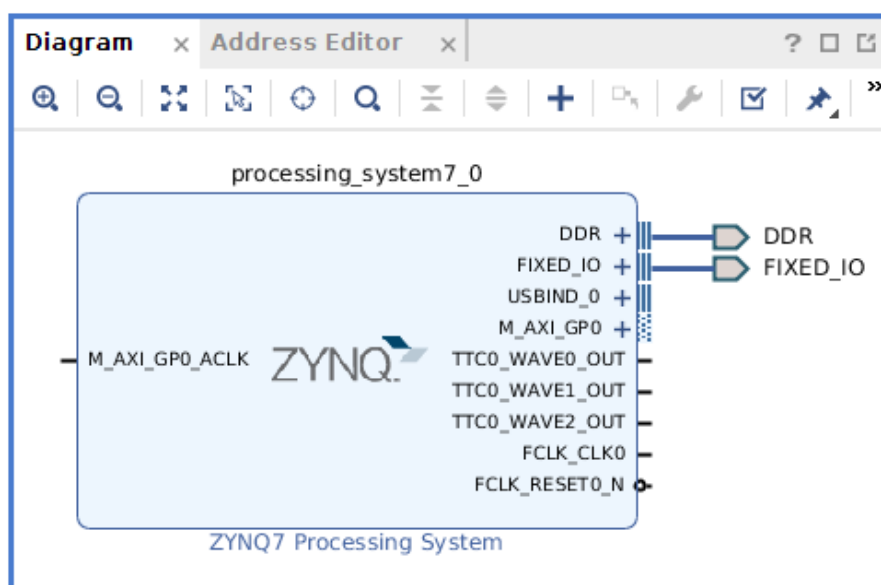
5. Click **Run Block Automation**, available in the green information bar.



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



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

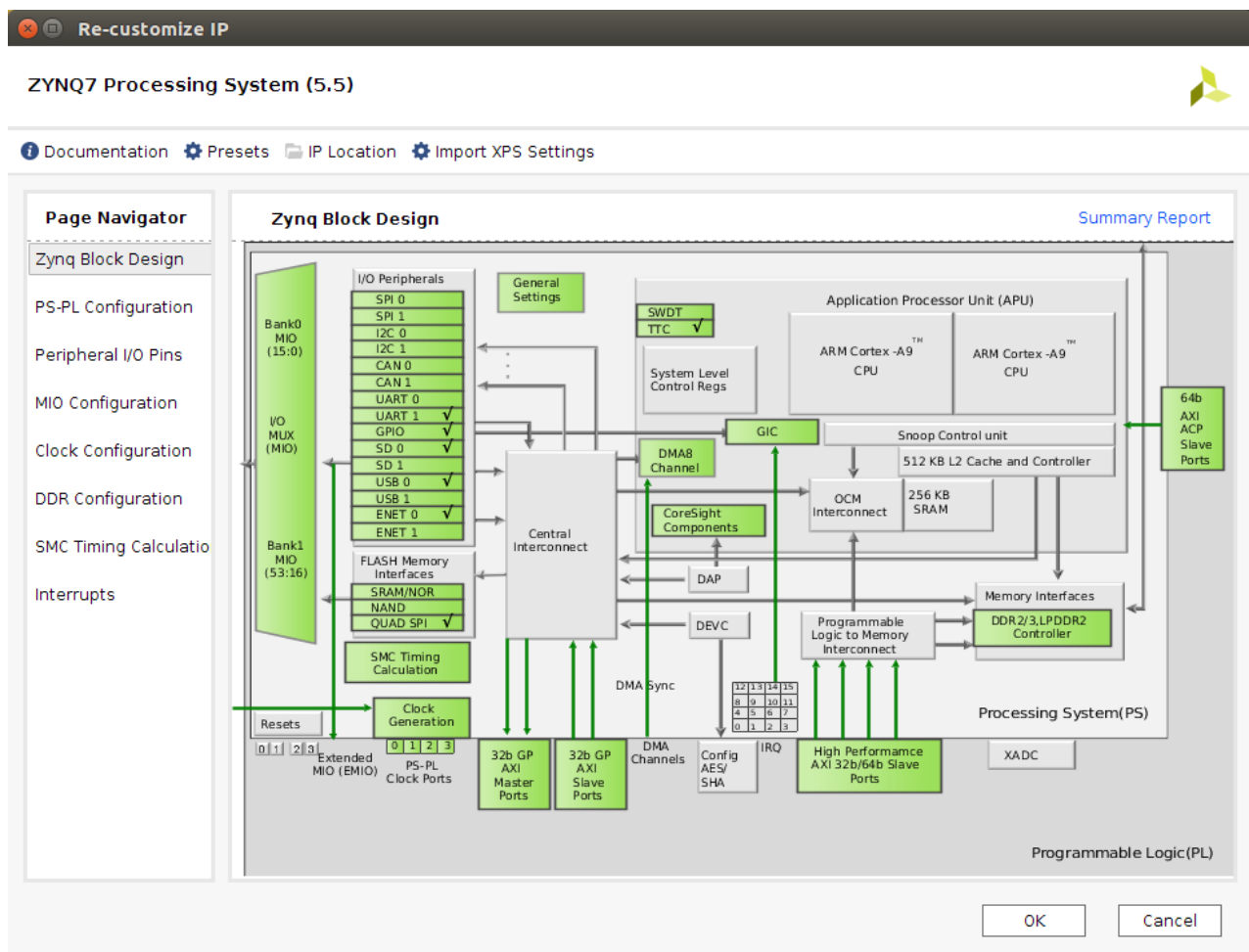


PS 7 Customization – Adding GPIO IP Cores

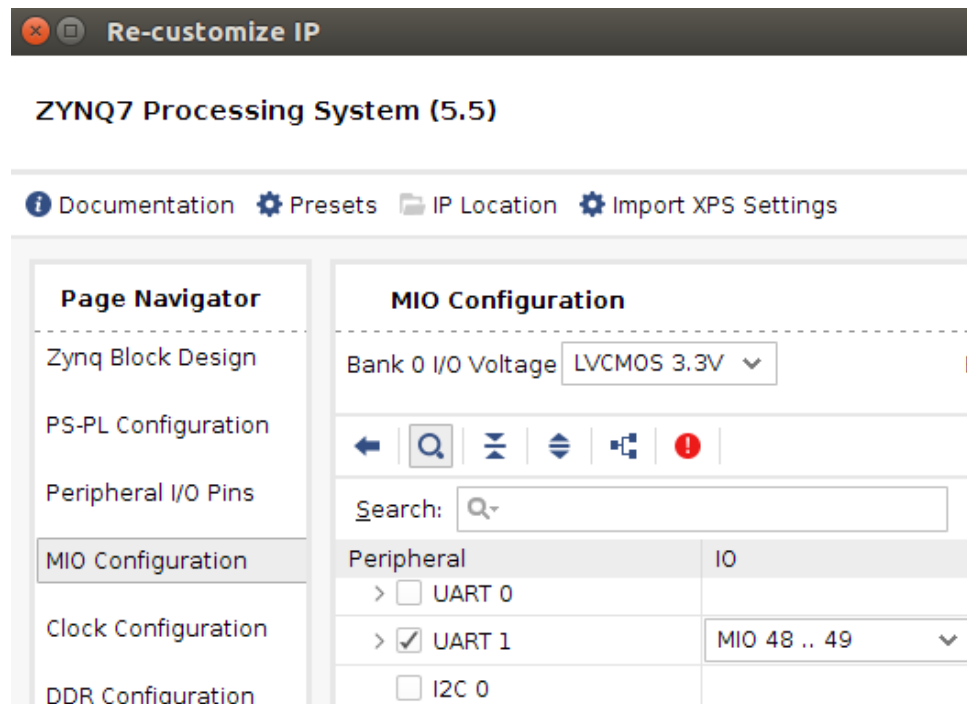
Objective Customizing the Zynq Processing System settings. For this particular lab we will enable the *AXI_M_GP0* interface, the *FCLK_RESET0_N*, and the *FCLK_CLK0* ports. We will also add two instances of a *GPIO IP Peripheral* from the IP catalog.

1. Double click in the **Zynq7 PS** block to open the **Zynq 7 Processing System Re-Customize IP** window.

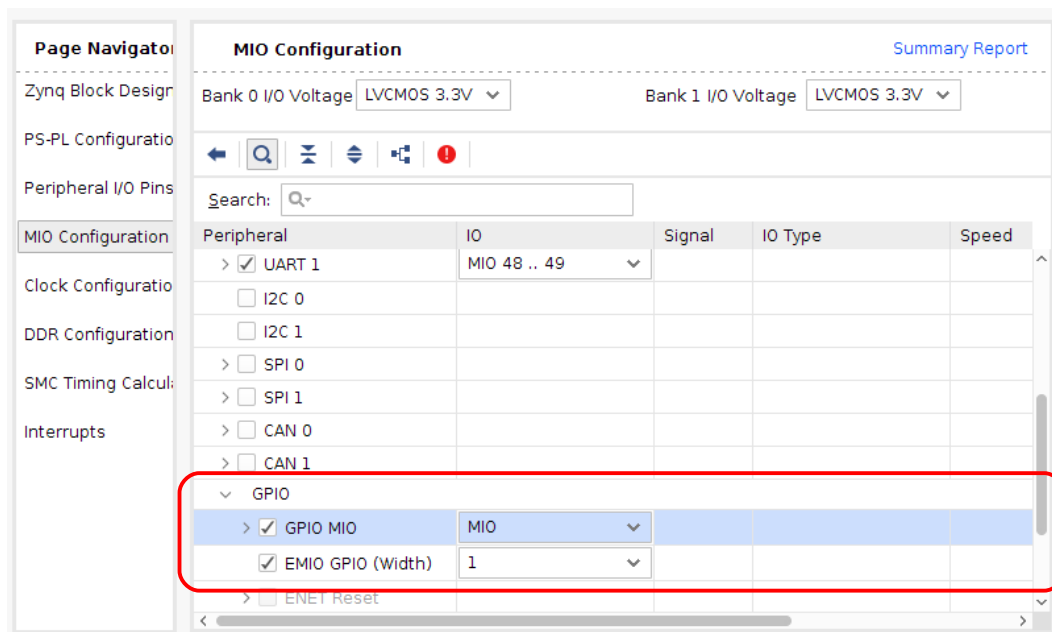
A block diagram of the **Zynq PS** should now be open, detailing the various configurable blocks of the Processing System (remember that the green block are the configurable ones).



2. Click on the **MIO Configuration** option under the **Page Navigator** pane. Expand I/O Peripherals, **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.



