

# Building a Complete Embedded System

## Introduction

This lab guides you through the process of using Vivado and IP Integrator to create a complete Zynq ARM Cortex-A9 based processor system targeting ZedBoard Zynq development board. You will use the Block Design feature of IP Integrator to configure the Zynq PS and add IP 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 using EMIO
- Use SDK to build a software project and verify the design functionality in hardware.

## Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

This lab comprises six 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 to the SDK, create an application in the SDK, and, test the design in hardware.

## Design Description

In this lab, you will design a complete embedded system consisting of the ARM Cortex-A9 PS, and two standard GPIO IPs to connect to on-board LEDs and their corresponding switches. The following block diagram represents the completed design (**Figure 1**).

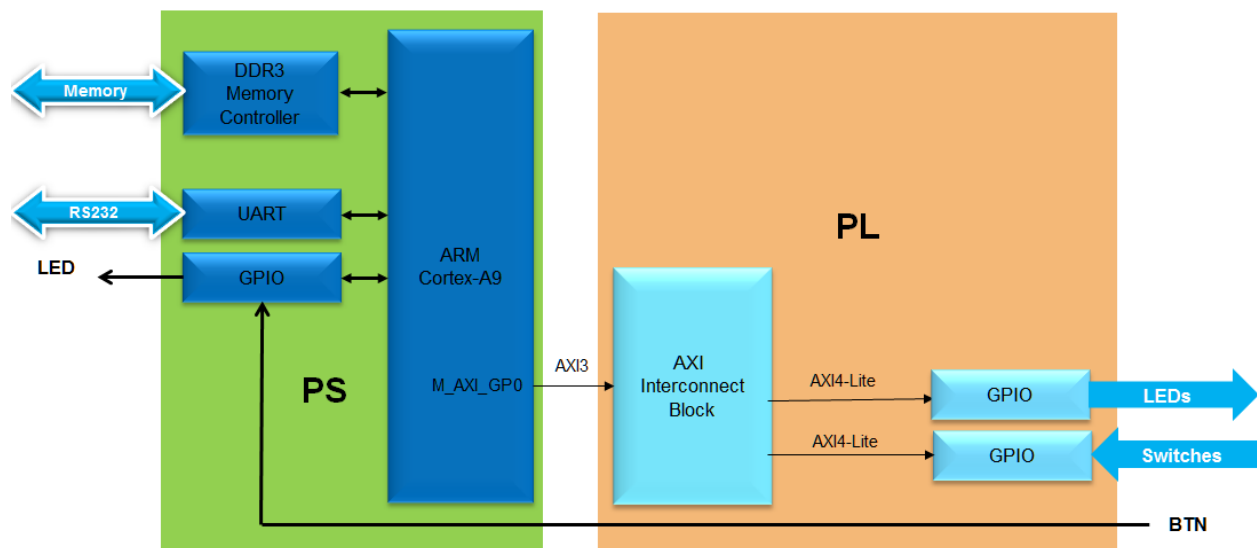
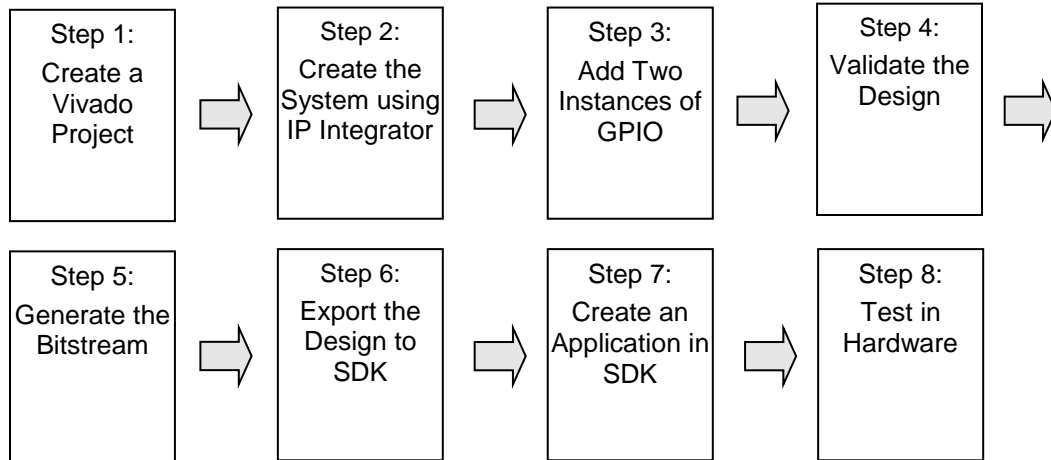


Figure 1. Completed Design

## General Flow for this Lab



In the instructions below;

{**sources**} refers to: C:\Xilinx\_trn\Zynq\_adv\lab\_sources

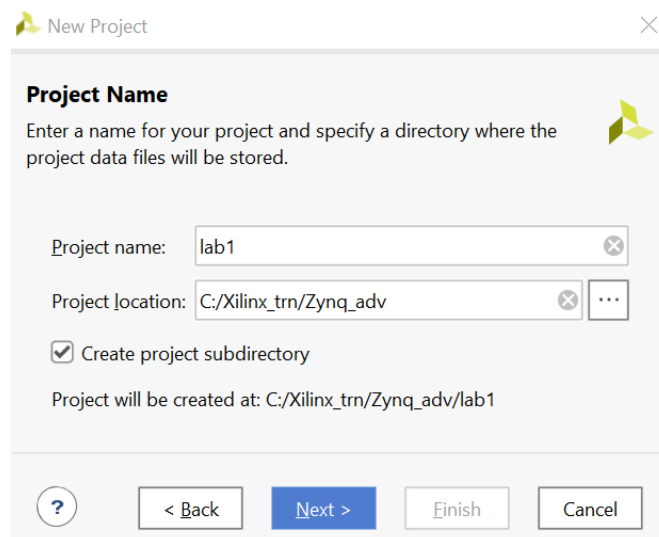
{**labs**} refers to : C:\Xilinx\_trn\Zynq\_adv

## Create a Vivado Project

## Step 1

### 1-1. Launch Vivado and create an empty project targeting the ZedBoard Zynq Evaluation and Development Kit, selecting Verilog language.

