

---

# *LABORATORY 4*

## *Zynq Design*

---

*Adding GPIO IP Cores in PL*

*GPIO IN (Switches) – GPIO OUT (LEDs)*

# *Building a Complete Embedded System*

## Introduction

This lab guides you through the process of using *Vivado* and *IP Integrator* to create a complete ARM Cortex-A9 based processor system targeting the ZedBoard Board. You will use the *Block Design* feature of the *IP Integrator* to add and configure the *PS7* and *IP Cores* to create the hardware system and SDK to create an application to verify the design functionality.

## Objectives

After completing this lab, you will be able to:

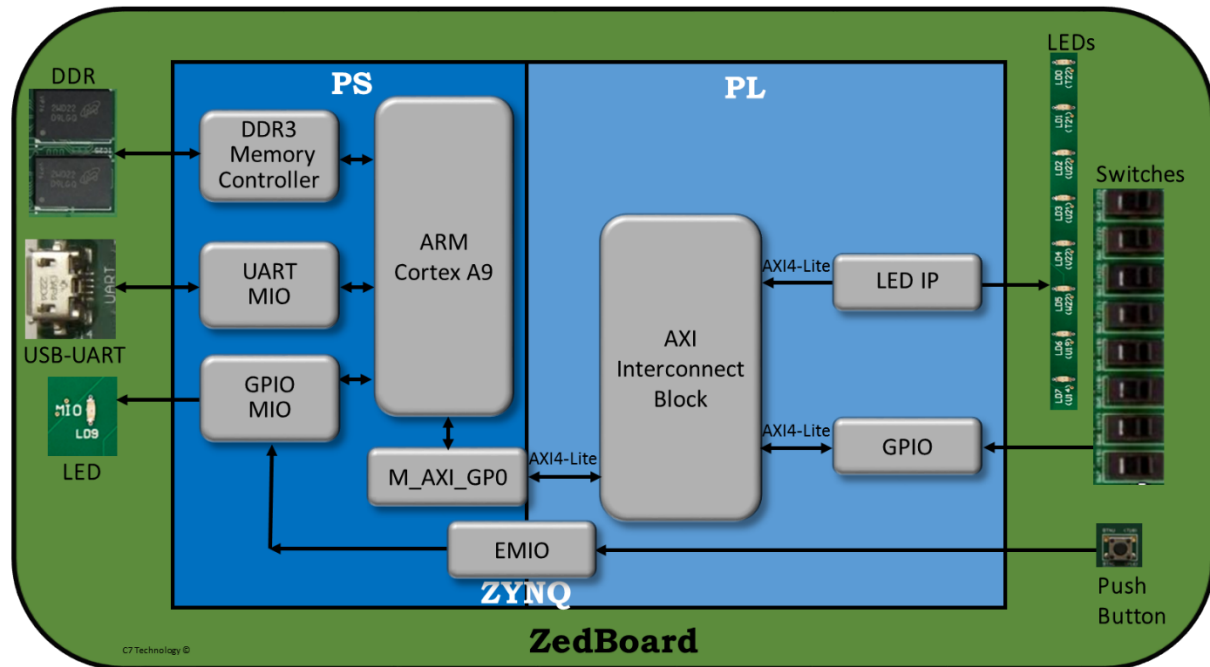
- Create an embedded system design using Vivado and SDK flow
- Configure the Processing System (PS)
- Add Xilinx standard IP in the Programmable Logic (PL) section
- Use and route the GPIO signal of the PS into the PL by using EMIO
- Use SDK to build a software project and verify the design functionality in the hardware by using the ZedBoard

## Procedure

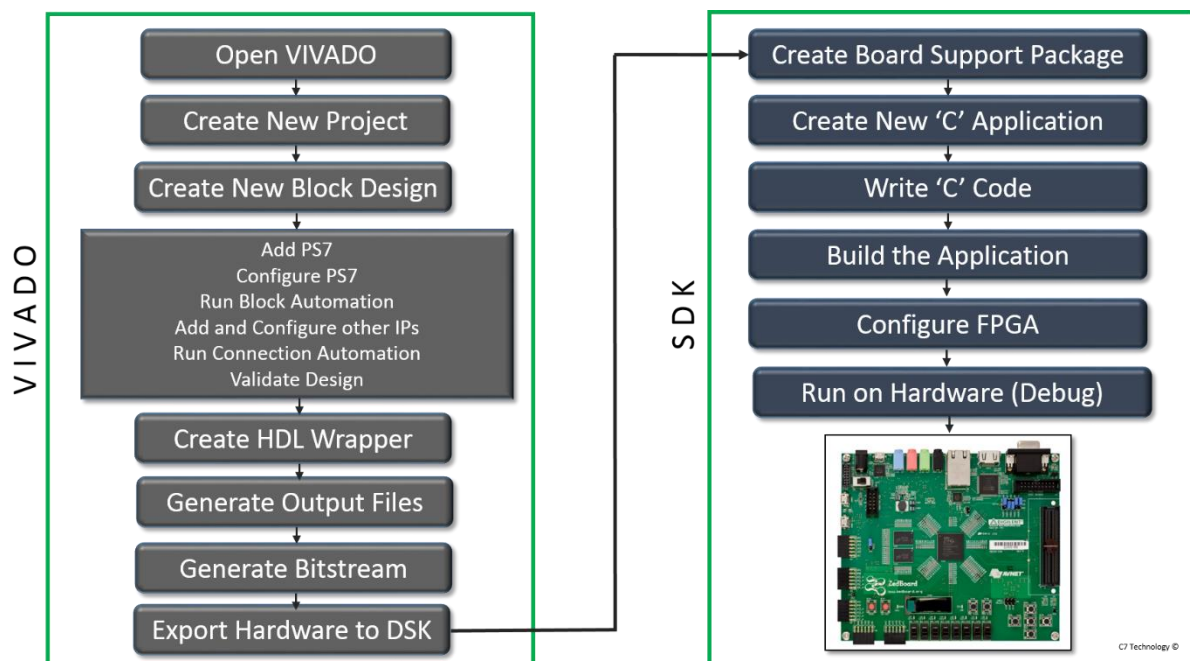
This lab comprises eight primary steps: You will create a top-level project using Vivado, create the processor system using the IP Integrator, add two instances of the GPIO IP, validate the design, generate the bitstream, export the design to the SDK, create an application in the SDK, and, test the design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