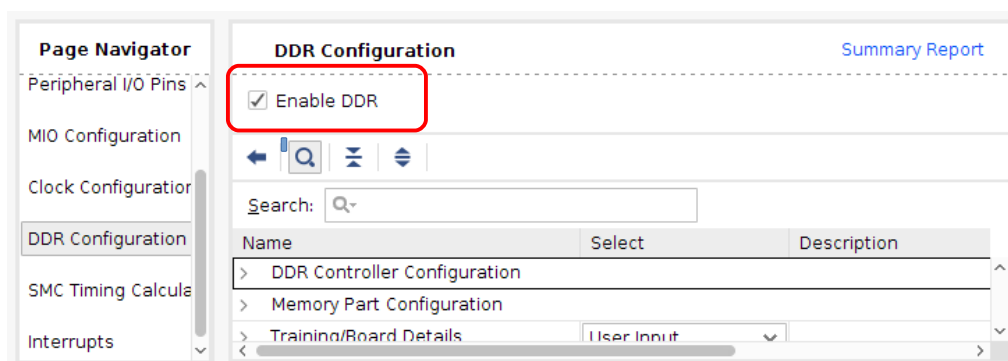
3. In the same **MIO Configuration** section, it is necessary to enable the **EMIO GPIO** to be able to route the signal coming from the PL to the PS just using the **EMIO GPIO**.
- 3.1. Find the **GPIO** peripheral and expand it.
 - 3.2. Check the **EMIO GPIO (Width)** box. Then click in the right side of the column and from the pull down menu select **1**, as the width of the 'bus' going from the PS to the PL. This is the input pin from the PL (push-button) that will go into the PS.
 - 3.3. Do check the **GPIO MIO** option, and leave the default value **MIO**. This is the output coming from the PS that will go through the **GPIO(PS)** peripheral to turn on or off the LED9.



4. 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 other cases) .

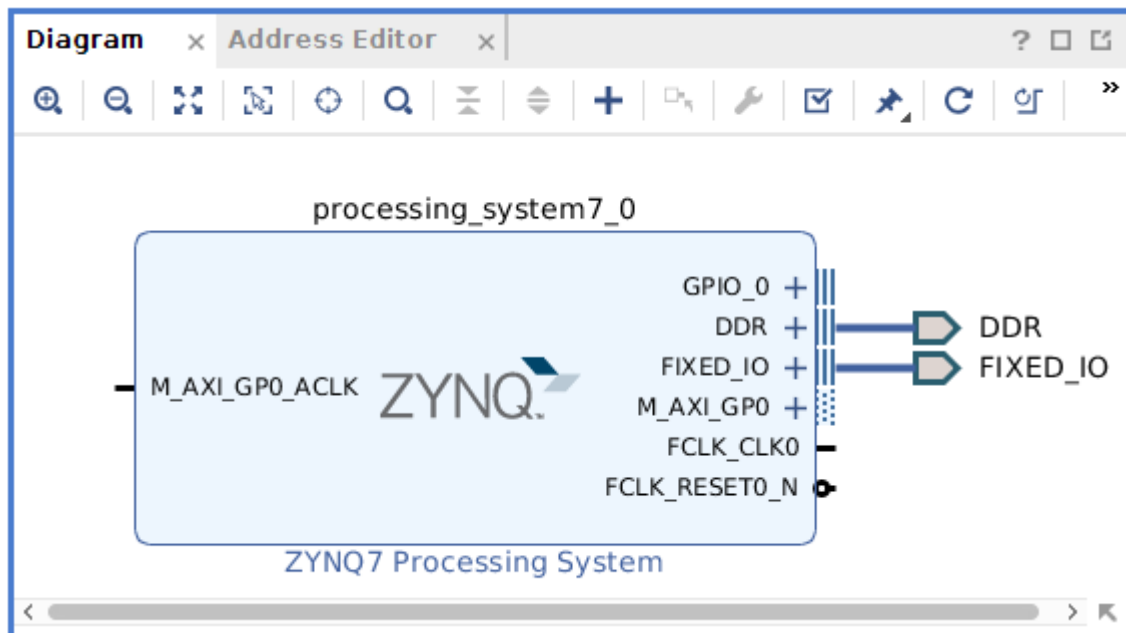


5. In the **DDR Configuration** option, be sure that the **Enable DDR** configuration is selected.



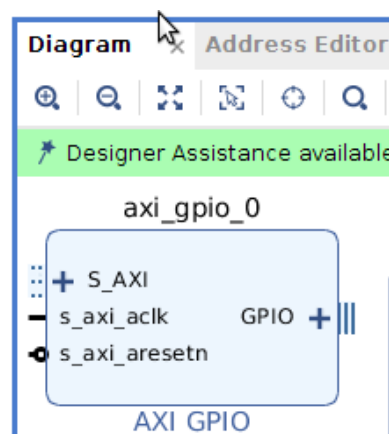
6. Finish with the **Zynq** (processing_system7_0) configuration by clicking the **OK** button in the **Re-Customize IP** window.

7. Back in the block design canvas of the project, you will notice the additional **M_AXI_GPO** interface, the **GPIO_0**, the **M_AXI_GPO_ACLK**, **FCLK_CLK0**, and **FCLK_RESET0_N** ports are now included on the **Zynq7 PS** block.

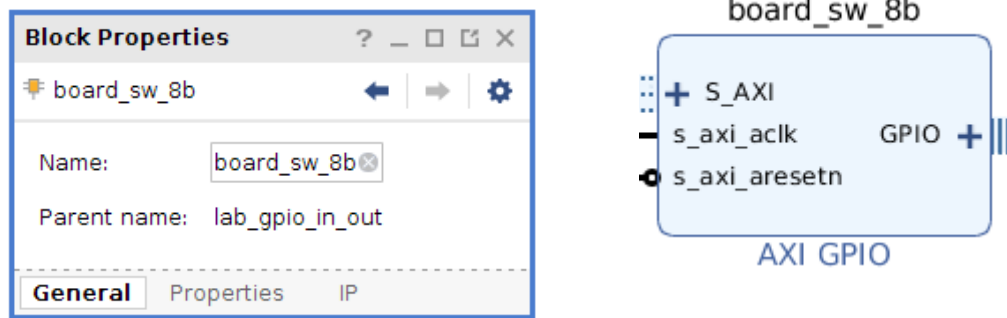


Objective Add a GPIO IP Core. Connect it to automatically to the others blocks.

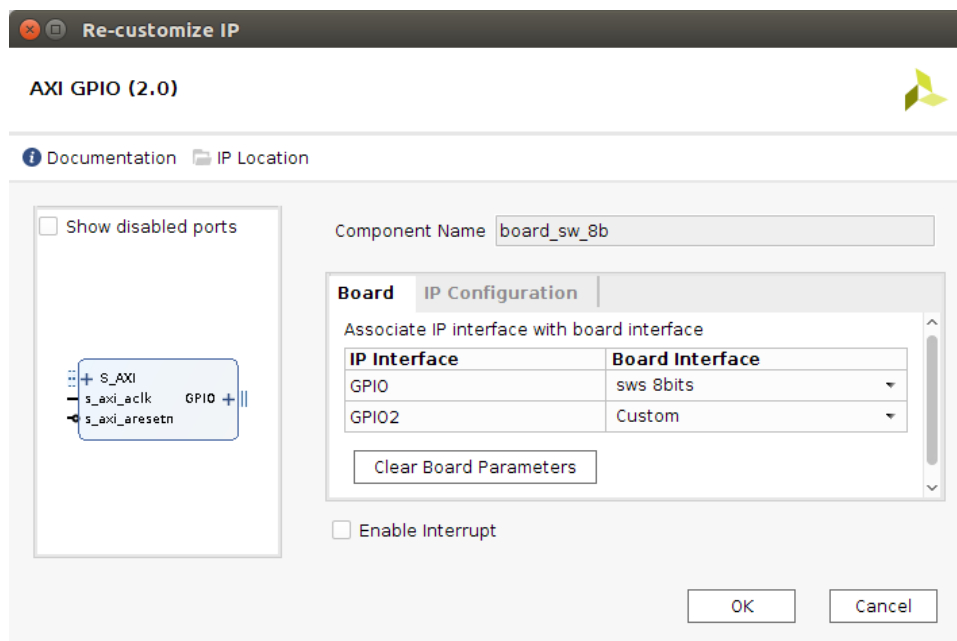
8. In the next steps, we are going to add the GPIO IP core to handle the LEDs and the switches. Click the **Add IP** icon **+** and search for **AXI GPIO** in the catalog.



9. Double-click the **AXI GPIO** to add the IP core to the design. Single click on the **AXI GPIO** block to select it, and in the **Block Properties** pane (in the middle of the Vivado main window), change the default name from **axi_gpio_0** to **board_sw_8b**.



10. Double click on the **AXI GPIO** switches block to open the customization window.
11. Select the **Board** tab and then use the pull down menu to select the **sws 8bits** (ZedBoard 8 bits switches) for GPIO.

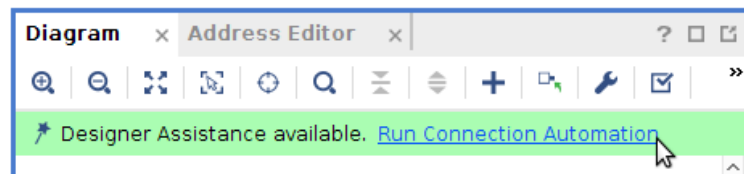


12. Click the **IP Configuration** tab. In the **GPIO Width** box, by default the 8 bits wide are showed (since it was selected the 8 bits switch interface for the **GPIO**, and Vivado is board aware due to the fact that this project was setup for the ZedBaord).