- 1-1-1. Open Vivado by selecting **Start > Xilinx Design Tools WEB 2018.3 > Vivado 2018.3**
- 1-1-2. Click **Create Project** to start the wizard. You will see the *Create a New Vivado Project* wizard page. Click **Next**.
- 1-1-3. Click the Browse button of the *Project Location* field of the **New Project** form, browse to **C:\Xilinx\_trn\Zynq\_adv**, and click **Select**.
- 1-1-4. Enter **lab1** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.



The screenshot shows the 'New Project' dialog box in Vivado. The 'Project Name' field is filled with 'lab1'. The 'Project Location' field is filled with 'C:/Xilinx\_trn/Zynq\_adv'. The 'Create project subdirectory' checkbox is checked. The 'Project will be created at' path is 'C:/Xilinx\_trn/Zynq\_adv/lab1'. The 'Next >' button is highlighted in blue.

Figure 2. Project Name Entry

- 1-1-5. Select the **RTL Project** option in the *Project Type* form, and click **Next**.
- 1-1-6. Select **Verilog** as the *Target Language* and as *Simulation Language* in the *Add Sources* form, and click **Next**.
- 1-1-7. Click **Next** again to skip adding constraints.
- 1-1-8. In the *Default Part* form, click **Boards** filter button.
- 1-1-9. Select the **ZedBoard Zynq Evaluation and Development Kit** of the appropriate Board Version based on the board you have.

It is important to select the correct revision of the board, as the FSBL created later will generate different code depending on the board revision (i.e. silicon version) you are using. For the Zedboard the revision is likely to be "C" or "D".

New Project ×



**Default Part**  
Choose a default Xilinx part or board for your project.

Parts | **Boards**

[Reset All Filters](#)

Vendor:  Name:  Board Rev:

Search:

Display Name	Preview	Vendor	File Version	Part	I/O Pin Count	Board Rev	Available IOBs
Nexys4 DDR		digilentinc.com	1.1	xc7a100tcsq324-1	324	C.1	210
ZedBoard Zynq Evaluation and Development Kit <a href="#">Add Daughter Card</a> <a href="#">Connections</a>		em.avnet.com	1.4	xc7z020clg484-1	484	d	200
Artix-7 AC701 Evaluation Platform <a href="#">Add Daughter Card</a> <a href="#">Connections</a>		xilinx.com	1.4	xc7a200tfbg676-2	676	1.1	400

? < Back Next > Finish Cancel

**Figure 3. Board Selection**

**1-1-10.** Click **Next**.

**1-1-11.** Click **Finish** to create an empty Vivado project.

## Creating the Hardware System Using IP Integrator

## Step 2

### 2-1. Create a block design in the Vivado project using IP Integrator to generate the ARM Cortex-A9 processor based hardware system.

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

2-1-2. Name the block as **system** and click **OK**.

2-1-3. In the Block Diagram click on the **+** button.

2-1-4. Once the IP Catalog is open, type zy into the Search bar, and double click on the **ZYNQ7 Processing System** entry to add it to the design.

2-1-5. Click on **Run Block Automation**.

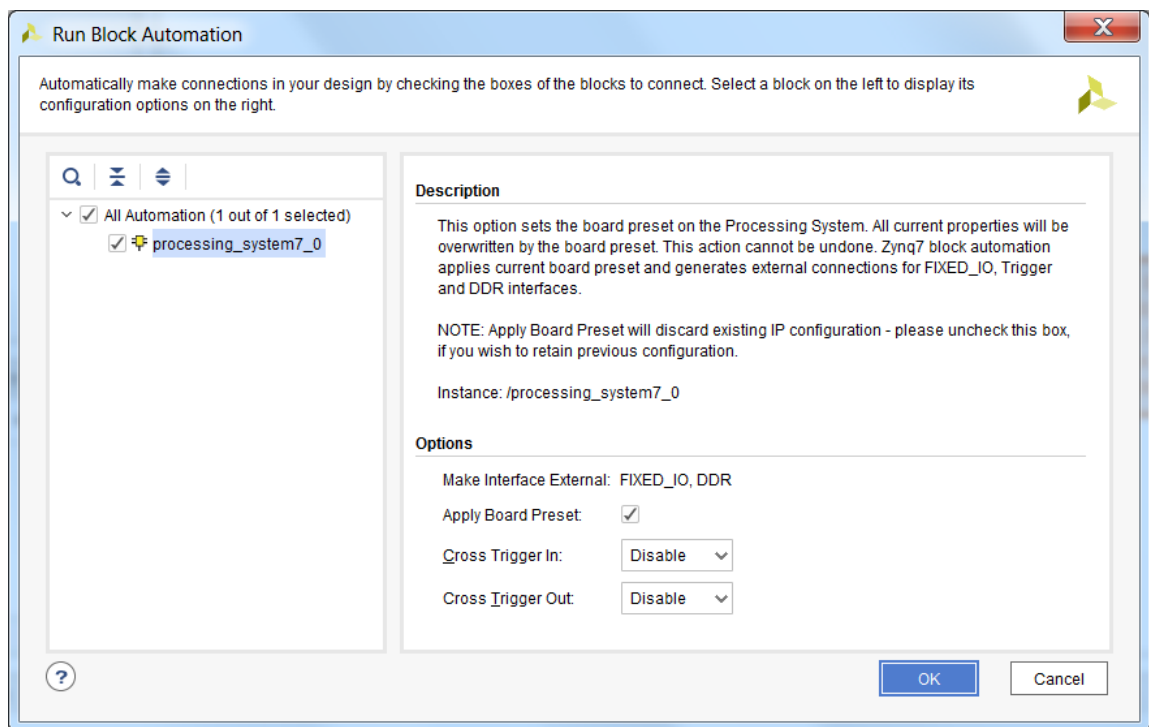


Figure 4. Zynq Block Automation View

2-1-6. Click **OK** to automatically configure the board presets.

