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 either the PYNQ-Z1 or PYNQ-Z2 boards. 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 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 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 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 three standard GPIO IPs to connect to on-board LEDs, push-buttons, and switches. The following block diagram represents the completed design (**Figure 1**).

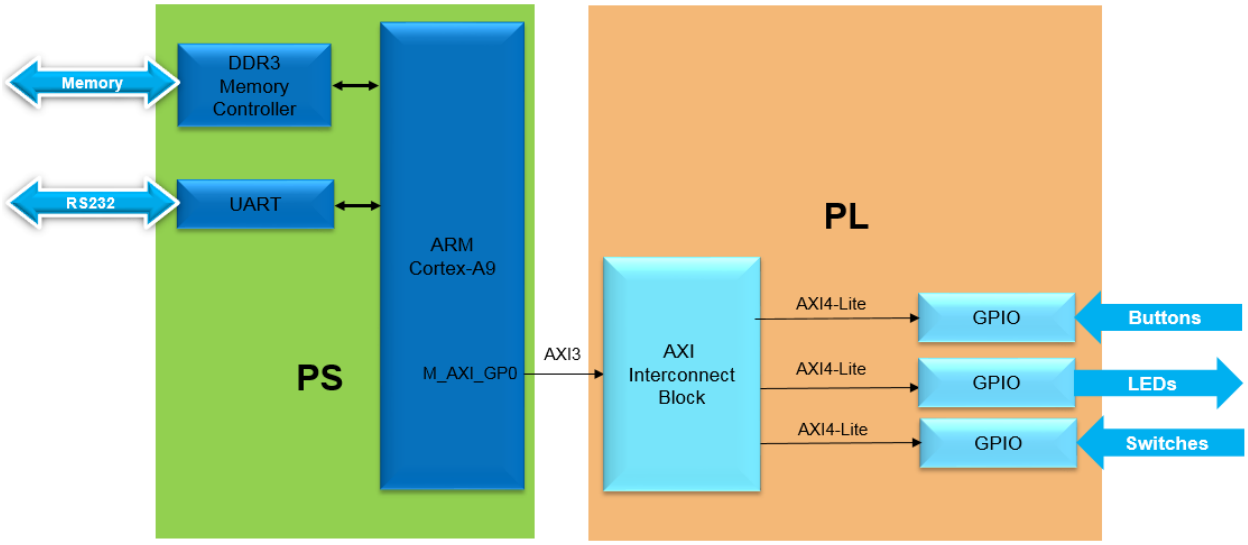
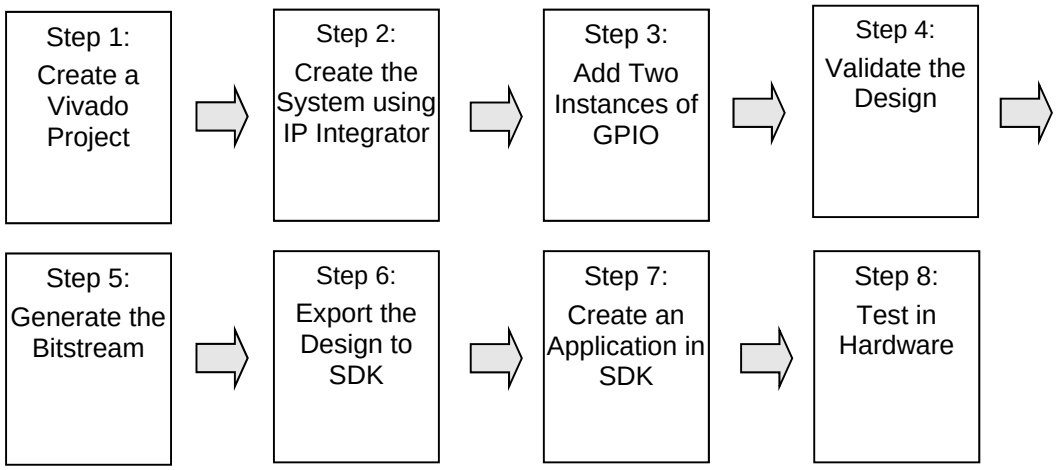


Figure 1. Completed Design

General Flow for this Lab



In the instructions below;
{***sources***} refers to: C:\xup\adv_embedded\2018_2_zynq_sources
{***labs***} refers to: C:\xup\adv_embedded\2018_2_zynq_labs

Board support for PYNQ-Z1 and PYNQ-Z2 are not included in Vivado 2018.2 by default. The relevant zip file need to be extracted and saved to: {Vivado installation}\data\boards\board_files\.