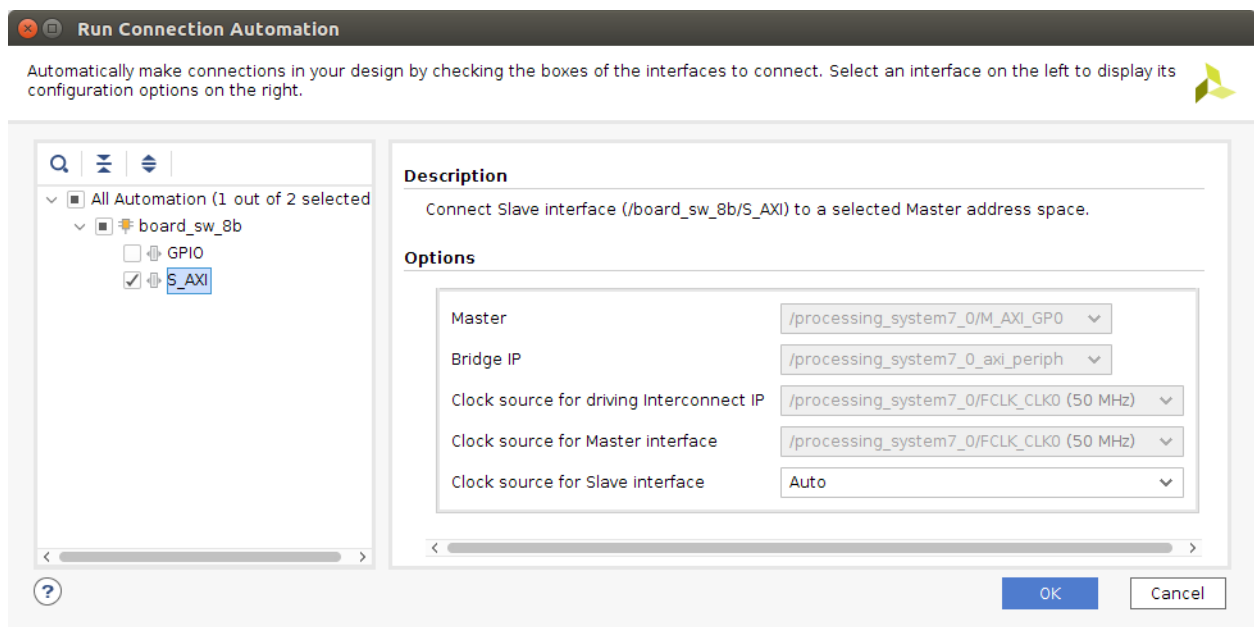
Notice that the **GPIO** peripheral can be configured for two channels, but, since in this project it is needed only one channel and without interrupt, leave the **Enable Interrupts** and **Enable Dual Channel** boxes **unchecked**.

13. Click **OK** to save and close the **GPIO** customization window, and get back to the block design pane.

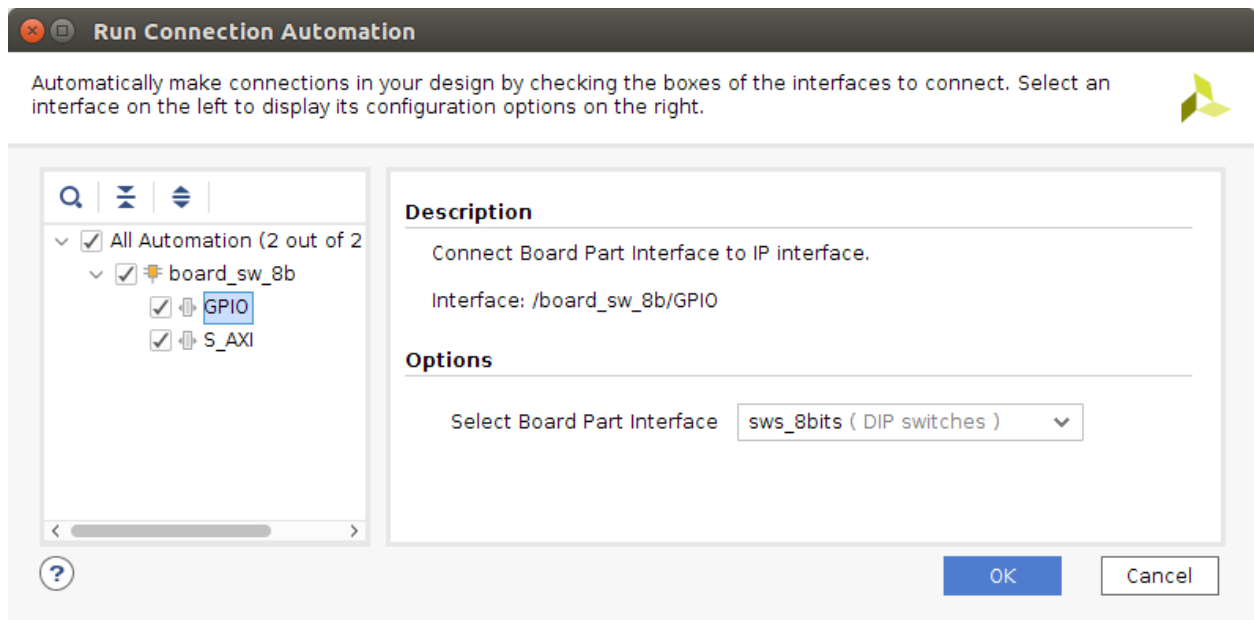
14. Notice that design assistance is now available on the green bar above the block design canvas. Click on **Run Connection Automation**.



15. First select **/board_sw_8b/S_AXI**, and left the other options unchecked. This will connect the slave part of the **GPIO IP** to the master part of the PS (actually it will be connected to the master port of the interconnect block as it will be seen soon).



16. Then, select **/board_sw_8b/GPIO** and in the pull down menu for **Select Board Part Interface**, select **sws_8bits**. This will create an output port for the design (for the Zynq) that will go to the switches on the board.



17. Click **OK** to finish this process.

18. The updated block design is now showed in the Diagram editor tab.

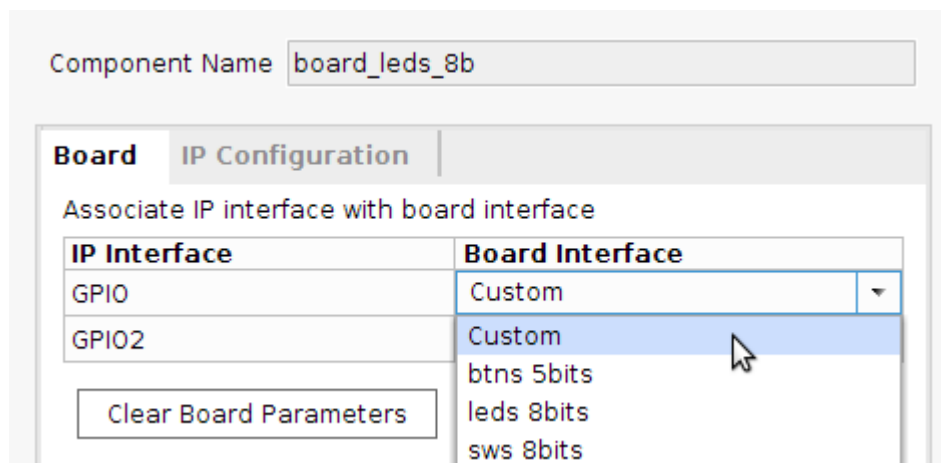
19. Click the regenerate  icon to redraw the diagram.

20. Notice two additional blocks, **Processor System Reset**, and **AXI Interconnect** have automatically been added to the design.

Objective Add another GPIO IP core and connect it manually to the other blocks.

21. Following the steps just explained and add another instance of an **AXI_GPIO** peripheral. Change the name to **board_leds_8b**.

22. Double click on the **board_leds_8b** block. Select the **Board** tab and then use the pull down menu to select the **Custom** for this GPIO.

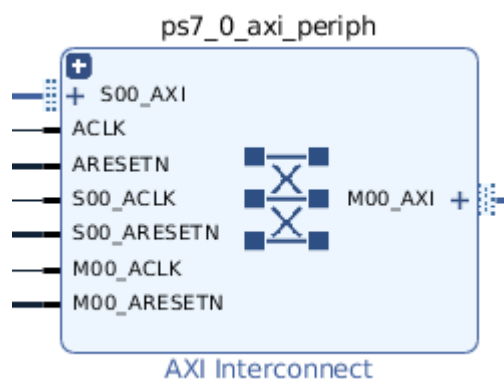


22.1. Select the **IP Configuration** tab :

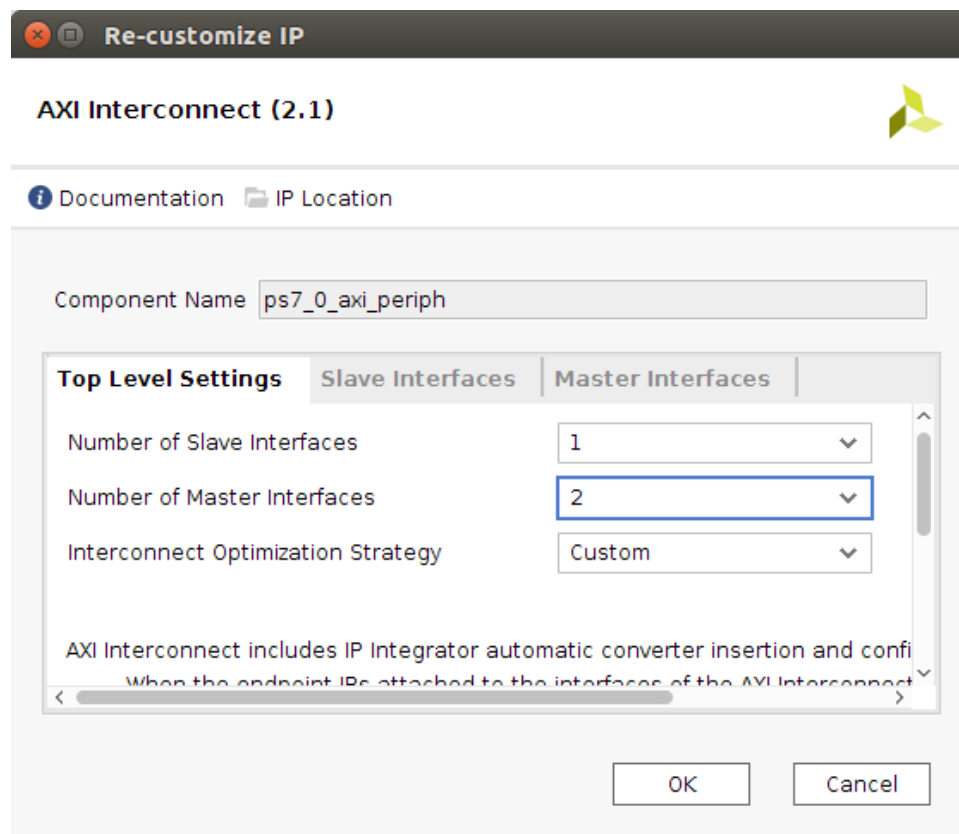
1. Set **GPIO Width** to 8 bits.
2. Check the box for **All Outputs**.
3. Then click **OK**.

23. At this point, **Run Connection Automation** could be executed (as it was done previously) to get the **AXI GPIO board_leds_8b** block connected to the PS7. The other option is to connect manually the **GPIO**. This time, in order to get familiar with this procedure as well, the **GPIO** block will be connected manually.