2-1-7. Double click on the **Zynq7 Processing System** block to open the *Customization* window for the Zynq processing system.

A block diagram of the Zynq PS should now be open, showing various configurable blocks of the Processing System.

At this stage, designer can click on various configurable blocks (highlighted in green) and change the system configuration.

Re-customize IP

✕

## ZYNQ7 Processing System (5.5)



Documentation Presets IP Location Import XPS Settings

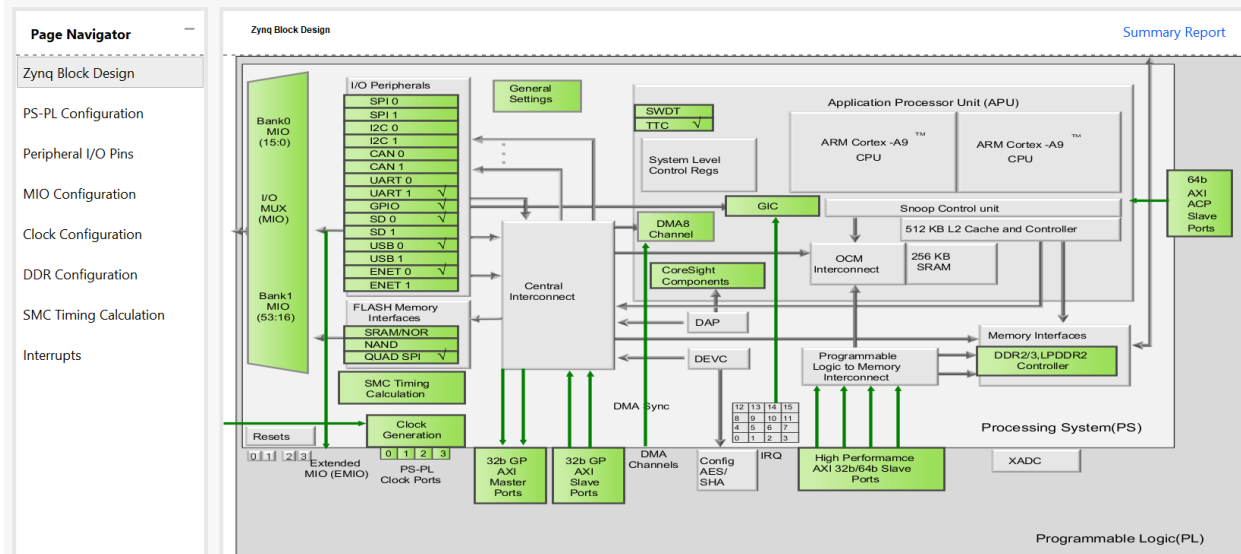


Figure 5. Zynq Processing System Configuration View (ZedBoard)

**2-2. 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.**

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

**2-2-2.** Expand the I/O Peripherals

**2-2-3.** Deselect ENET 0, USB 0, SD 0

**2-2-4.** Expand GPIO and deselect USB Reset and I2C Reset.

Re-customize IP

## ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

**Page Navigator**

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration**
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

**MIO Configuration**

Bank 0 I/O Voltage: LVC... Bank 1 I/O Voltage: LVC...

Search: Q-

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction	Polarity
I/O Peripherals							
> <input type="checkbox"/> ENET 0							
> <input type="checkbox"/> ENET 1							
<input type="checkbox"/> USB 0							
<input type="checkbox"/> USB 1							
> <input type="checkbox"/> SD 0							
> <input type="checkbox"/> SD 1							
> <input type="checkbox"/> UART 0							
> <input checked="" type="checkbox"/> UART 1	MIO 48 .. 49						
<input type="checkbox"/> I2C 0							
<input type="checkbox"/> I2C 1							
> <input type="checkbox"/> SPI 0							
> <input type="checkbox"/> SPI 1							
> <input type="checkbox"/> CAN 0							
> <input type="checkbox"/> CAN 1							
GPIO							
> <input checked="" type="checkbox"/> GPIO MIO	MIO						
<input type="checkbox"/> EMIO GPIO (Width)							
> <input type="checkbox"/> ENET Reset							
> <input type="checkbox"/> USB Reset							
> <input type="checkbox"/> I2C Reset							

**Figure 6. Selecting UART 1 and GPIO Peripherals of PS**

- 2-2-5. Route the PS section GPIO of a 1-bit width to the PL side pad using the EMIO interface by doing the following:
- Under GPIO, select the check-box for the *EMIO GPIO (Width)* to use the EMIO GPIO. Then click in the right-column and select 1 as the width. The EMIO will be connected to the first user GPIO available which will be channel number 54 (Channels 0 to 53 are available to PS).

GPIO

> ☒ GPIO MIO MIO

☒ EMIO GPIO (Width) 1

**Figure 7. Routing GPIO to PL**

## 2-3. Deselect TTC device.

- 2-3-1. Expand the **Application Processing Unit** and uncheck the **Timer 0**.