These files can be downloaded from the XUP webpage (http://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-adv-embedded-design-zynq.html) where this material is also hosted.

Create a Vivado Project — Step 1

1-1. Launch Vivado and create an empty project targeting the PYNQ-Z1 or PYNQ-Z2 board, selecting Verilog as a target language.

1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2018.2 > Vivado 2018.2**

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 **{labs}**, 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**.

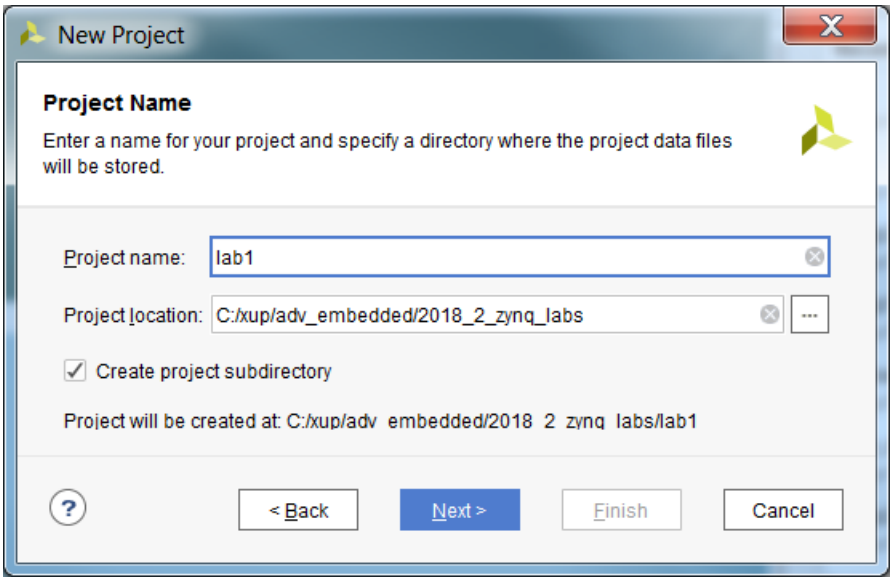


Figure 2. Project Name Entry

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 *Simulation Language* in the *Add Sources* form, and click **Next**.

1-1-7. Click **Next** to skip adding constraints.

1-1-8. In the *Default Part* form, click **Boards** filter.

1-1-9. Select **www.digilentinc.com** for the *PYNQ-Z1* board, **tul.com.tw** for the *PYNQ-Z2* board in the Vendor field, select *PYNQ-Z1 or pynq-z2*, and click **Next**.