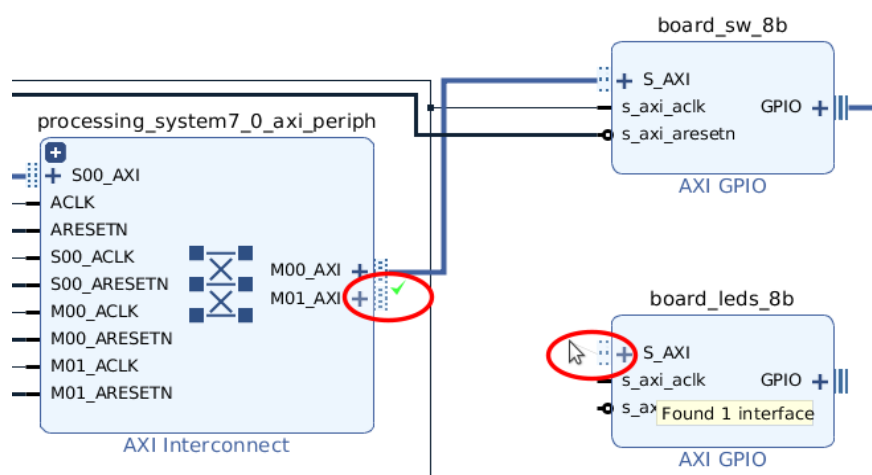
24. Double click on the **AXI Interconnect** block to open the configuration window.



25. Since we need to connect a new slave block, **board_leds_8b**, we need to add another master interface to this block. Change the **Number of Master Interfaces** from 1 to 2 and click **OK**.



26. Back into the block design, click on the **S_AXI** port (slave AXI port) of the **AXI GPIO board_leds_8b** block, and drag the pointer towards the **AXI Interconnect** block. The message **Found 1 Interface** should appear in the canvas, and a green tick should be shown beside the **M01_AXI** port on the **AXI Interconnect** block indicating this is a valid port to connect to. Drag the pointer to this port and release the mouse button to make the connection.



27. In a similar way, connect the following ports (remember to look for the green tick mark):

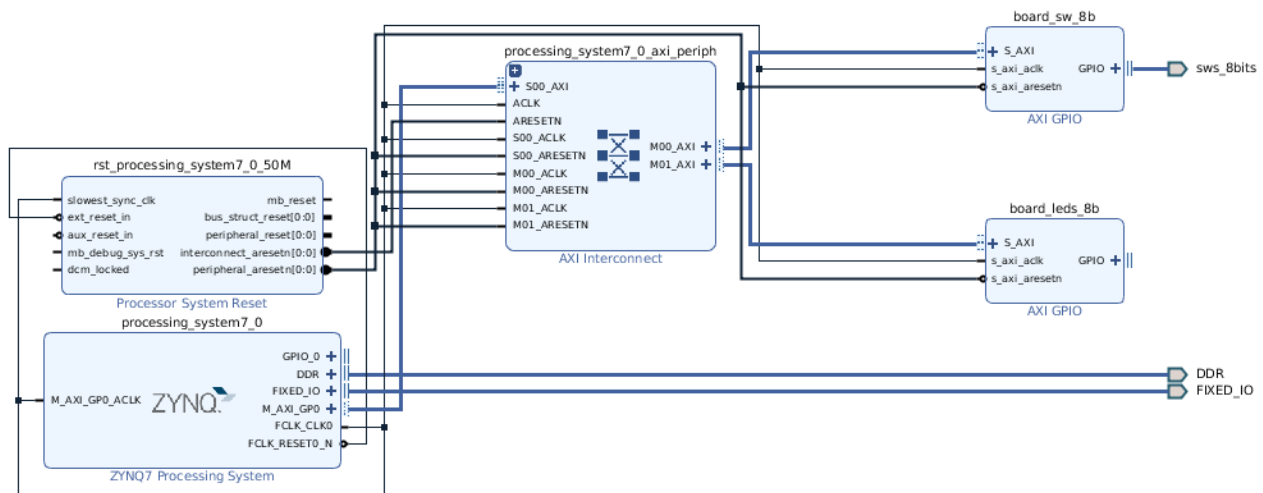
27.1. **s_axi_aclk** port -> Zynq7 Processing System **FCLK_CLK0** port

27.2. s_axi_aresetn port -> Processor System Reset peripheral_aresetn port

27.3. AXI Interconnect M01_ACLK port -> Zynq7 Processing System FCLK_CLK0 port

27.4. AXI Interconnect M01_ARESETN port -> Processor System Reset peripheral_aresetn port.


28. After finishing doing the connections, the block should look like this:

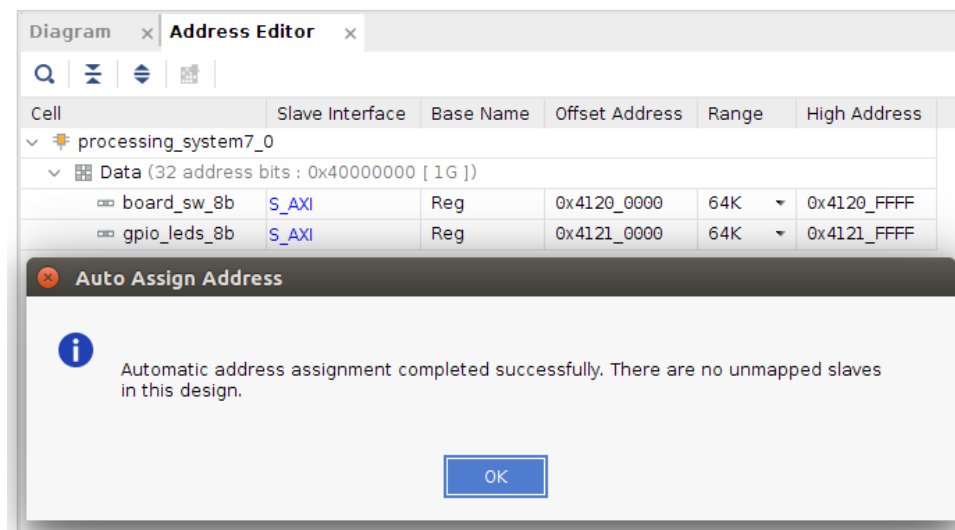


29. Click on the **Address Editor** tab, and expand **processing_system7_0 > Data > Unmapped Slaves** if necessary.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
board_sw_8b	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
Unmapped Slaves (1)					
gpio_leds_...	S_AXI	Reg			

Notice that the **switches GPIO** block has a range of memory addresses automatically assigned (0x4120_0000 - 0x4120_FFFF).

However, **gpio_leds_8b** has not. This is due to the fact that we connect this GPIO block manually. So, let's assign a range of memory addresses: right click on **board_leds_8bits** and select **Assign Address** or click on the  icon. Then, a range of addresses is automatically assigned to the **gpio_leds_8bits** GPIO block.



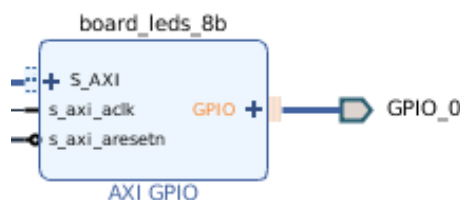
Make external the GPIO connections

Objective The GPIO IPs that we add will be connected to the corresponding pins of the Zynq.

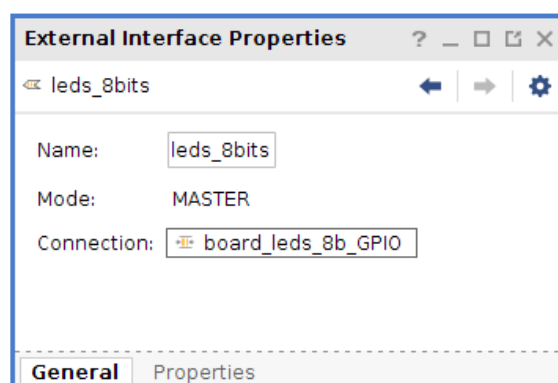
1. Click on the **Diagram** tab to return to the block diagram. Right-Click on the **GPIO** port of the **board_leds_8b** IP block.



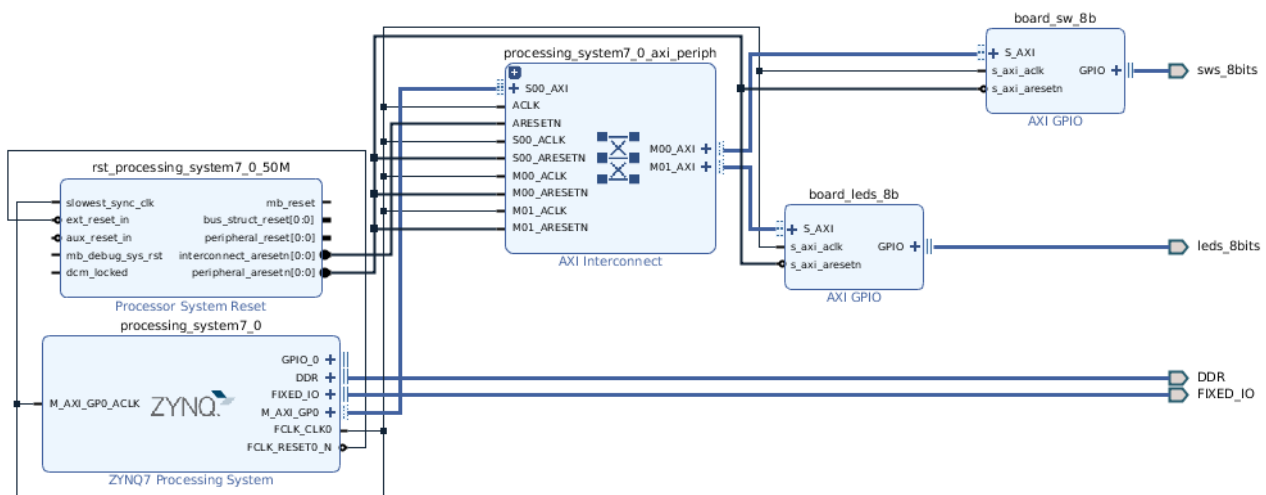
2. From the menu, select **Make External** to create the external port. This will create the external port named (by default) **GPIO_0** and will connect it to the respective peripheral (LEDs on the ZedBoard).