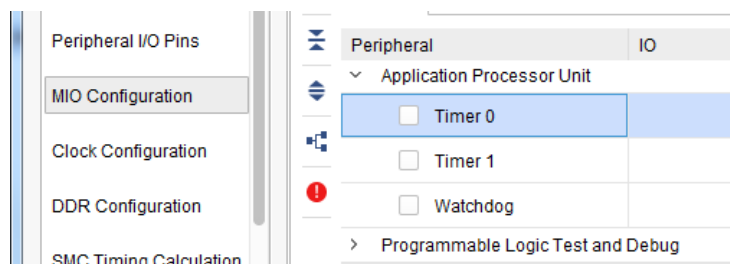



Figure 8. Deselecting Timer

2-3-2. Click **OK**.

2-3-3. In the Block Diagram click on Regenerate Layout  button.

The configuration form will close and the block diagram will be updated as shown below.

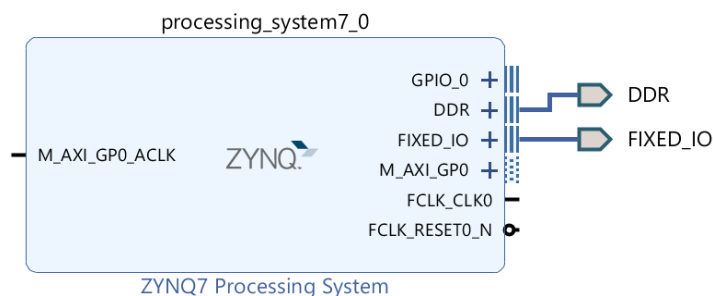



Figure 9. ZYNQ7 Processing System block

2-4. **Add one instance of GPIO and name it *switches*. Connect the block to the Zynq.**

2-4-1. Click the **Add IP**  button and search for **AXI GPIO** in the catalog.

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

2-4-3. Click on the **AXI GPIO** block to select it, and in the *Block properties* tab, change the name to **switches**.

2-4-4. Double click on the **AXI GPIO** block to open the customization window.

2-4-5. Under *Board Interface*, for *GPIO*, click on **Custom** to view the dropdown menu options, and select **sws 8Bits** for the Zedboard.

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

2-4-6. Click the **IP Configuration** tab. Notice the GPIO Width is set to 8 (Zedboard) and is greyed out. If a board support package was not available, the width of the IP could be configured here.

2-4-7. Click **OK** to finish configuring the GPIO and to close the *Re-Customize IP* window.



- 2-4-8. Click on **Run Connection Automation**, and select **switches** (which will include GPIO and S\_AXI)

Click on GPIO and S\_AXI to check the default connections for these interfaces.

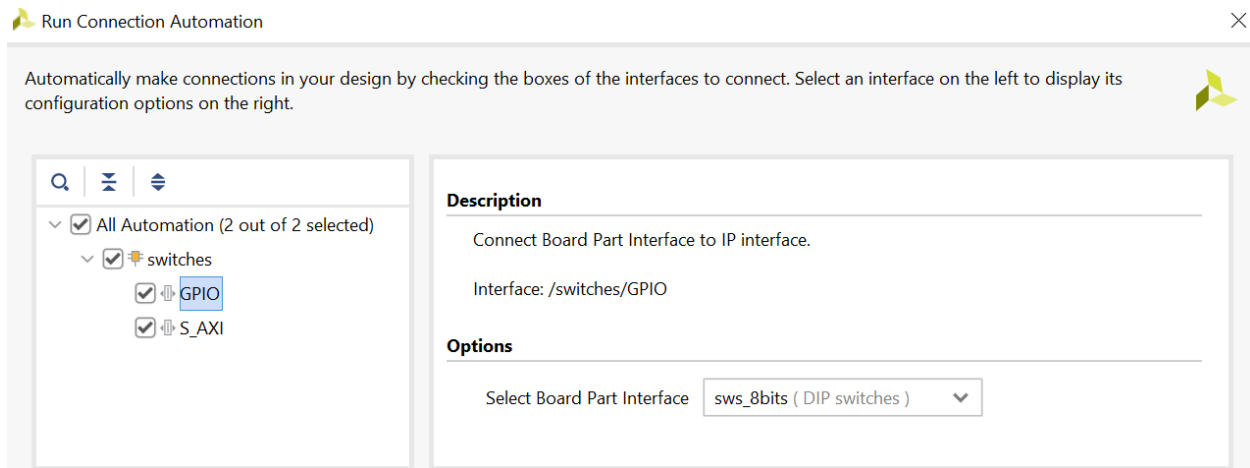


Figure 10. Connection Automation for the GPIO (ZedBoard)

- 2-4-9. Click **OK** to automatically connect the *S\_AXI* interface to the Zynq *M\_AXI\_GP0* port (through the AXI interconnect block), and the GPIO port of the **switches** to an external interface.

Notice that after block automation has been run, two additional blocks that are required to connect the initial blocks, *Processor System Reset* and *AXI Interconnect*, have automatically been added to the design.

- 2-5. **Add another instance of GPIO, name the instance *leds* and connect it to the Zynq. Configure its GPIO port.**

- 2-5-1. Add another instance of the *GPIO* peripheral.

- 2-5-2. Change the name of the block to **leds**.

- 2-5-3. Double click on the leds block, and select **leds 8bits** Zedboard for the *GPIO* interface

- 2-5-4. Click on **Run Connection Automation**

- 2-5-5. Click **leds**, and check the connections for *GPIO* and *S\_AXI* as before

- 2-5-6. Click **OK** to automatically connect the interfaces as before.

Notice that the AXI Interconnect block has the second master AXI (*M01\_AXI*) port added and connected to the *S\_AXI* of the leds.