## Design Description

In this lab, you will design a completed embedded system consisting of the ARM Cortex-A9 processor SoC, two standard GPIO IPs to connect to eight on-board LEDs and to eight on board switches. The following block diagram represents the completed design.



## General Flow for this Lab



## Create a Vivado Project

## Step 1

**Objective:** Launch Vivado and create an empty project targeting the ZedBoard.

- 1-1.** Start *Vivado*.
- 1-2.** Click *Create New Project* to start the wizard. In the *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	lab_gpio_in_out
	Project Location	C:\....\SoC_School\labs\lab3
	Create Project Subdirectory	Check this option.
Click <b>Next</b>		
Project Type	Specify RTL	<b>Select RTL.</b> Keep do not specify sources at this time box unchecked
Click <b>Next</b>		
Add Sources	Do nothing	
Click <b>Next</b>		
Add Existing IP	Do Nothing	
Click <b>Next</b>		
Add Constraints	Do Nothing	
Default Part	Specify	Select <b>Boards</b>
	Board	Select <b>ZedBoard Zynq Evaluation and Development Kit, Rev. D.</b>
Click <b>Next</b>		
New Project Summary	Project Summary	Review the project summary
Click <b>Finish</b>		

## Creating the Hardware System Using IP Integrator Step 2

**2-1. Objective:** Create block design in the Vivado project using IP Integrator to generate the ARM Cortex-A9 processor based hardware system. Enable AXI\_M\_GP0 interface, FCLK\_RESET0\_N, and FCLK\_CLK0 ports. Add two instances of a GPIO Peripheral from the IP catalog to the processor system

**2-1-1.** In the Flow Navigator, click *Create Block Design* under IP Integrator.

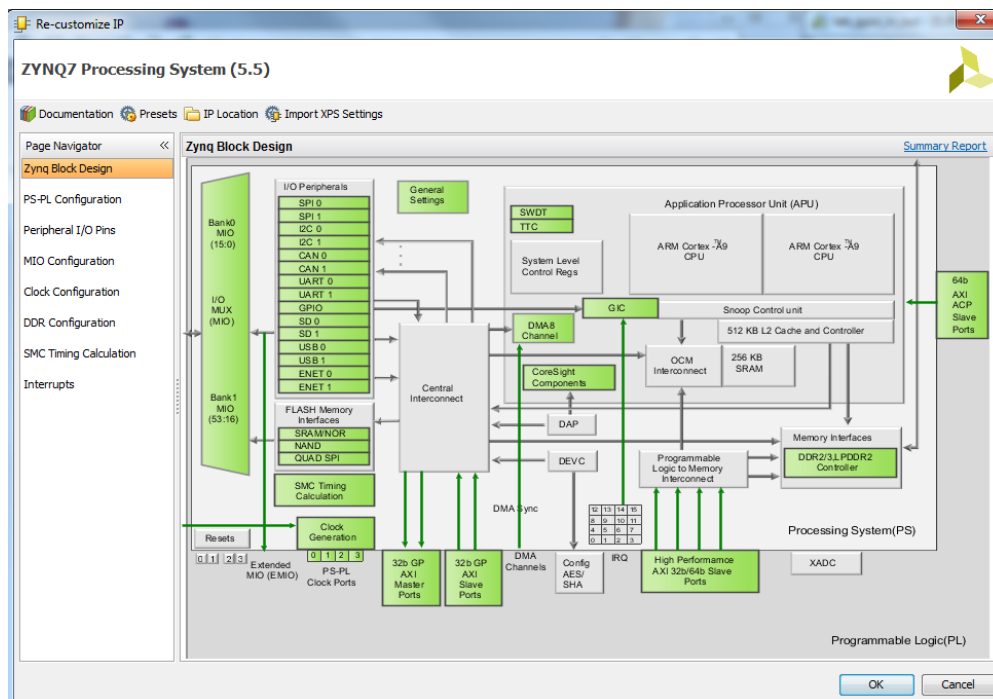
**2-1-2.** Name the block *lab\_gpio\_in\_out* and click *OK*.

**2-1-3.** Following the steps explained in previous labs add a *Zynq7 Processing System* block.

**2-1-4.** Click *Run Block Automation*, available in the green information bar. Then, select /processing\_system7\_0. Make sure *Apply Board Presets* option is checked and select *OK* in the *Run Block Automation* window (leave everything else as default).