3. Select the **GPIO_0** port and change the name to **leds_8bits** in its **External Interface Properties** form.

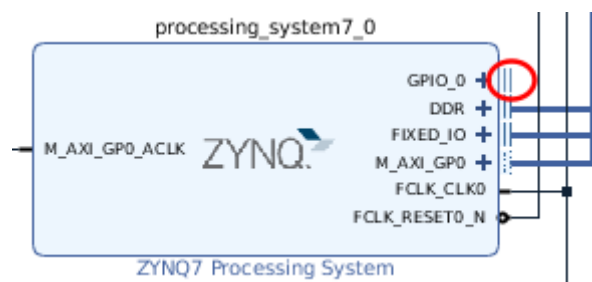


- The other GPIO IP block, the **board_sw_8b**, has already been assigned the external port previously, so there is no need to do anything with it (it was done automatically).
- Run **Design Validation (Tools -> Validate Design)** and verify there are no errors.
- The design should now look similar to the diagram below.



Objective Create an output port for the PS7 GPIO.

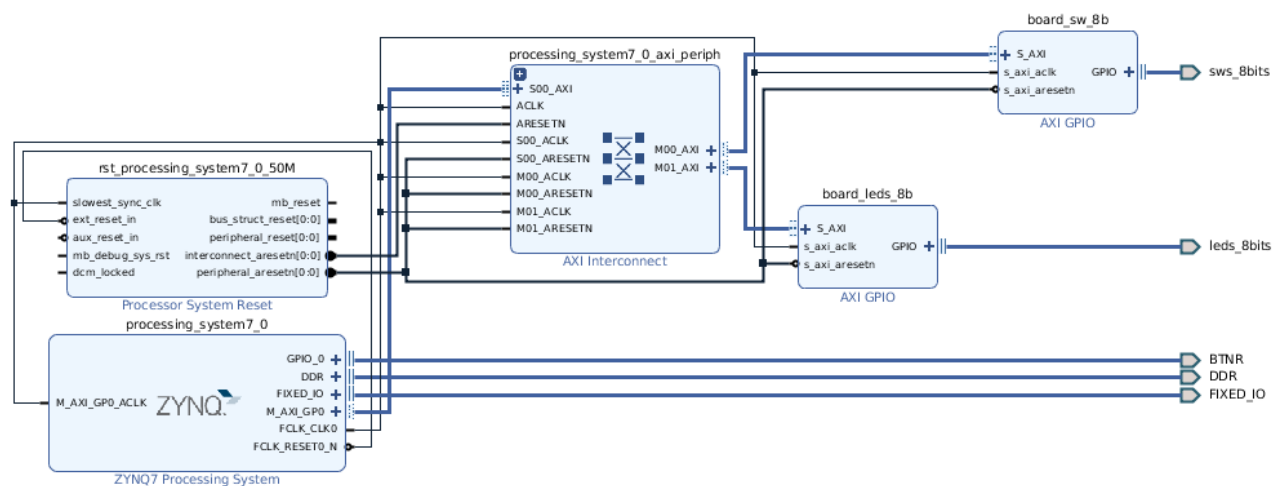
- Right-click on the **GPIO_0** pin of the **processing_system7_1** instance, and select **Make External**.



- Select the just created **GPIO_0** port, and in the **External Interface** window change the name to **BTNR**.



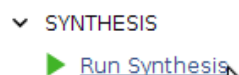
- We have completed the hardware of our system.
- Click the **Save** button to save the current block diagram, which should look like the following.



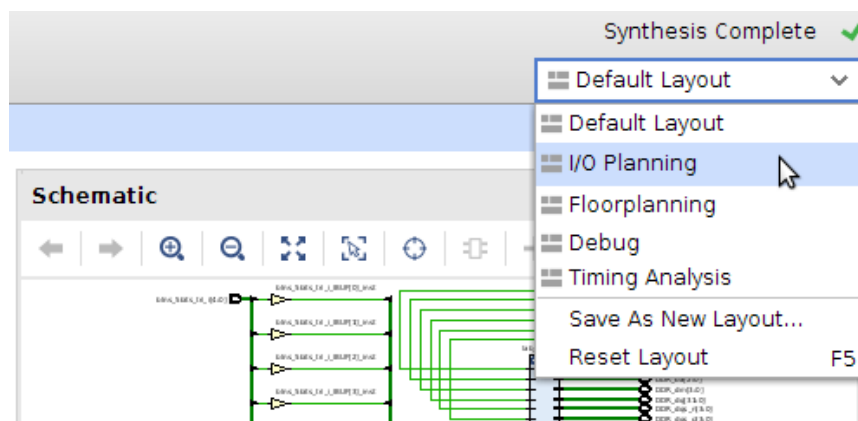
Constraint, Generate & Synthesize the design

Objective Create the HDL wrapper, generate the output product, and synthesize the design. Open the I/O Planning layout, and add the necessary I/O port constraints.

1. In the central pane, click on the **Sources** tab. The **lab_gpio_in_out.bd** (board design) file contains all the settings and configuration of the block diagram created in the block diagram editor window. Right click on **lab_gpio_in_out.bd** and select **Generate Output Products**. In the **Generate Output Products** window, leave the options as given by default, then click **Generate**. Click **OK** in the upcoming window.
2. Next, right click on **lab_gpio_in_out.bd** and select **Create HDL Wrapper** to create the top level VHDL/Verilog file from the block diagram. In the **Create HDL Wrapper** window leave the option '**Let Vivado manage wrapper and auto-update**' checked, and click **OK**.
3. Next, from the **Flow Navigator** pane, run **Synthesis**.



4. When synthesis completes, select **Open Synthesized Design** and click **OK**.
5. In the shortcut bar, select **I/O Planning** from the **Layout** drop down menu (if the option I/O Planning is not in the list, be sure that the synthesized design is open).



6. In the **I/O Ports** tab, expand the two **GPIO** icons, and expand **sws_8bits** and **leds_8bits**. Notice that for the **sws_8bits** GPIO ports, **FPGA** pin locations have automatically been assigned, as well as an **I/O Std** of **LVCMOS25** has been applied to each I/O port. These assignments are done automatically because Vivado software is board aware and is using the constraints related to the ZedBoard.

The screenshot shows the 'I/O Ports' window in Vivado. It lists two main GPIO blocks: 'sws_8bits_16670 (8)' and 'GPIO_57396 (8)'. The 'sws_8bits_16670' block is expanded, showing its 'Scalar ports (0)' which are 'sws_8bits_tri_i (8)'. This port is further expanded to show individual pins from [7] down to [0], all configured as 'IN' ports with 'LVCMOS25*' standard. The 'GPIO_57396' block is also expanded, showing its 'Scalar ports (0)' which are 'led_8bits_tri_o (8)'. This port is further expanded to show individual pins from [7] down to [0], all configured as 'OUT' ports with 'LVCMOS33*' standard. The 'Fixed' column for the 'sws_8bits' pins is checked, while for the 'leds_8bits' pins it is unchecked.

Name	Dir...	Board Part...	Boar...	Ne...	Package ...	Fixed	Bank	I/O Std
sws_8bits_16670 (8)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCMOS25*
Scalar ports (0)								
sws_8bits_tri_i (8)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCMOS25*
sws_8bits_tri_i[7]	IN	sws_8bits_...			M15	<input checked="" type="checkbox"/>	34	LVCMOS25*
sws_8bits_tri_i[6]	IN	sws_8bits_...			H17	<input checked="" type="checkbox"/>	35	LVCMOS25*
sws_8bits_tri_i[5]	IN	sws_8bits_...			H18	<input checked="" type="checkbox"/>	35	LVCMOS25*
sws_8bits_tri_i[4]	IN	sws_8bits_...			H19	<input checked="" type="checkbox"/>	35	LVCMOS25*
sws_8bits_tri_i[3]	IN	sws_8bits_...			F21	<input checked="" type="checkbox"/>	35	LVCMOS25*
sws_8bits_tri_i[2]	IN	sws_8bits_...			H22	<input checked="" type="checkbox"/>	35	LVCMOS25*
sws_8bits_tri_i[1]	IN	sws_8bits_...			G22	<input checked="" type="checkbox"/>	35	LVCMOS25*
sws_8bits_tri_i[0]	IN	sws_8bits_...			F22	<input checked="" type="checkbox"/>	35	LVCMOS25*
GPIO_57396 (8)	OUT					<input type="checkbox"/>		LVCMOS33*
Scalar ports (0)								
led_8bits_tri_o (8)	OUT					<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[7]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[6]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[5]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[4]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[3]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[2]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[1]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*
led_8bits_tri_o[0]	OUT	leds_8bits...				<input type="checkbox"/>		LVCMOS33*

However, the I/O pins for the **leds_8bits** GPIO block have not been assigned, therefore we do need to do that manually. Check the **ZedBoard User Guide** to find the FPGA I/O pins assigned to the available LEDs on the board. You will find them in Table 14, page 21.

2.7.3 User LEDs

The ZedBoard has eight user LEDs, LD0 – LD7. A logic high from the Zynq-7000 AP SoC I/O causes the LED to turn on. LED's are sourced from 3.3V banks through 390Ω resistors.

Table 14 - LED Connections









Signal Name	Subsection	Zynq pin
LD0	PL	T22
LD1	PL	T21
LD2	PL	U22
LD3	PL	U21
LD4	PL	V22
LD5	PL	W22
LD6	PL	U19
LD7	PL	U14
LD9	PS	D5 (MIO7)

Use **Table 14** to complete the I/O pin assignment for the **LEDs** GPIO. Also, change the default voltage to LVCMOS33.

The I/O Ports should look similar to the following:

▼ GPIO_57396 (8)	OUT					✓	33	LVCMOS33*
Scalar ports (0)								
▼ led_8bits_tri_o (8)	OUT					✓	33	LVCMOS33*
led_8bits_tri_o[7]	OUT	leds_8bits...			U14 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[6]	OUT	leds_8bits...			U19 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[5]	OUT	leds_8bits...			W22 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[4]	OUT	leds_8bits...			V22 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[3]	OUT	leds_8bits...			U21 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[2]	OUT	leds_8bits...			U22 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[1]	OUT	leds_8bits...			T21 ▼	✓	33	LVCMOS33*
led_8bits_tri_o[0]	OUT	leds_8bits...			T22 ▼	✓	33	LVCMOS33*

7. We also need to assign the pin number to the **BTNR** input port. Search for the **BTNR** port in the I/O Ports tab.

Tcl Console	Messages	Log	Reports	Design Runs	Package Pins	I/O Ports	x	
<div><div><div><div><div></div></div><div><div></div></div><div><div></div></div><div><div></div></div><div><div></div></div><div><div></div></div></div><div><div></div></div><div><div></div></div><div><div></div></div><div><div></div></div><div><div></div></div></div></div>								
Name	Dire...	Board Part...	Boar...	Ne...	Package ...	Fixed	Bank	I/O Std
>  FIXED_IO_16670 (59)	IN...					<input checked="" type="checkbox"/>	(Multiple)	(Multiple)*
▼  BTNR_16670 (1)	IN...					<input type="checkbox"/>		default (LVC...
 Scalar ports (0)								
>  BTNR_tri_io (1)	IN...					<input type="checkbox"/>		default (LVC...
▼  sws_8bits_16670 (8)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCMOS25*
 Scalar ports (0)								
▼  sws_8bits_tri_i (8)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCMOS25*
 sws_8bits_tri_i[7]	IN	sws_8bits_...			M15 ▼	<input checked="" type="checkbox"/>	34	LVCMOS25*

As you can see there is no pin assignment done for this port.