- 2-6. **Connect the EMIO to the BTN**

- 2-6-1. Right-click on the **GPIO\_0** pin of the *processing\_system7\_0* instance, and select **Make External** to create an external port.

2-6-2. Select the newly created *GPIO\_0* port, and change the name to **btn** in *External Interface Pro* form.

2-6-3. In the Block Diagram click on Regenerate Layout button.

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

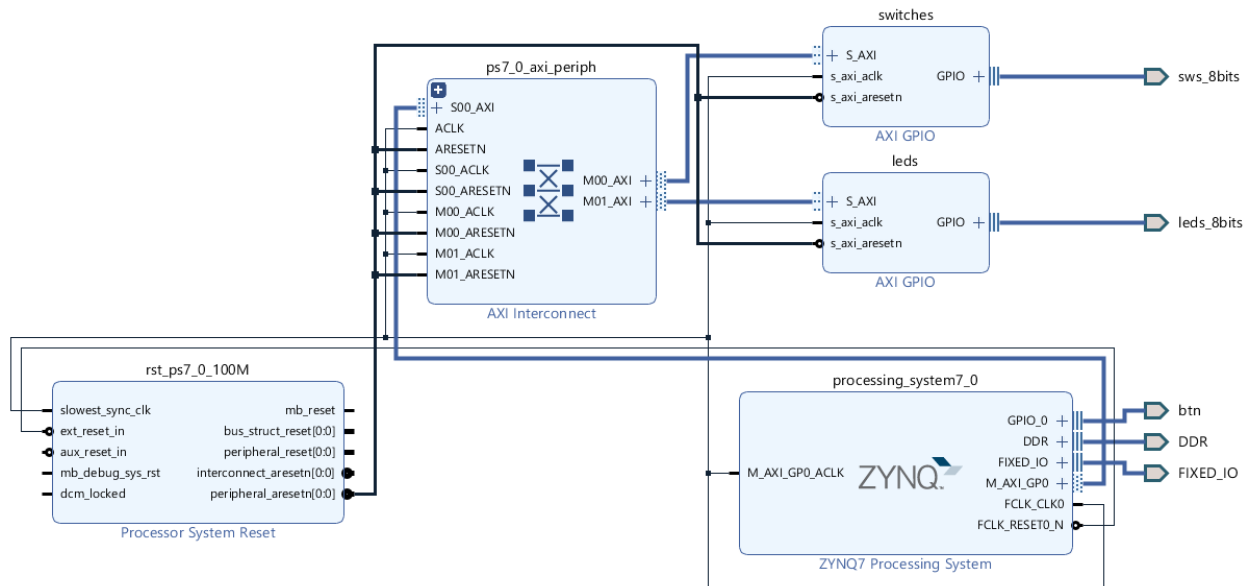


Figure 11. Completed design

2-7. **Verify that the addresses are assigned to the two GPIO instances and validate the design for no errors.**

2-7-1. 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 Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [ 1G ])					
switches	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
leds	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

Figure 12. Assigned addresses

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

2-7-2. Select the *Diagram* tab, and click on the  (Validate Design) button to make sure that there are no errors.

Ignore warnings.

2-7-3. Select **File > Save Block Design** to save the design.

**2-8. Add the provided Xilinx Design Constraints file (lab1\_<board>.xdc), which contains the BTN's location constraint, to the project.**

**2-8-1.** Board awareness is not being used for the EMIO button, so the pin constraints need to be provided for this interface. Click the **Add Sources** button in the *Flow Navigator*.

**2-8-2.** Select **Add or create constraints**, and click **Next**.

**2-8-3.** The *Add or Create* constraints window will appear. Click the “plus” then **Add Files...** and browse to the **C:\Xilinx\_trn\Zynq\_adv\lab\_sources\lab1** directory.

**2-8-4.** Select the **lab1\_zedboard.xdc** file, and click **OK**.

**2-8-5.** In the Add Source window click **Copy constraints files into project**

**2-8-6.** Click **Finish** to add the constraint file to the project.

## Generate the Bitstream

## Step 3

### 3-1. Create the top-level HDL of the embedded system. Add the provided constraints file and generate the bitstream.

- 3-1-1. In Vivado, select the *Sources* tab, expand the *Design Sources*, right-click the *system.bd* and select **Create HDL Wrapper...**

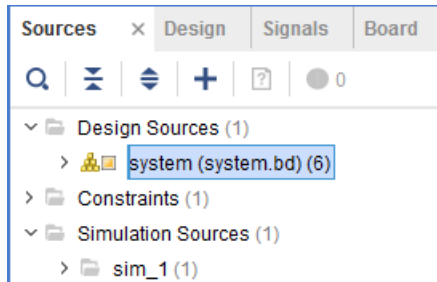


Figure 13. Selecting the system design to create the wrapper file

- 3-1-2. Click **OK** when prompted to allow Vivado to automatically manage this file.

The wrapper file, *system\_wrapper.v*, is generated and added to the hierarchy.

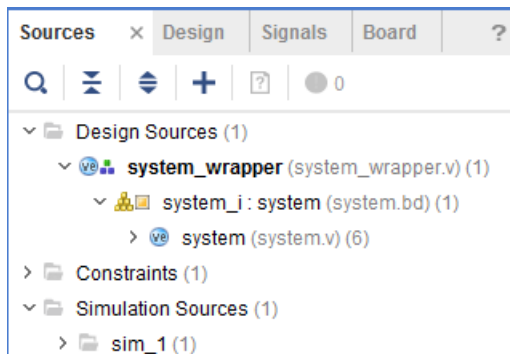


Figure 14. Design Hierarchy View

- 3-1-3. Click on the **Generate Bitstream** in the *Flow Navigator* pane to synthesize and implement the design, and generate the bitstream. Click **Save** and **Yes** if prompted. Click **OK** to launch the runs.
- 3-1-4. When the bitstream generation is complete, click **Cancel**.