**2-1-5.** Double click on the Zynq block to open the *Zynq7 Processing System Re-Customize IP* window.

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

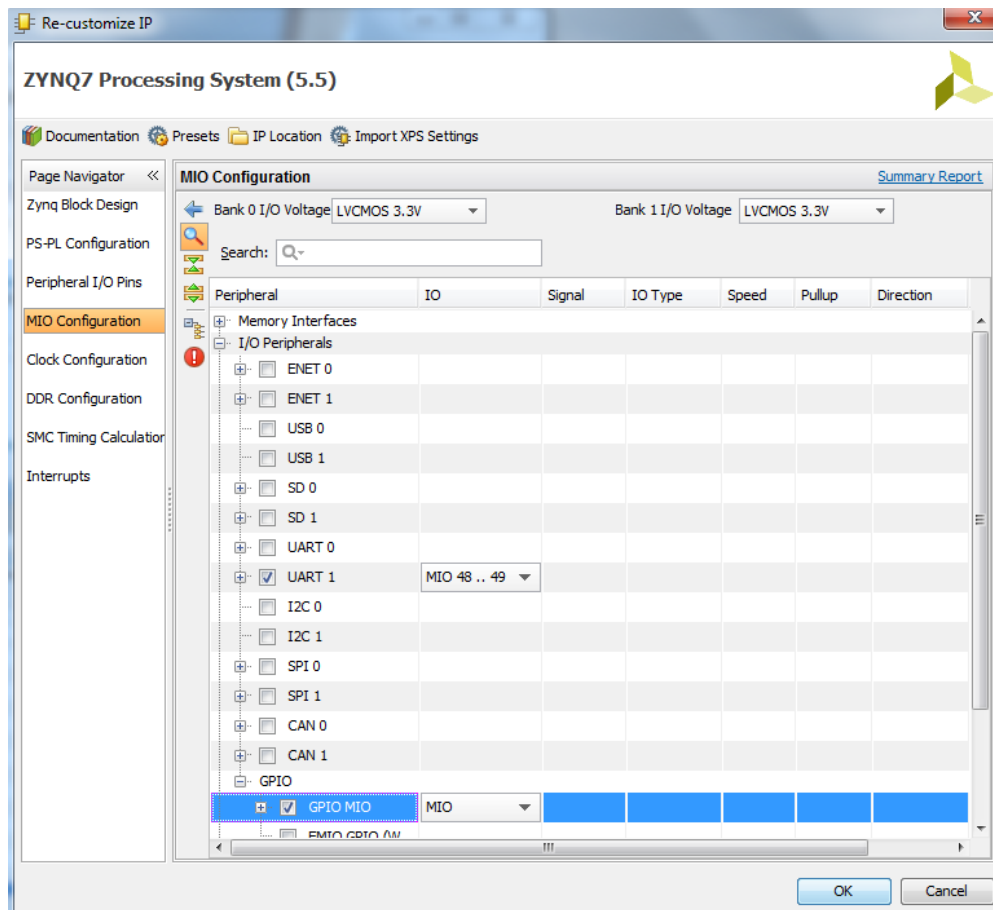


**2-1-6.** Configure the I/O Peripherals block to have UART 1 and GPIO support. Route 1-bit wide GPIO\_I port to the EMIO so it can be connected to a user IO pin

**2-1-6.1.** Click on the *MIO Configuration* panel to open its configuration form.

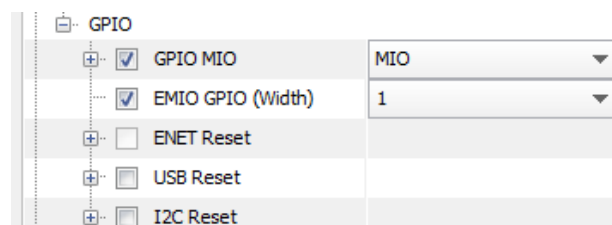
**2-1-6.2.** Expand the IO Peripherals.

**2-1-6.3.** Deselect all the peripherals except *UART 1* and *GPIO*.



**2-1-7.** Route the *GPIO PS* section of a 1-bit width to the *PL* side pad using the *EMIO* interface by doing the following:

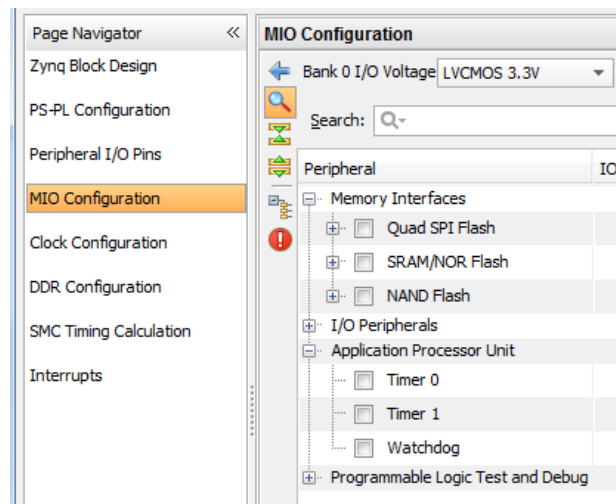
- Expand the *GPIO* tree
- Check the *EMIO GPIO (Width)* box. Then click in the right-column and select 1, from the pull down menu, as the width of the 'bus' going from the PS to the PL.