8. Again, check the **ZedBoard User Guide** to find the FPGA I/O pins assigned to the available buttons on the board, in particular the **BTNR** button. You will find them in Table 12, page 2.

2.7.1 User Push Buttons

The ZedBoard provides 7 user GPIO push buttons to the Zynq-7000 AP SoC; five on the PL-side and two on the PS-side.

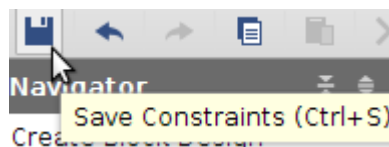
Pull-downs provide a known default state, pushing each button connects to Vcco.

Table 12 - Push Button Connections

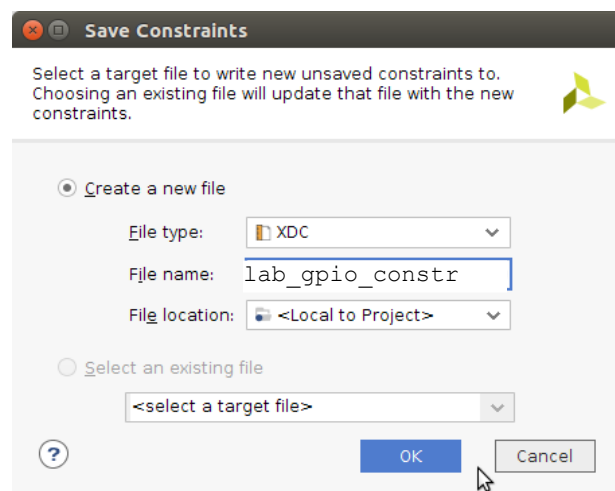
Signal Name	Subsection	Zynq pin
BTNU	PL	T18
BTNR	PL	R18
BTND	PL	R16
BTNC	PL	P16
BTNL	PL	N15
PB1	PS	D13 (MIO 50)
PB2	PS	C10 (MIO 51)

9. Assign the **R18** pin to the **BTNR** port, and set the **I/O Std** to **LVCMOS25**.

10. Click on the save icon to save the I/O pin assignment.



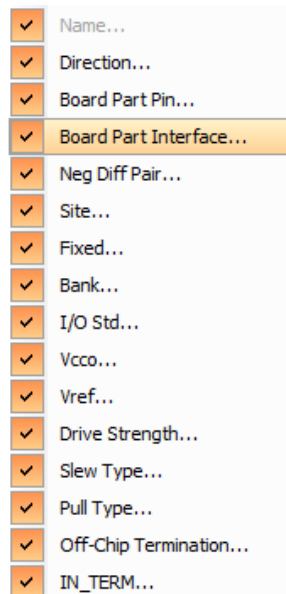
11. A dialog window will come up asking the constraint file name, use **lab_gpio_constr.xdc**.



Note 1: The LVCMOS25 standard is set for these particular IOs, due to the fact that both the push buttons and the switches are buy default connected to 2.5V.

Note 2: The I/O Ports table presents a lot of information about each IO pin in different columns. It is possible to enable or disable each column by checking or unchecking the needed information from the options column that is obtained by

click right mouse button *on any of the names of the first row (Name, Direction, Bank, etc.)*.

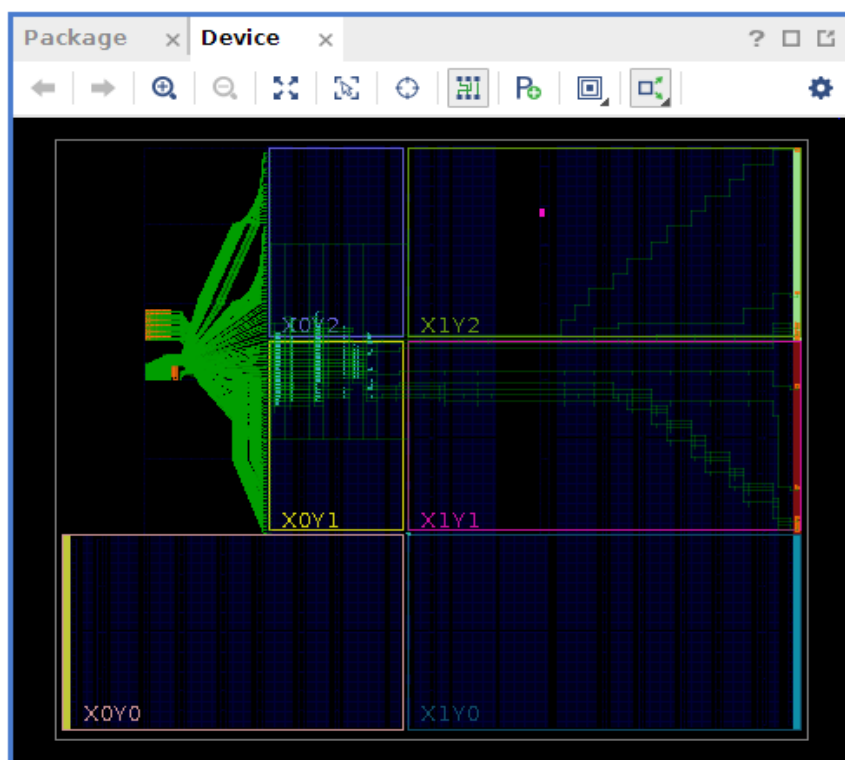


12. In the Flow Navigator pane, click in Run Implementation.



It could happens that a message window come up stating that “Synthesis is out of Data’. This is due to the fact that since we have update the I/O constraint the synthesis should be run again to get a better and optimized resultant netlist.

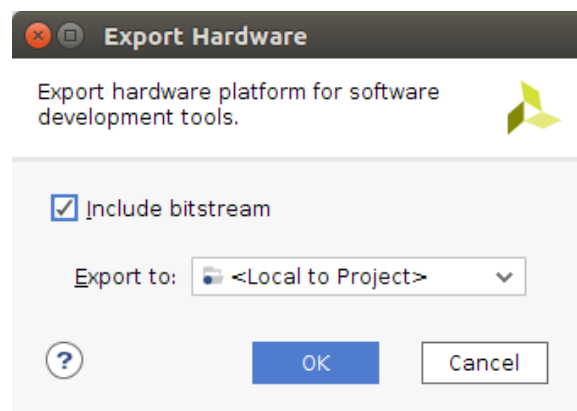
13. Once completed the implementation task, click on the option **Open Implemented Design**. Then, click in the **device** tab. You will get a figure detailing the place and route of the current design.



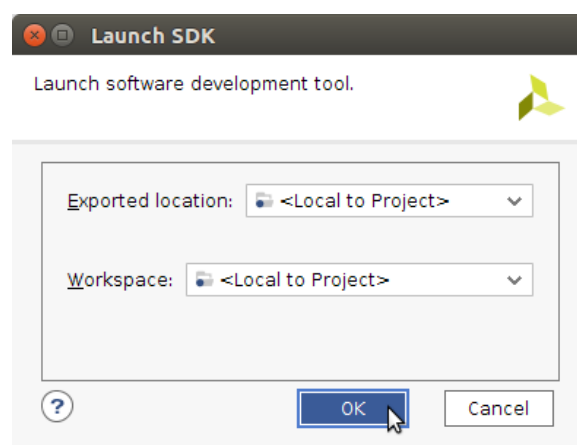
Generate Bitstream and Export Hardware

Objective Generate the bistream, and export the hardware along with the generated bitstream to SDK.

1. In the **Flow Navigator** menu, click on **Generate Bitstream**. Click **OK** after finishing the bitstream generation (leave the '**View Report**' option checked).
2. Export the hardware to the **SDK** environment. Do **File -> Export -> Export Hardware**. Since there is some logic in the PL part of the Zynq, the respective bitstream has to be included in the export task. Hence, do check the **Include bitstream** box. Click **OK**.



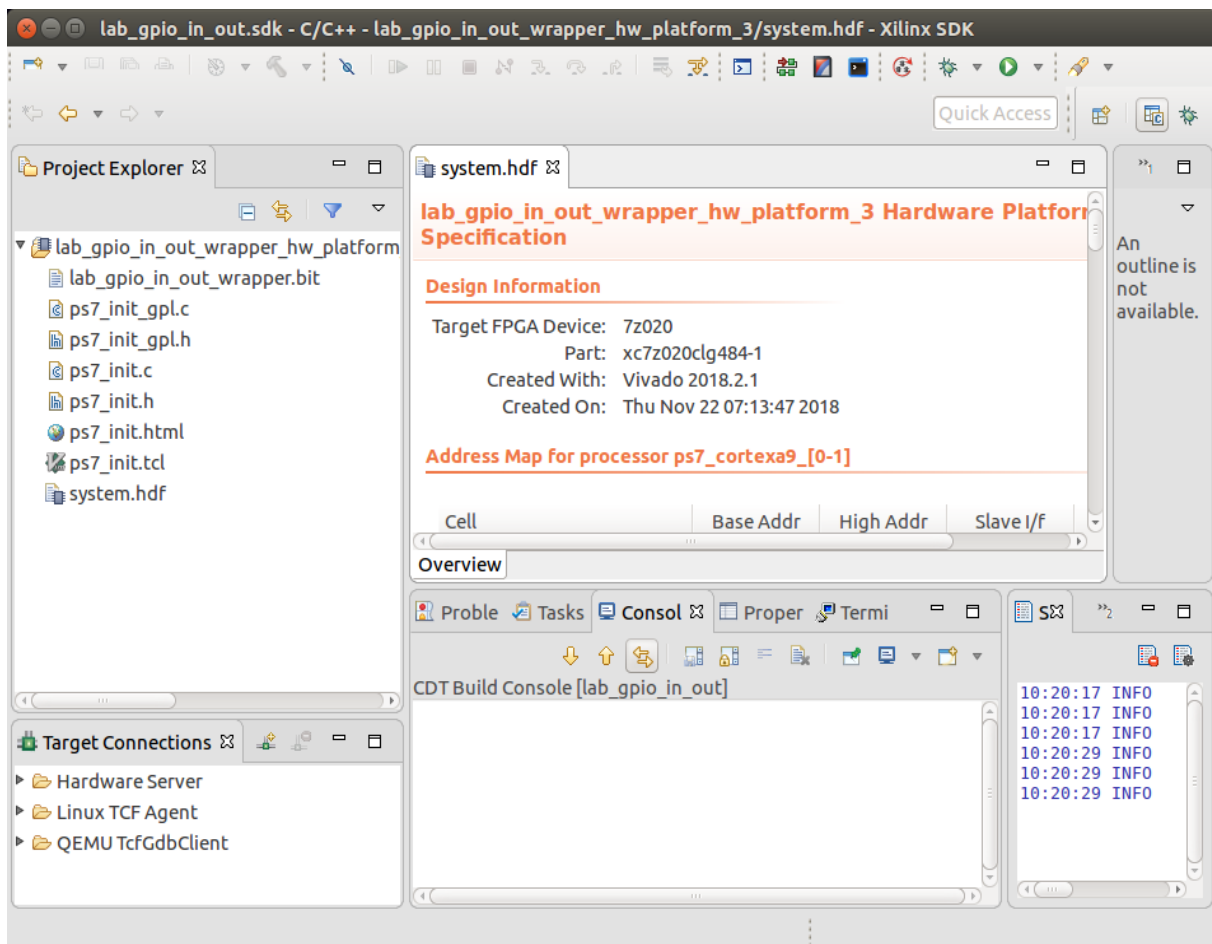
3. Launch SDK by doing **File -> Launch SDK** and click **OK**.