## Export the Design to the SDK

## Step 4

### 4-1. Exporting the design and launch SDK

- 4-1-1. Export the hardware configuration by clicking **File > Export > Export Hardware...** Tick the box to include the bitstream.

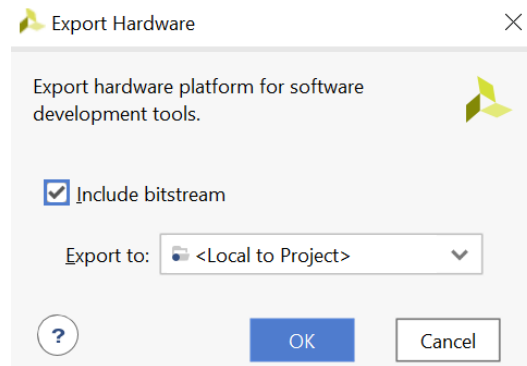


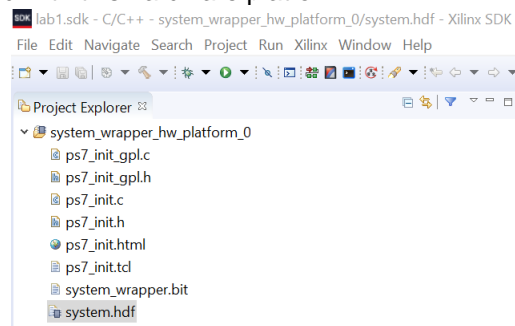
Figure 15. Exporting the hardware

- 4-1-2. Click **OK**.

- 4-1-3. Launch SDK by clicking **File > Launch SDK** and click **OK**

(Launching SDK from Vivado will automatically load the SDK workspace associated with the current project. If launching SDK standalone, the workspace will need to be selected.)

SDK should open and automatically create a hardware platform project based on the configuration exported from Vivado. A board support package and software application will be created and associated with this hardware platform

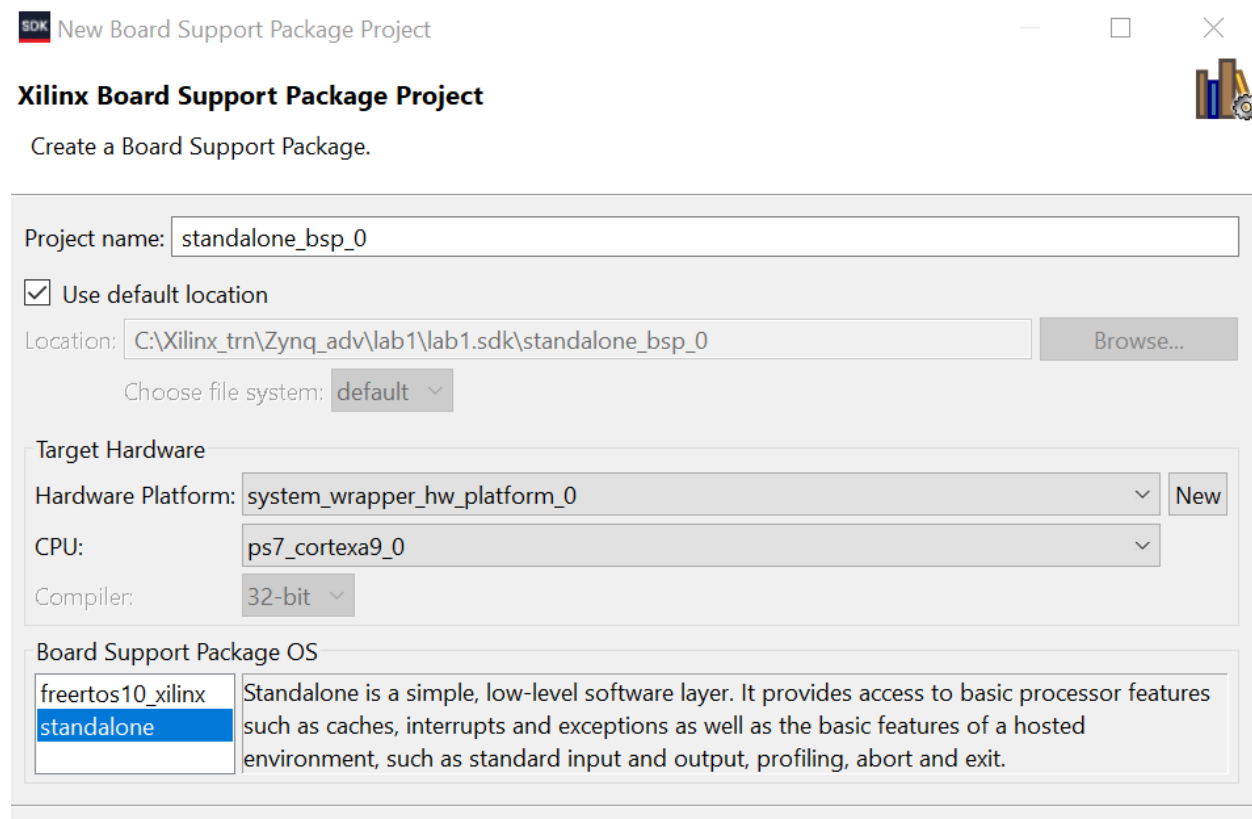


## Generate an Application in SDK

## Step 5

### 5-1. Generate a board support package project with default settings and default software project name.

#### 5-1-1. Select **File > New > Board Support Package**



**Figure 16. Create BSP**

#### 5-1-2. Click **Finish** with the default settings selected (using the Standalone operating system).

This will open the Software Platform Settings form showing the OS and libraries selections.