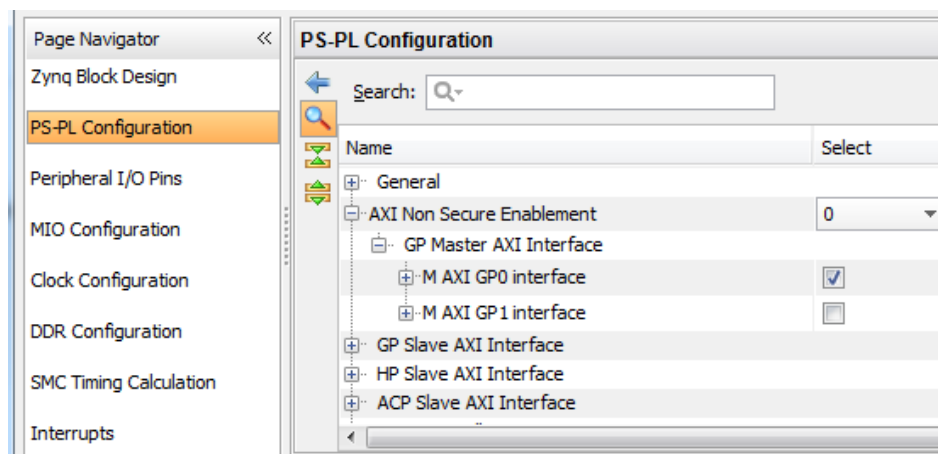
**2-1-8.** Uncheck *USB Reset* and *I2C Reset* options.

**2-1-9.** In the *MIO Configuration* panel, expand the *Memory Interfaces* and uncheck *QSPI*.

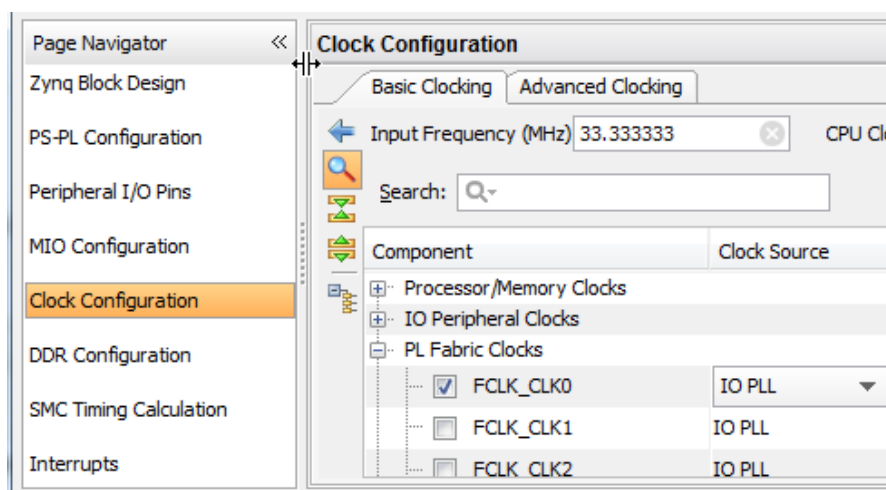
**2-1-10.** Expand the *Application Processing Unit* and uncheck the *Timer 0*.



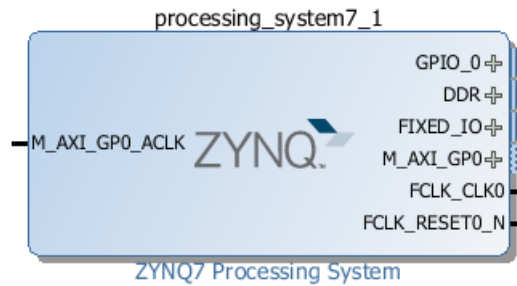
**2-1-11.** Go to the *PS-PL Configuration* options. Confirm that the *M AXI GP0 Interface* of the *GP Master AXI Interface* is checked.




**2-1-12.** Go to the *Clock Configuration* options. Confirm that the *FCLK-CLK0* option is checked. This clock will clock the PL logic (generated in the PS side).



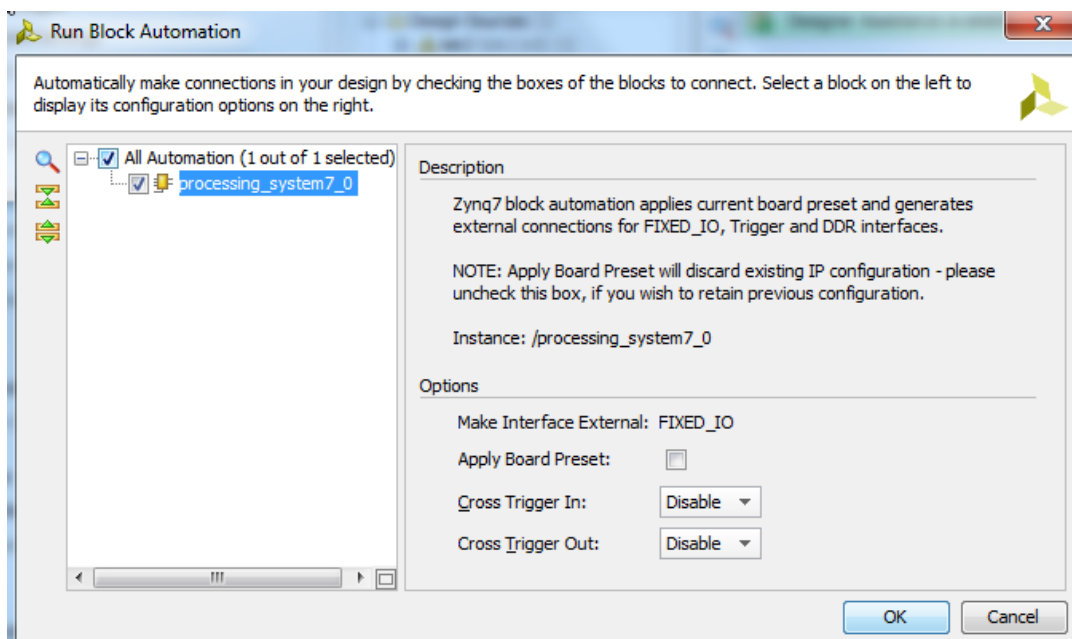
**2-1-13.** Click *OK* to close the configuration window. Then, the block diagram will be updated as shown below.