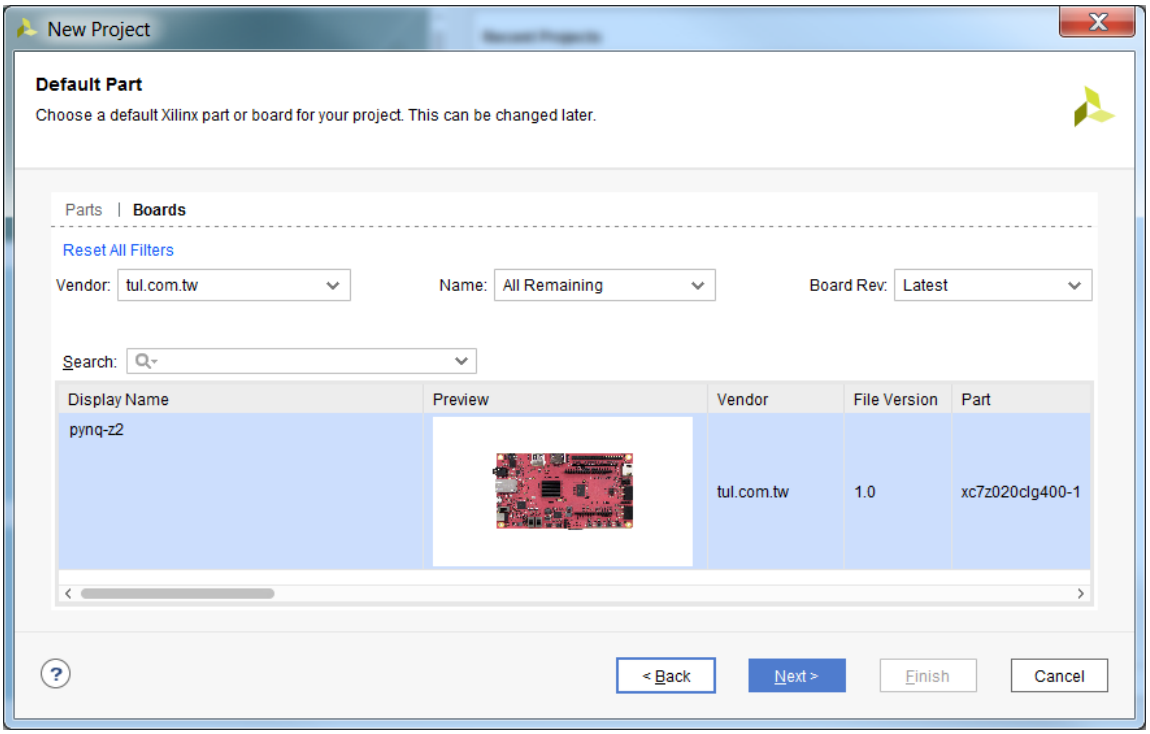


Figure 3. Board Selection (pynq-z2)

1-1-10. 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 **system** and click **OK**.

2-1-3. 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** and click **OK** to automatically configure the board presets.

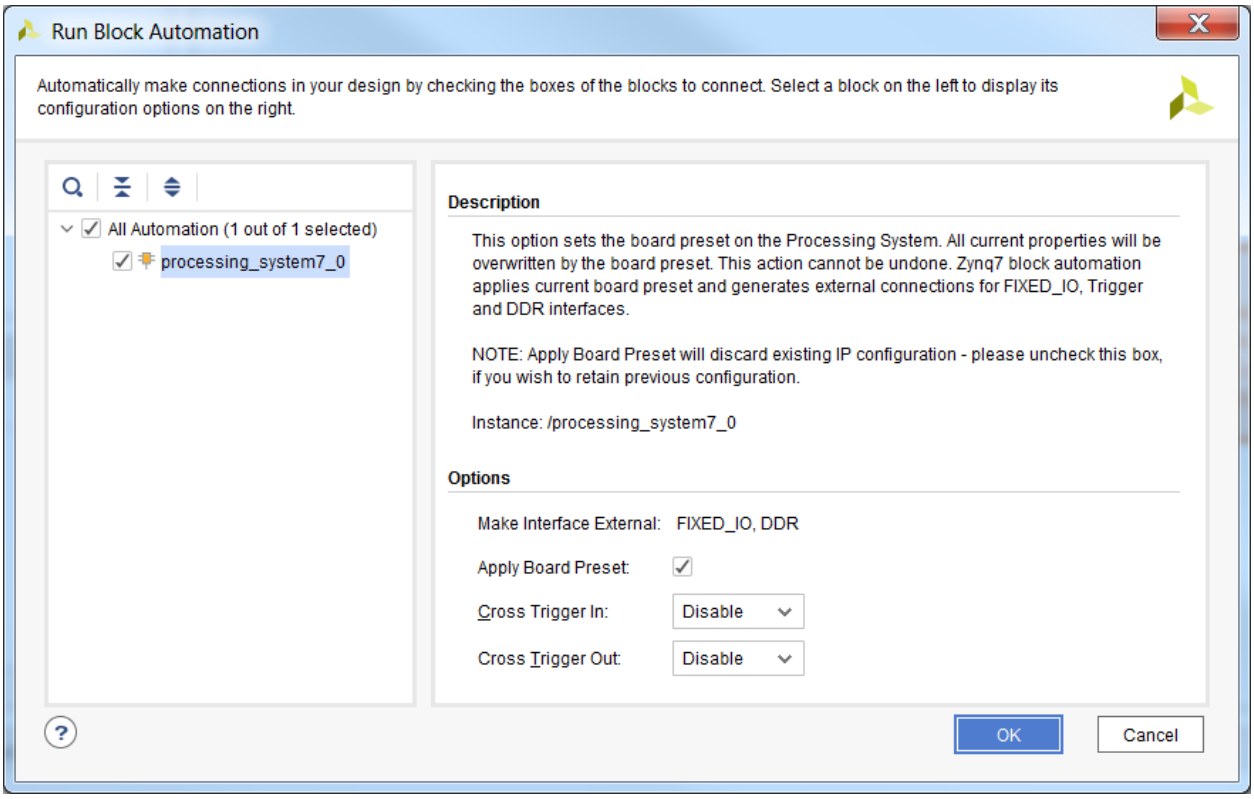


Figure 4. Zynq Block Automation View (pynq-z2)

2-1-6. Double click on the Zynq 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.