#### 5-1-3. Click **OK** to accept the default settings as we want to create a **standalone\_bsp\_0** software platform project without any additional libraries.

#### 5-1-4. The library generator will run in the background and will create the **xparameters.h** file in the **lab1.sdk\standalone\_bsp\_0\ps7\_cortexa9\_0\include** directory.

### 5-2. Create an empty application project, named lab1, and import the provided lab1.c file.

#### 5-2-1. Select **File > New > Application Project**.

#### 5-2-2. In the *Project Name* field, enter **lab1** as the project name.

5-2-3. Select the *Use existing* option in the *Board Support Package* field.

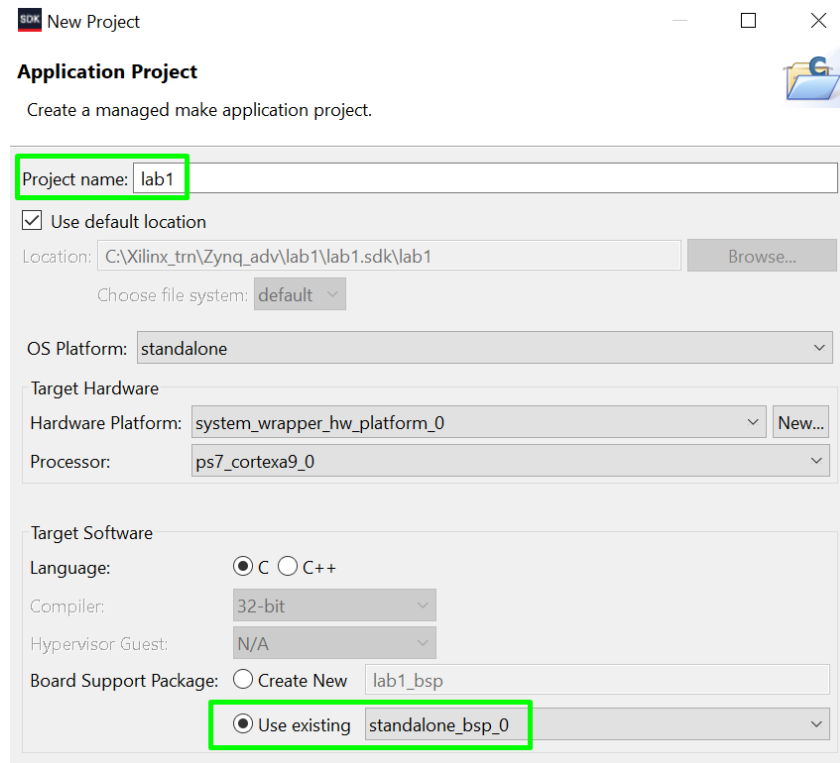


Figure 17. Create a Blank Application Project

5-2-4. Click **Next**.

5-2-5. Select the **Empty Application** template and click **Finish**.

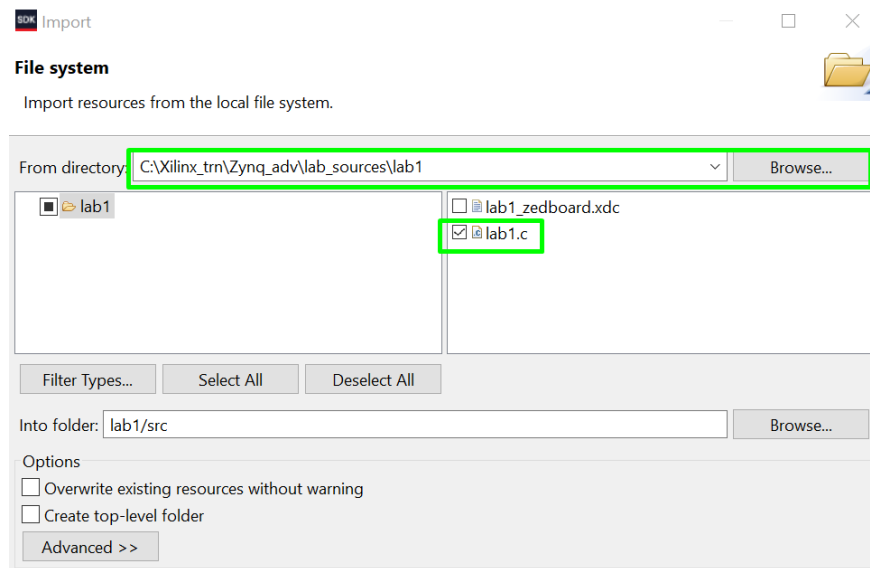
The lab1 project will be created in the Project Explorer window of SDK.

5-2-6. Select **lab1 > src** directory in the project view, right-click, and select **Import**.

5-2-7. Expand the **General** category and double-click on **File System**.

5-2-8. Browse to the **C:\Xilinx\_trn\Zynq\_adv\lab\_sources\lab1** folder and click **OK**.

5-2-9. Select the **lab1.c** source file and click **Finish**.



**5-2-10.** Open imported lab1.c file.

A snippet of the source code is shown in the following figure.

The code reads from the switches, and writes to the LEDs.