**2-2. Objective: Run Block Automation. Add one instance of GPIO. Connect the GPIO to the processing\_system7\_0 block.**

**2-2-1.** Click on  Designer Assistance available. [Run Block Automation](#).

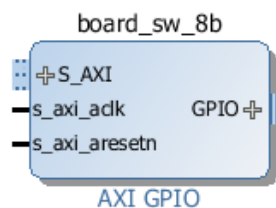
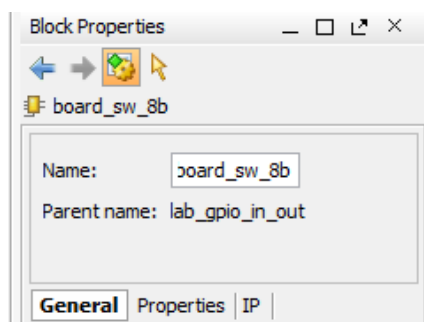
**2-2-2.** Select `/processing_system7_0`. Make sure *Apply Board Presets* is unchecked and select *OK* in the *Run Block Automation* window (leave everything else as default).



**2-2-3.** Click the Add IP icon  and search for *GPIO* in the catalog.

**2-2-4.** Double-click the *AXI GPIO* to add an instance of the *GPIO* core to the design.

**2-2-5.** Click on the *AXI GPIO* block to select it, and in the *General* tab of the *Block Properties* window, change the name to *board\_sw\_8b*.






- 2-2-6.** Double click on the *AXI GPIO* block to open the customization window. On the *Board* tab, for *GPIO*, click on *Custom* to view the dropdown menu options, and select the *sws\_8bits* option.

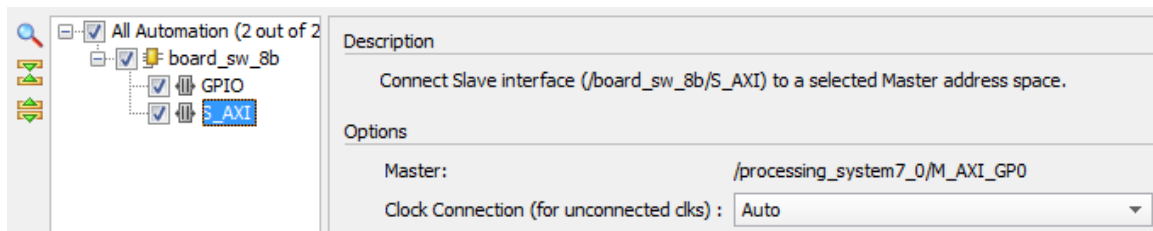
As the Zedboard was selected during the project creation, and a board support package is available for the Zedboard, Vivado has knowledge of available resources on the board.

- 2-2-7.** Click the *IP Configuration* tab. Notice the *GPIO Width* is set to 8 and is greyed out. If a board support package was not available, the width of the IP could be configured here. Leave unchecked *Enable Dual Channel* and *Enable Interrupt*.

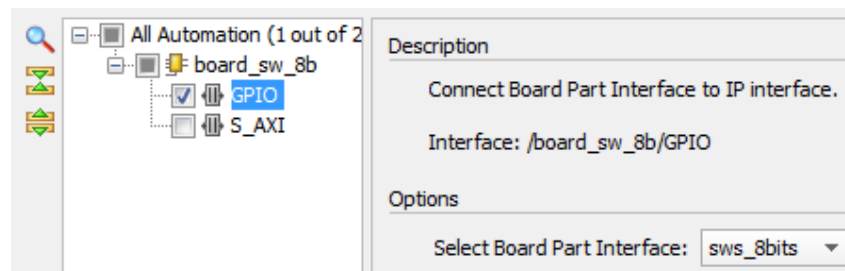
- 2-2-8.** Click *OK* to close the GPIO configuration window.

- 2-2-9.** Click on  **Designer Assistance available.** [Run Connection Automation](#)


- 2-2-10.** First select */board\_sw\_8b/S\_AXI*, and left the options with the values by default. This will connect the slave part of the GPIO 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).



- 2-3.** 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 that will go to the switches on the board.



- 2-3-1.** Click *OK* when prompted to automatically connect the master and slave interfaces.

The updated block design is now showed in the Diagram editor tab. Click the regenerate button (  ) to redraw the diagram.


Notice two additional blocks, one refereed to the reset block and other to the AXI interconnect block, have automatically been added to the design.

- 2-4.** ***Objective:*** Add another instance of GPIO with width of 8 bits. Name the instance as *led\_8bit* and connect it to the *processing\_system7\_0* instance.

- 2-4-1.** Follow the steps described above to add another instance of the *GPIO* peripheral.

- 2-4-2.** Change the name of the block to *board\_led\_8bit*.

- 2-4-3.** Double click on the *board\_led\_8b* block to customize the *Board Interface* to *led\_8bits*. Leave the *IP Configuration* options with the default values.

**2-4-4.** Click on  **Designer Assistance available. [Run Connection Automation](#)**. Select `/board_leds_8b/s_axi` and leave the *Clock Connection* as *Auto*.

**2-4-5.** Then, select `/board_leds_8b/gpio` and check that the *Select Board Part Interface* option be configured as *leds\_8bits*.

**2-4-6.** Click *OK* to automatically connect the master and slave interfaces.

Note that now the *AXI Interconnect* block has the second *AXI master (M01\_AXI)* port added and connected to the *S\_AXI* of the *board\_leds\_8b* block.