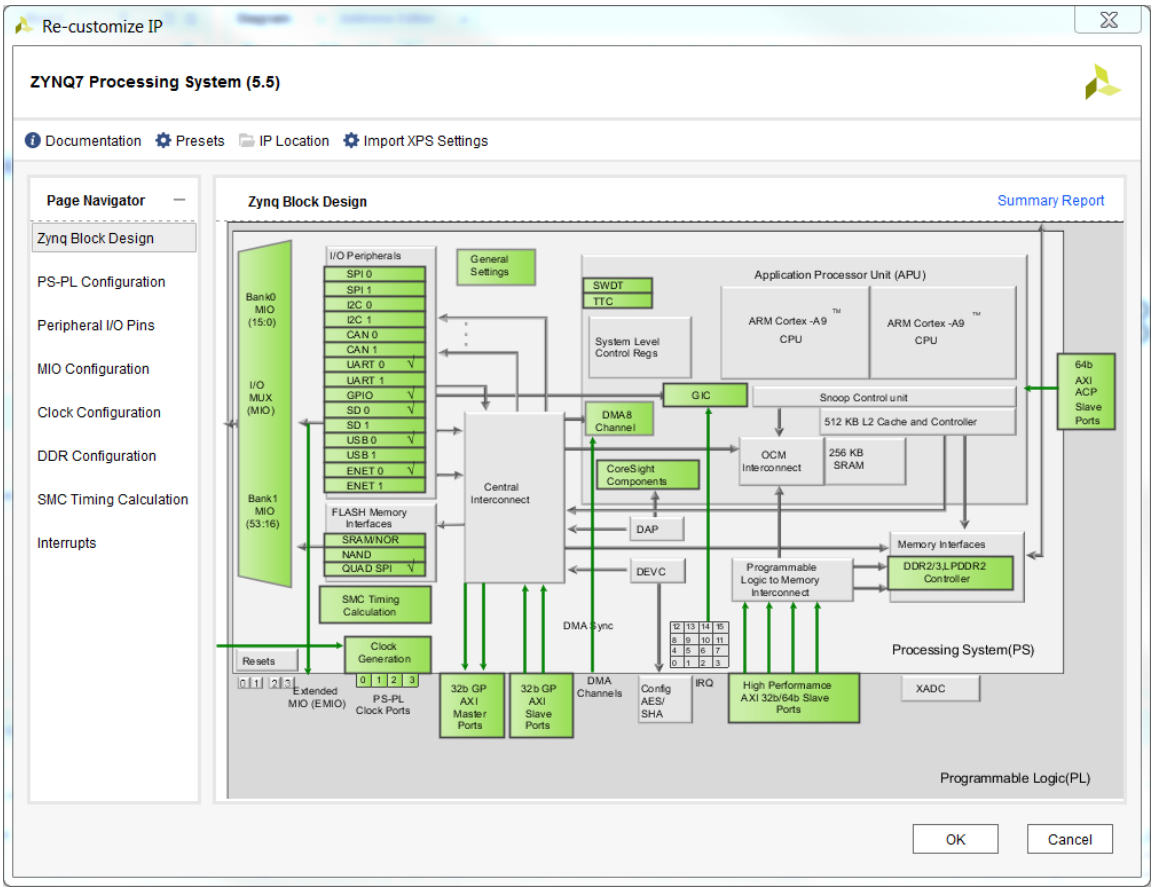


Figure 5. Zynq Processing System Configuration View (pynq-z2)

2-2. Configure the I/O Peripherals block to only have UART 0 support.

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

2-2-2. Expand the I/O Peripherals (and GPIO).

2-2-3. Deselect all the peripherals except *UART 0* (Deselect ENET 0, USB 0, SD 0, and GPIO).