Generate 'C' Application in SDK

Objective Generate the project in the SDK environment, and the 'C' application code to read from the switches and write to the LEDs.

1. In SDK environment, the hardware platform related files (generated by Vivado Design Suite) are imported and showed in the folder **lab_gpio_in_out_wrapper_hw_platform_0**. Note: if there is any previous project open in the Project Explorer pane, close it by right clicking on the project name and select Close Project.



2. The **system.hdf** file holds all the hardware information generated by Vivado Design Suite. For instance, we can see the memory map and the IP Cores used. Browse the file until you find the assigned memory map for **switches** and **LEDs**, also browse it to find the GPIO IP cores used for **switches** and **LEDs**.

3. From the **File** menu select **File -> New -> Application Project**. Use as **Project Name: lab_gpio_inout_sdk**. Leave the other settings in its default value/name.

New Project

Application Project
Create a managed make application project.

Project name:

☒ Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Hardware Platform:

Processor:

Target Software

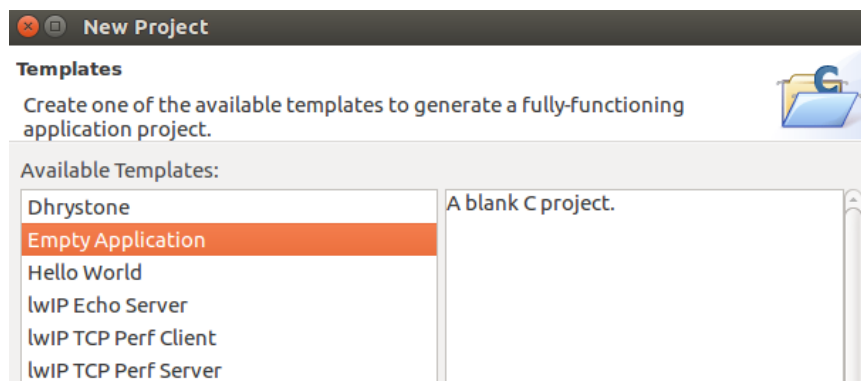
Language: ☒ C ☐ C++

Compiler:

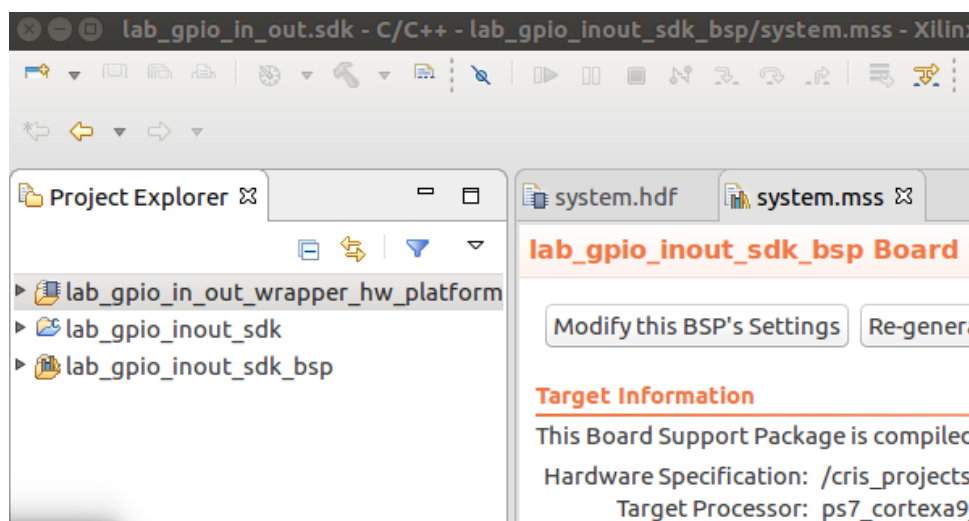
Hypervisor Guest:

Board Support Package: ☒ Create New ☐ Use existing

4. Click **Next**, to open the **Templates** settings window. Select **Empty Application** and click **Finish**. This will create a new application project.



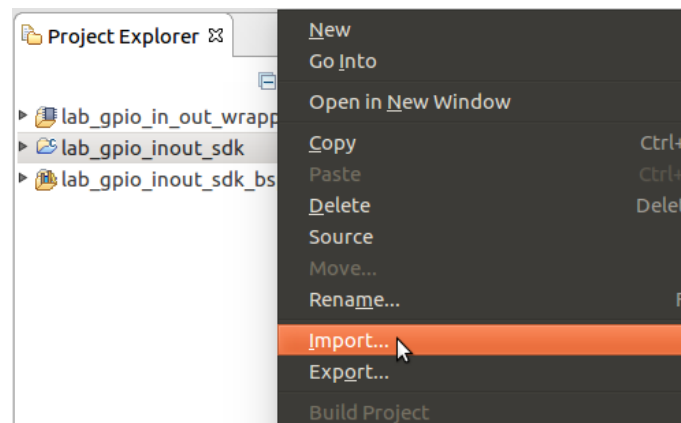
5. The SDK will create the project itself as well as the BSP.



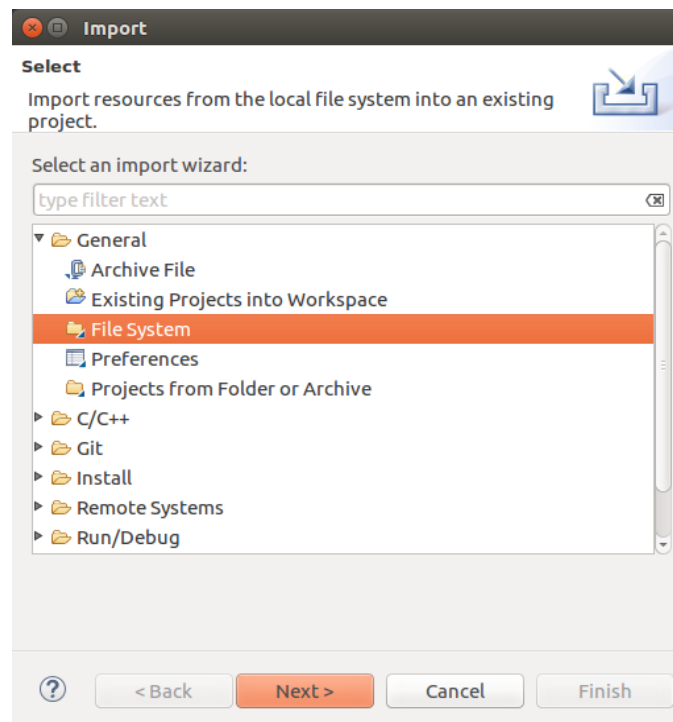
6. One important file that is generated, among others, is the **xparameters.h**, which include all the board related definitions. The **xparameters.h** file is in the folder **lab_gpio_inout_sdk_bsp→ps7_cortexa9_0→include**. This information will be very useful in the next steps and future labs.

Browse and try to find the file.

7. In the **Project Explorer** view, expand **lab_gpio_inout_sdk**, and right-click on the **src** folder, and select **Import**.

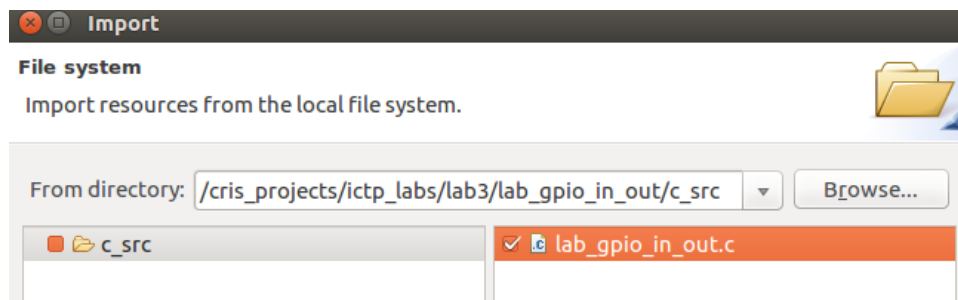


8. Expand **General** category and double-click on **File System**.



9. Browse to the [c:/../labs/lab_gpio_inout/c_src](#) folder.

10. Select the **lab_gpio_inout.c** file, to be imported into the application project and click **Finish**.



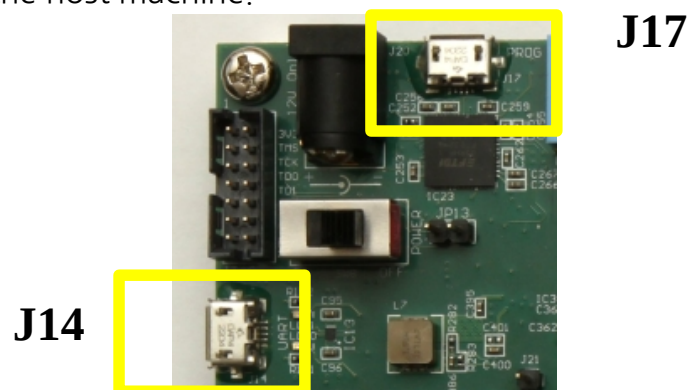
11. Open to edit the **lab_gpio_inout.c** file, by double clicking on the file name. As you can see there are some missing functions in the code represented by '???' characters. Based on what you have learned in the theoretical classes, replace the '???' characters with the 'C' function(s) to be able to execute correctly the **GPIO** read from the switches and the **GPIO** writes to the LEDs.

```
37  int main (void)
38  {
39
40      XGpio sw, led;
41      int i, pshb_check, sw_check;
42      static XGpio GPIOInstance Ptr;
43      XGpioPs_Config*GpioConfigPtr;
44      int xStatus;
45      int iPinNumberEMIO = 54;
46      u32 uPinDirectionEMIO = 0x0;
47      u32 uPinDirection = 0x1;
48
49      xil_printf("-- Start of the Program --\r\n");
50
51      // AXI GPIO switches Intialization
52      ???(&sw, XPAR_BOARD_SW_8B_DEVICE_ID);
53
54      // AXI GPIO leds Intialization
55      ???(&???, XPAR_BOARD_LEDS_8B_DEVICE_ID);
56
```