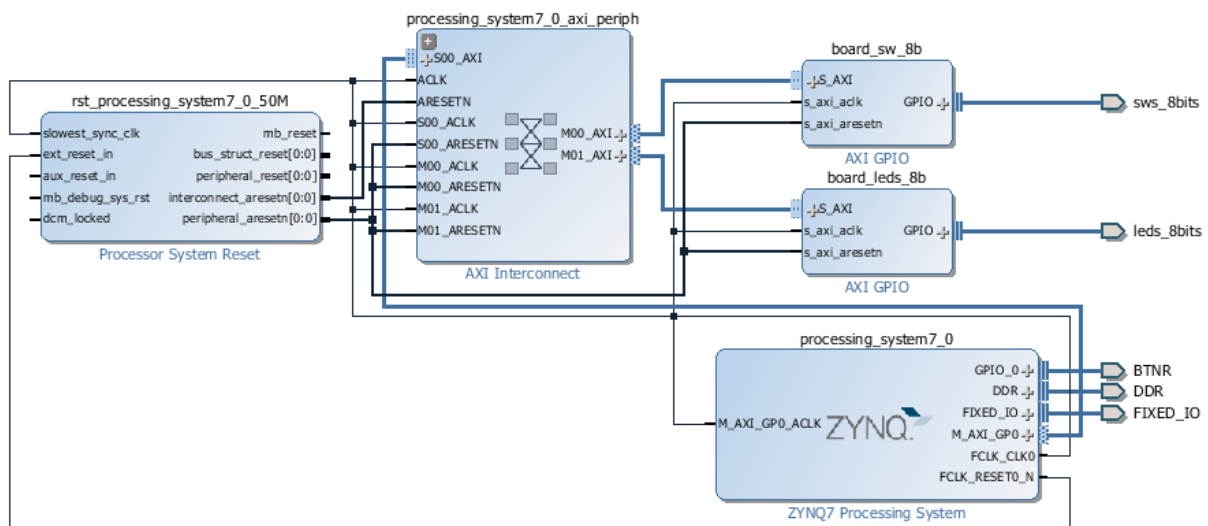
## 2-5. **Objective:** Make *GPIO\_0* external.

**2-5-1.** Right-click on the *GPIO\_0* pin of the *processing\_system7\_1* instance, and select *Make External*.

**2-5-2.** Select the *GPIO\_0* connector, and change the name to *BTNR* in its properties form.



At this stage the design should look like as shown below.



**2-6. Verify that the addresses are assigned to the two GPIO instances and validate the design for no errors.**

**2-6-1.** Select the *Address Editor* tab and see that the addresses are assigned to the two GPIO instances. They should look like as follows.

Cell	Slave In...	Base ...	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
board_leds_8b	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

The addresses should be in the `0x40000000` to `0xbfffffff` range as the instances are connected to *M\_AXI\_GP0* port of the processing system (PS).

**2-6-2.** Select *Tools > Validate Design* to run the design rule checker and to make sure that there are no design errors.

**2-6-3.** Select *File > Save Block Design* to save the design.

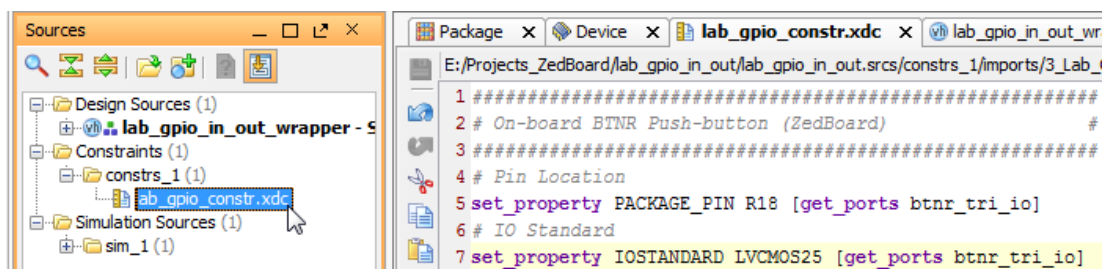
**2-7.** **Objective:** Add the provided Xilinx Design Constraints file (*lab\_gpio\_constr.xdc*).

**2-7-1.** Click the *Add Sources* button in the *Flow Navigator*.

**2-7-2.** Select *Add or Create Constraints*, and click *Next*.

**2-7-3.** The *Add or Create constraints* window will come up. Click *Add Files...* and browse to the *c:\.....\SoC\_School\labs\lab4\_gpio\_in\_out\lab4\_Constraint\_File* directory.

**2-7-4.** Select the *lab\_gpio\_constr.xdc* constraint file and click *OK*. Click *Finish* to add the constraint file to the project. The constraint file is very simple as it can be seen below. Double click on the file name to open it.



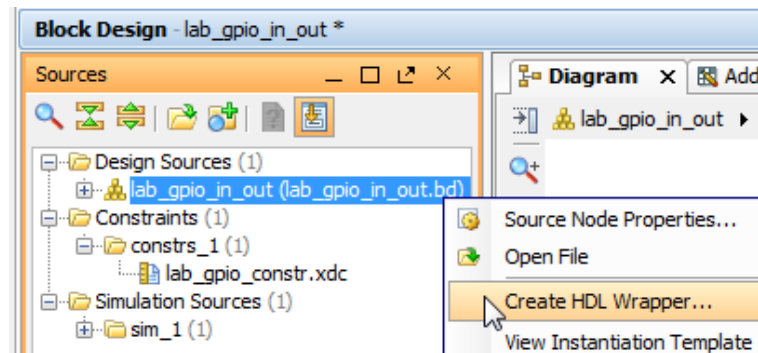
It only provides the constraint for the push button BTNR; its location, R18, and its IO Standard, LVCMOS25. The location and IO standard for the *switches* and the *LEDs* are already defined in the constraint for the board. This constraint is automatically added to the project when the board is selected as target of the project.