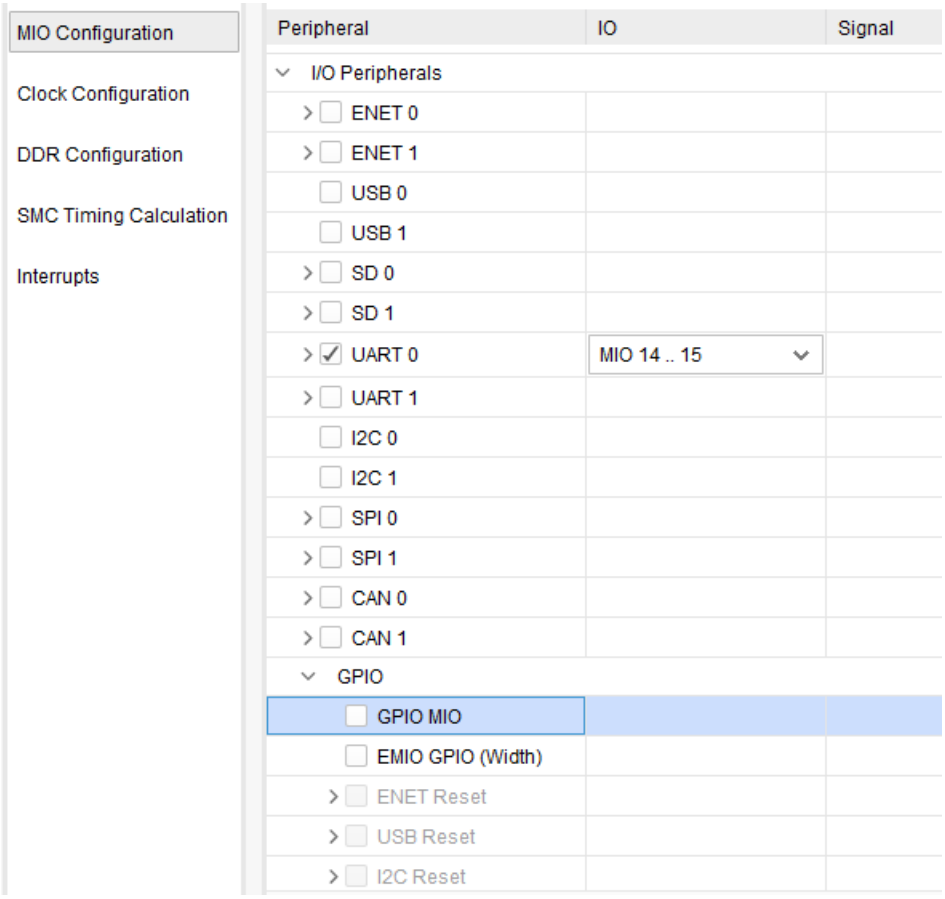


Figure 6. Selecting only UART 0 Peripheral of PS

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

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

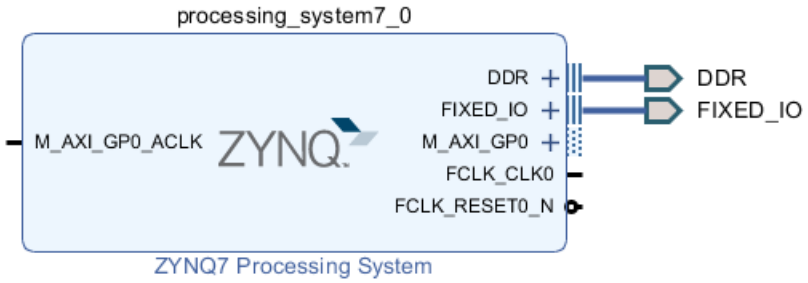


Figure 7. ZYNQ7 Processing System configured block

2-3. Add one instance of GPIO, name it *buttons*, and configure for the board. Connect the block to the Zynq.

2-3-1. Click the + button and search for **AXI GPIO** in the catalog.

2-3-2. Double-click the **AXI GPIO** to add an instance of the core to the design.

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

2-3-4. Double click on the **AXI GPIO** block to open the customization window. Under *Board Interface*, for *GPIO*, click on **Custom** to view the dropdown menu options, and select **btns 4Bits** for the PYNQ-Z2 or the PYNQ-Z1 board.

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

2-3-5. Click the **IP Configuration** tab. Notice the GPIO Width is set to 4 (PYNQ-Z1 and PYNQ-Z2) and is greyed out. If a board support package was not available, the width of the IP could be configured here.