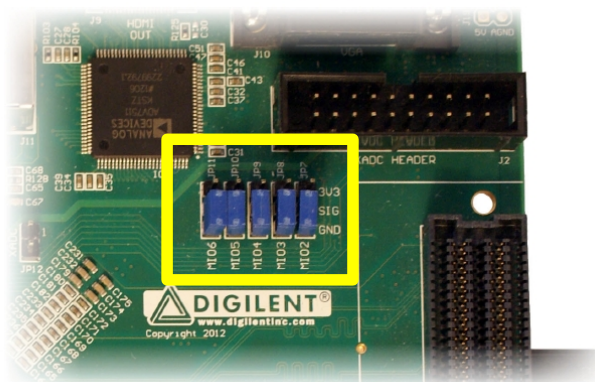
Download the design into the Zynq SoC

Objective Connect the ZedBoard with the two micro-usb cables and power it ON. Establish the serial communication using a serial utility software.

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

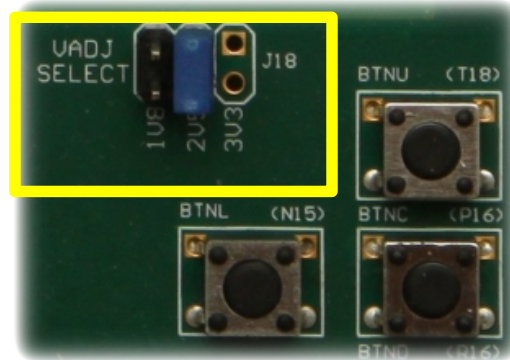


IMPORTANT 1: Ensure that jumpers **JP7** to **JP11** are set as shown in the figure below for the JTAG configuration mode.



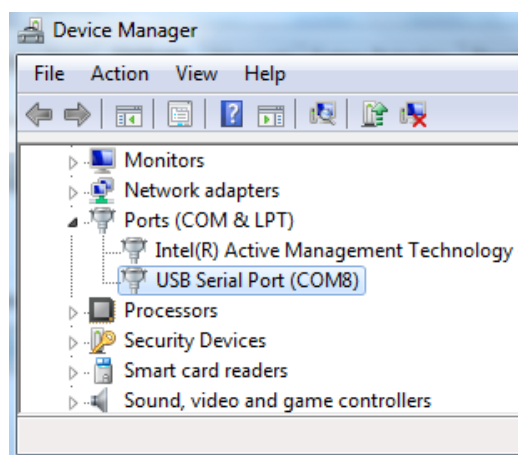
IMPORTANT 2: be sure to already have installed the Cypress device driver for the USB-UART chip on the ZedBoard.

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.
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.



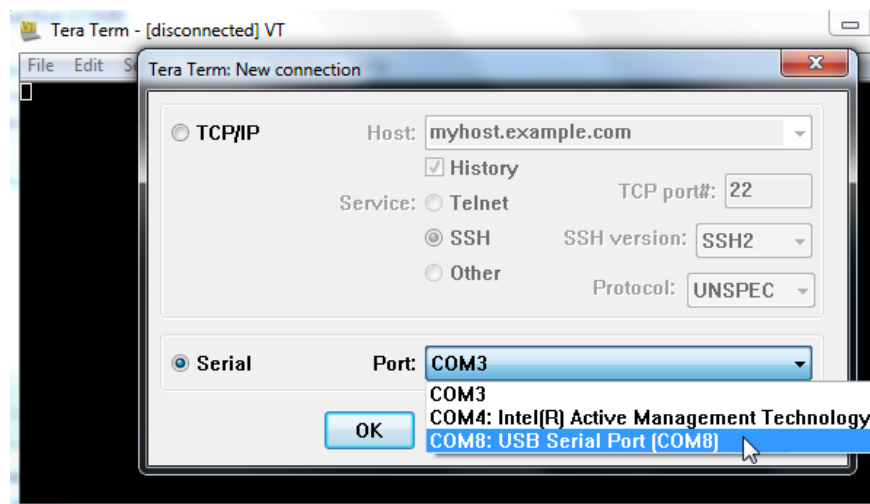
7. Setting up the serial utility software:

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.

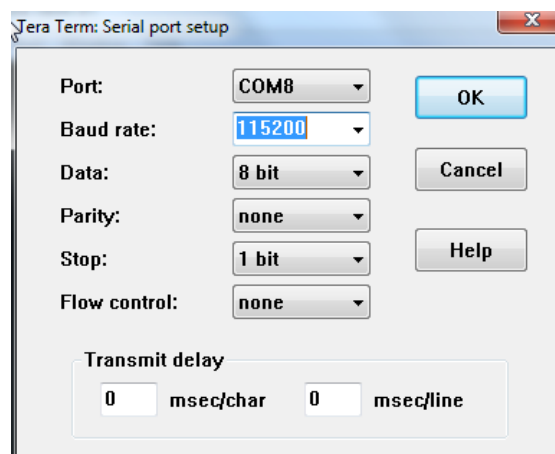
For the Tera Term configuration, first select Serial as communication protocol, and from the pull down menu select the Port to be used.


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

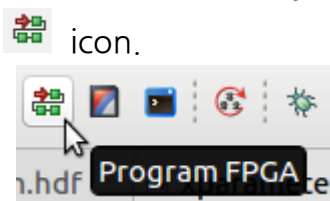
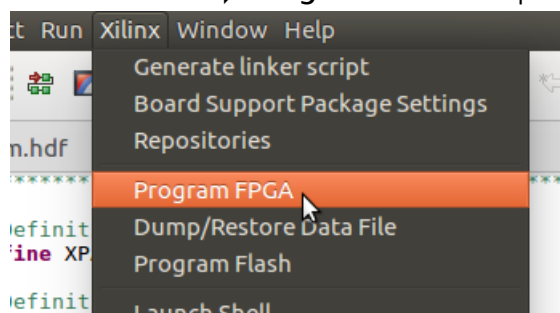
system, e.g. in Windows use the Device Manager utility. In case that no serial port is detected, re-install the Cypress device driver for the USB-UART chip on the ZedBoard.



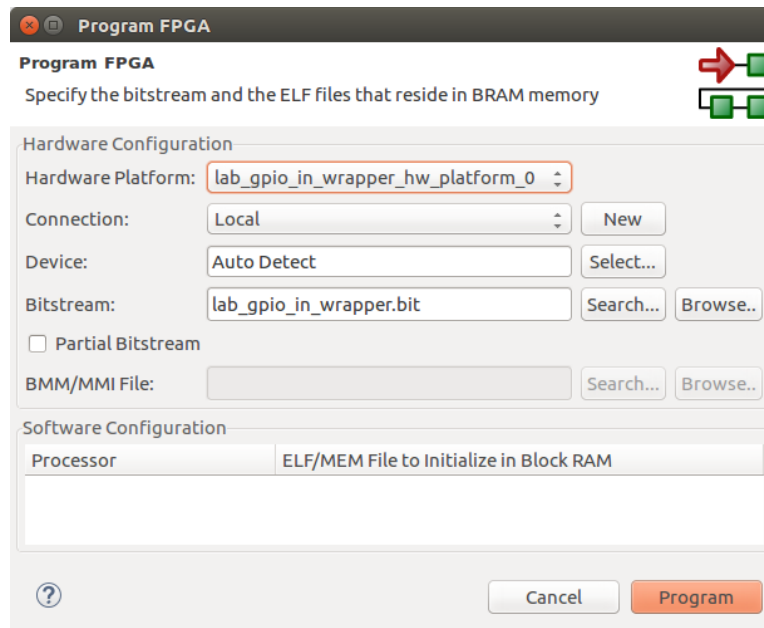
7.2. Do **Setup** → **Serial Port** and configure the serial port of the Tera Term or the software you are using, with the values detailed in the following figure.




8. Since in this project there is logic implemented in the PL part of the Zynq device it is needed to configure the PL. To do so, from the SDK environment you can either select **Xilinx Tools** → **Program FPGA** or press the  icon.



9. Click the **Program** icon to download the hardware bitstream to the PL part (FPGA).



10. When **PL** is successfully programmed, the **ZedBoard DONE LED** (blue LED) will light.
11. Once the **PL** part is programmed, the next step is to execute the 'C' code in the PS part of the Zynq. Select **lab_gpio_inout_sdk** in the **Project Explorer** pane, right-click and select **Run As -> Launch on Hardware (GDB)** to download the application, execute **ps7_init**, and execute **lab_gpio_inout_sdk.elf**.
12. Play with the switches, the LEDs should follow any change.
13. Next, press **BTNR**, the **LED9** should light.
14. Select **Console** tab and click on the **Terminate** button () to stop the program.
15. Close **SDK** and **Vivado** programs by selecting **File -> Exit** in each program.
16. **Power OFF** the board.

CONGRATULATIONS YOU HAVE THE COMPLETE LAB !!!

Conclusion

GPIO peripherals were added from the Vivado IP catalog and connected to the Processing System through the 32bits Master GP0 interface. The peripherals were configured and external FPGA connections were established. Also, a **PS** output pin, that control an LED, was connected with a push-button associated with a **PL** input pin.

A 'C' application project was created in the SDK environment and the functionality was verified after downloading the bitstream into the ZedBaord and executing the program.