## Generate the Bitstream – Check IOs

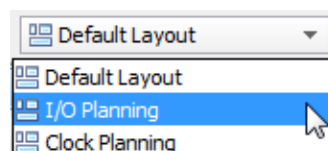
## Step 3

### 3-1. **Objective:** Create the top-level HDL of the embedded system.

- 3-1-1.** In the *Sources* pane, expand the *Design Sources*, right-click on the *lab\_gpio\_in\_out.bd* and select **Create HDL Wrapper**.



- 3-1-2.** In the Create HDL Wrapper window, leave the option “Let Vivado manage wrapper and auto-update” checked. Click **OK**.
- 3-1-3.** In the *Flow Navigator* pane, click *Run Synthesis*. Click on *Open Synthesized Design* once it finishes.
- 3-1-4.** From the *Default Layout* options select *I/O Planning*.



- 3-1-5.** On the *IO Ports* pane, check that the location for the *BTNR* input be *R18* as it was stated in the constraint file.

I/O Ports							
	Name	Direction	Boar...	Site	Fixed	Bank	I/O Std
	All ports (147)						
	DDR_50469 (71)	INOUT			✓		502 (Multiple)*
	FIXED_IO_50469 (59)	INOUT			✓		(Multiple) (Multiple)*
	GPIO_0_50469 (1)	INOUT			✓		34 LVCMOS25*
	btmr_tri_io (1)	INOUT			✓		34 LVCMOS25*
	btmr_tri_io[0]	INOUT		R18	✓		34 LVCMOS25*

- 3-1-6.** Click on the *Generate Bitstream* in the *Flow Navigator* pane to implement the design, and generate the bitstream.
- 3-1-7.** Click *Yes* to save the design and *OK* to run the necessary processes.
- 3-1-8.** Select the *Open Implemented Design* option when the bitstream generation is complete, and click *OK*.

---

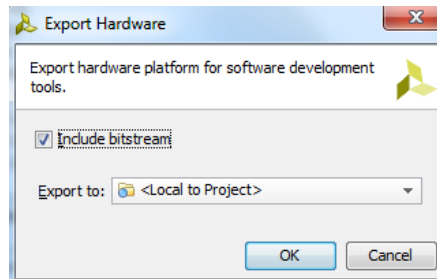
## Export the Design to the SDK

---

## Step 4

**4-1. Objective:** Start the SDK by exporting the implemented design.

**4-1-1.** First, select *File > Export > Export Hardware*. In the dialog box check the *Include Bitstream* option, since there is hardware implemented in the PL part of the Zynq. Click *OK*.



**4-1-2.** Select *File > Lunch SDK*.

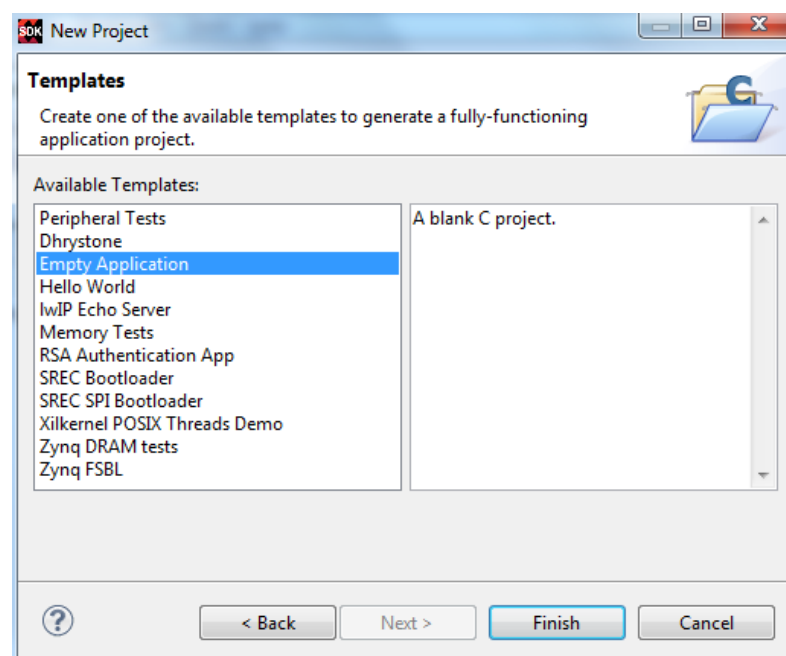
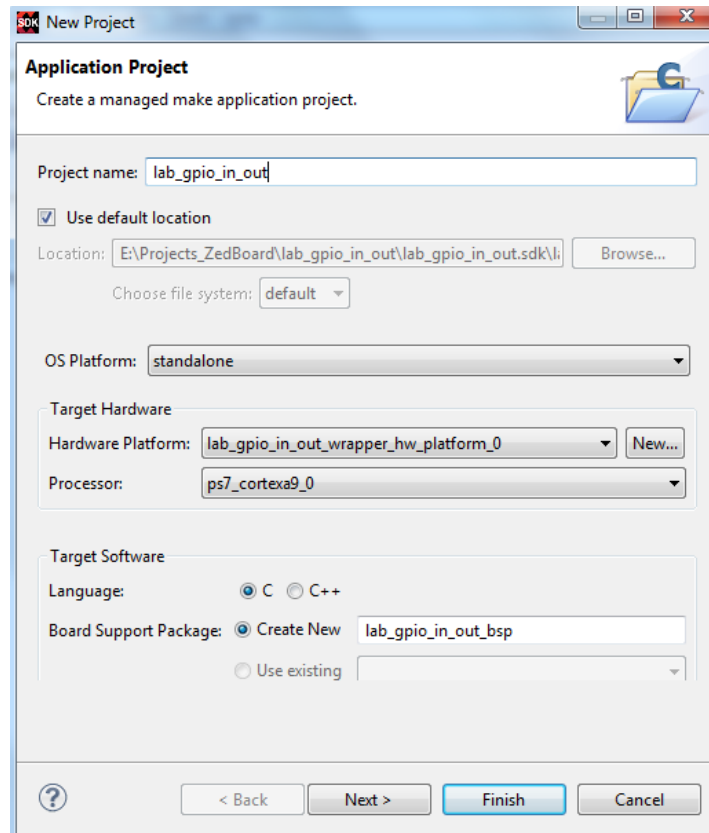
## Generate an Application in SDK