The BTN is read, and written to the LED.



```

1  #include "xparameters.h"
2  #include "xgpio.h"
3  #include "xgpiops.h"
4
5  static      XGpioPs psGpioInstancePtr;
6  static int   iPinNumber = 7; /*Led LD9 on ZedBoard is connected to MIO pin 7*/
7  //=====
8  int main (void){
9      XGpio          sw, led;
10     int            i, pshb_check, sw_check;
11     XGpioPs_Config *GpioConfigPtr;
12     int            iPinNumberEMIO = 54;
13     u32            uPinDirectionEMIO = 0x0;
14     u32            uPinDirection = 0x1;
15     // Start of the Program
16     xil_printf("-- Start of the Program --\r\n");
17
18     // AXI GPIO switches Initialization
19     XGpio_Initialize(&sw, XPAR_SWITCHES_DEVICE_ID);
20
21     // AXI GPIO leds Initialization
22     XGpio_Initialize(&led, XPAR_LEDS_DEVICE_ID);
23
24     // PS GPIO Initialization
25     GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
26     XGpioPs_CfgInitialize(&psGpioInstancePtr, GpioConfigPtr, GpioConfigPtr->BaseAddr);
27     //PS GPIO pin setting to Output
28     XGpioPs_SetDirectionPin(&psGpioInstancePtr, iPinNumber, uPinDirection);
29     XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, iPinNumber, 1);
30
31     //EMIO PIN Setting to Input port
32     XGpioPs_SetDirectionPin(&psGpioInstancePtr, iPinNumberEMIO, uPinDirectionEMIO);
33     XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, iPinNumberEMIO, 0);
34
35     xil_printf("-- Press BTNR (Zedboard) to see the LED light                --\r\n");
36     xil_printf("-- Change slide switches to see corresponding output on LEDs  --\r\n");
37     xil_printf("-- Set slide switches to 0xEE to exit the program              --\r\n");
38
39     while (1)
40     {
41         sw_check = XGpio_DiscreteRead(&sw, 1);
42         XGpio_DiscreteWrite(&led, 1, sw_check);
43         pshb_check = XGpioPs_ReadPin(&psGpioInstancePtr, iPinNumberEMIO);
44         XGpioPs_WritePin(&psGpioInstancePtr, iPinNumber, pshb_check);
45         if((sw_check & 0xFF) == 0xEE) break;
46         // delay loop
47         for (i=0; i<9999999; i++);
48     }
49     //End of Program
50     xil_printf("-- End of Program --\r\n");
51     return 0; }

```

Figure 18. Snippet of Source Code


## Test in Hardware


## Step 6

### 6-1. Connect and power up the board. Establish serial communications using the SDK's Terminal tab. Verify the design functionality.

6-1-1. Connect and power up the board and TWO usb cables (one cable should be connected to PROG connector; one cable should be connected to UART connector).

6-1-2. Set slide switches to 0x00 – switch all switches to OFF position (switch these down).

6-1-3. Select the  **Terminal** tab. If it is not visible then select **Window > Show view > Other > Terminal > Terminal**.

6-1-4. Click on  and select appropriate COM port (depending on your computer), and configure the terminal with the parameters as shown below.

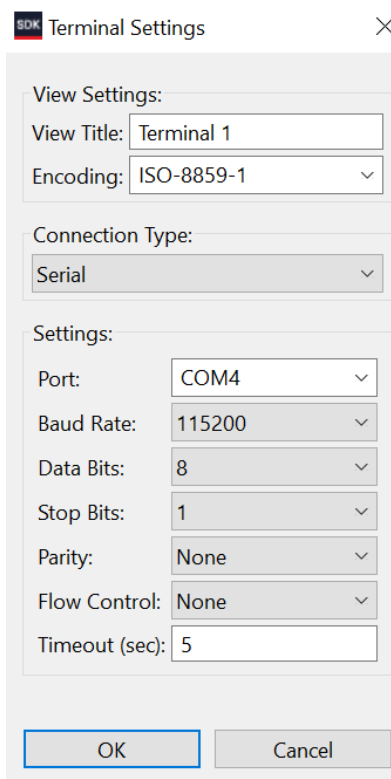


Figure 19. SDK Terminal Settings

6-1-5. Select **Xilinx > Program FPGA** and then click the **Program** button.

6-1-6. Select the **lab1** project in the *Project Explorer*, right-click and select **Run As > Launch on Hardware(GDB)** to download the application, execute ps7\_init, and execute lab1.elf.

6-1-7. You should see the following output on the Terminal console.

```
-- Start of the Program --  
-- Press BTNR (Zedboard) to see the LED light  
-- Change slide switches to see corresponding output on LEDs  
-- Set slide switches to 0xEE to exit the program
```

**Figure 20. SDK Terminal Output**

**6-1-8.** Press the BTNR (Zedboard) and see the LED light up.

**6-1-9.** Change the slide switches and see the corresponding LED turning ON and OFF.

**6-1-10.** Set slide switches to 0xEE to exit the program.

You should see the following output on the Terminal console.

```
-- Start of the Program --  
-- Press BTNR (Zedboard) to see the LED light  
-- Change slide switches to see corresponding output on LEDs  
-- Set slide switches to 0xEE to exit the program  
-- End of Program --
```

---

## Independent work

## Step 7

---

### 7-1. Changing the program. Verify the design functionality.

7-1-1. In the current program the PS LED (green LED) displays the status of the BTNR button.

7-1-2. Change the program so that:

PS LED (green LED) displays the status of the switch [0]

LED[0] displays the status of the BTNR button.

7-1-3. Compile program, load it to the board and check the functionality.

7-1-4. Set slide switches to 0xEE to exit the program.

7-1-5. Close SDK and Vivado programs by selecting **File > Exit** in each program.

7-1-6. Turn OFF the power to the board.

## Conclusion

In this lab, you created an ARM Cortex-A9 processor based embedded system using the Zynq device for the 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 to provide input and output functionality. You also saw that whenever the dedicated pins are not used, you need to provide pin constraints through the user constraints file (xdc).

You created the project in Vivado, created the hardware system using IPI, implemented the design in Vivado, exported the generated bitstream to the SDK, created a software application in the SDK, and verified the functionality in hardware after programming the PL section and running the application from the DDR memory.