2-3-6. Click **OK** to finish configuring the GPIO and to close the *Re-Customize IP* window.

2-3-7. Click on **Run Connection Automation**, and select **buttons** (which will include GPIO and S_AXI)

Click on GPIO and S_AXI to check the default connections for these interfaces.

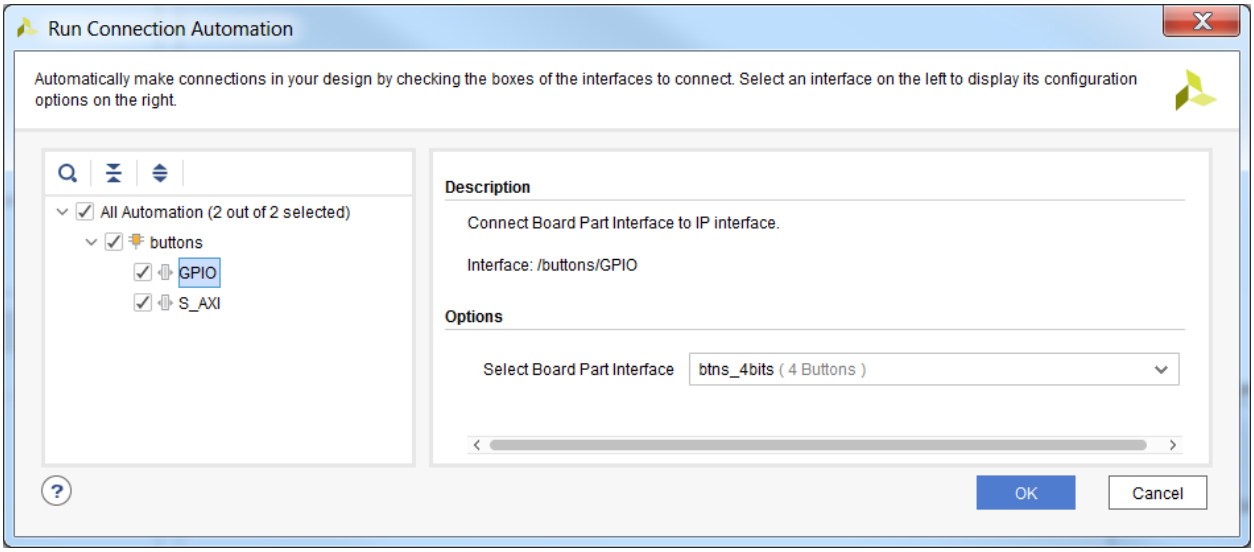


Figure 8. Connection Automation for the GPIO (PYNQ-Z2)

2-3-8. Click **OK** to automatically connect the *S_AXI* interface to the Zynq *GP0* port (through the AXI interconnect block), and the GPIO port to an external interface.

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

2-4. Add another instance of GPIO, name the instance *leds*, configure it and connect it to the Zynq.

2-4-1. Add another instance of the *GPIO* peripheral**.**

2-4-2. Change the name of the block to **leds.**

2-4-3. Double click on the *leds* block, and select **leds 4bits** (PYNQ-Z1 and PYNQ-Z2) for the *GPIO* interface and click **OK.**

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

2-4-5. Click **leds**, and check the connections for *GPIO* and *S_AXI* as before

2-4-6. 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.

2-5. Add another instance of GPIO, name the instance *switches*, configure it and connect it to the Zynq.

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

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

2-5-3. Double click on the *switches* block, and select **sws 2bits** (PYNQ-Z1 and PYNQ-Z2) for the *GPIO* interface and click **OK.**

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

2-5-5. Click **switches**, 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 third master AXI (M02_AXI) port added and connected to the S_AXI of the leds.