## Step 5

**5-1. Objective:** Generate a software platform project with default.

**5-1-1.** In SDK, select *File > New > Application Project*.

**5-1-2.** Use the following settings for your project (or something similar).



**5-1-3.** Select *lab\_gpio\_in\_out* in the project view, right-click, and select *Import*.

- 5-1-4.** Expand the *General* category and double-click on *File System*.
- 5-1-5.** Browse to the `c:\....\SoC_School\labs\lab_gpio_in_out\lab4_C_Code` folder.
- 5-1-6.** Select the `lab_gpio_in_out.c` source file and click *Finish*. A snippet of the source code is shown below.

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

static XGpioPs psGpioInstancePtr;
static int iPinNumber = 7; /*Led LD9 is connected to MIO pin 7*/

//=====

int main (void)
{
    XGpio sw, led;
    int i, pshb_check, sw_check;
    static XGpio GPIOInstance_Ptr;
    XGpioPs_Config*GpioConfigPtr;
    int xStatus;
    int iPinNumberEMIO = 54;
    u32 uPinDirectionEMIO = 0x0;
    u32 uPinDirection = 0x1;

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

    // AXI GPIO switches Initialization
    XGpio_Initialize(&sw, XPAR_BOARD_SW_8B_DEVICE_ID);

    // AXI GPIO leds Initialization
    XGpio_Initialize(&led, XPAR_BOARD_LEDS_8B_DEVICE_ID);
```

## Test in Hardware

## Step 6

**6-1. Objective:** Connect and power up the ZedBoard. Establish serial communications using a serial communication utility software. Verify the design functionality.

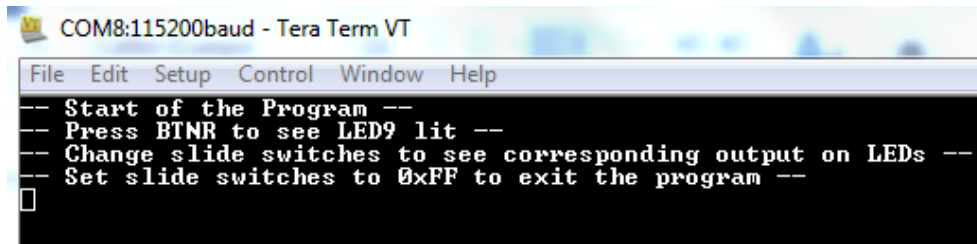
**6-1-1.** Connect the two micro-USB cables as it was explained in Lab2. Power up the ZedBoard.

**6-1-2.** Configure the serial communication software. Use Baud Rate of 115200, data bits 8, no parity, no flow control one stop bit.

**6-1-3.** Select *Xilinx Tools > Program FPGA*. Make sure that the *bitstream* path is pointing to right project. Then click the *Program* button. Look for the blue light on the ZedBoard indicating that the PL was successfully configured.

**6-1-4.** In the Project Explorer pane, select *lab\_gpio\_in\_out*, right-click and select **Run As > Launch on Hardware** to download the application, execute *ps7\_init*, and execute *lab\_gpio\_in\_out.elf*.

**6-1-5.** You should see something similar to the following output in the serial terminal console.




```
COM8:115200baud - Tera Term VT
File Edit Setup Control Window Help
-- Start of the Program --
-- Press BTNR to see LED9 lit --
-- Change slide switches to see corresponding output on LEDs --
-- Set slide switches to 0xFF to exit the program --
█
```

**6-1-6.** Press *BTNR* and the *LED9* should light up. Release the *BTNR* button and the *LED9* should go *OFF*.

**6-1-7.** Play with the slide switches and see the corresponding *LED* turning *ON* and *OFF*.

**6-1-8.** Set the slide switches to the *0xFF* position to exit the program.

Click the Terminate button (  ) on the console ribbon bar to terminate the execution if you want to terminate the application at anytime before setting the slide switches to the *0xFF* position.

**6-1-9.** Close *SDK* and *Vivado* programs by selecting *File > Exit* in each program.

**6-1-10.** Turn *OFF* the power on the board.



---

## Conclusion

---

In this lab, you create an ARM Cortex-A9 processor based embedded system in the Zynq device located on ZedBoard. You learned how to route the GPIO connected to the PS section to the FPGA (PL) pin using the EMIO. You instantiated the Xilinx standard GPIO IP two times to provide input and output functionality.

## Note

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