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

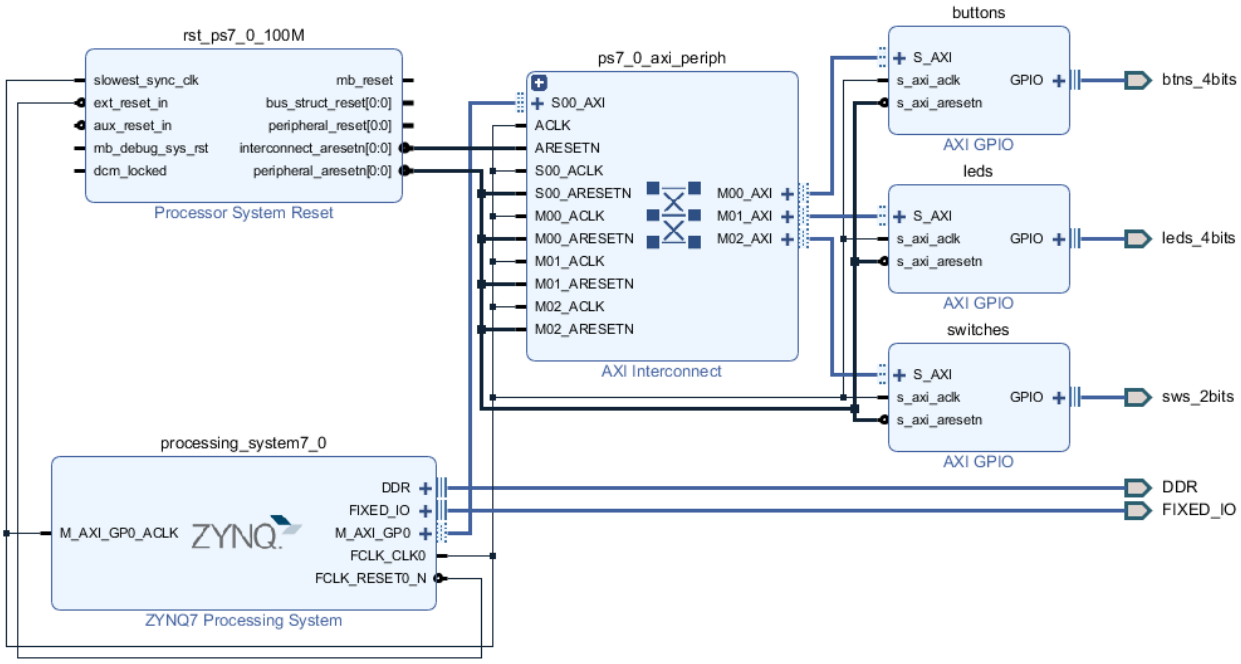


Figure 9. Completed design

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 three GPIO instances. They should look like as follows.

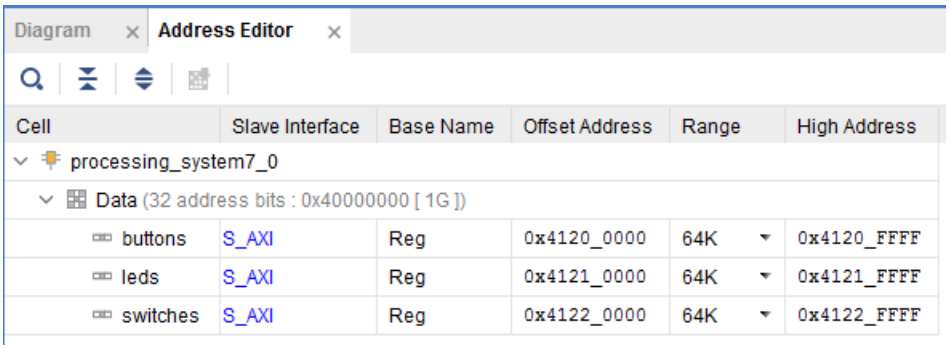


Figure 7. Assigned addresses

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.

2-6-2. Select the *Diagram* tab, and click on the (Validate Design) button to make sure that there are no errors.

Ignore warnings.

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

2-6-4. Since all IO pins are board-aware no additional user constraints are need.

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 11. 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. The wrapper file will be displayed in the Auxiliary pane.

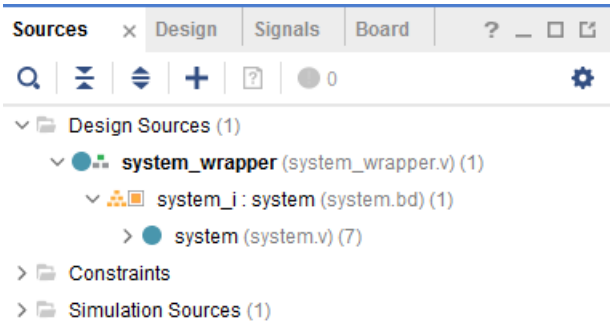


Figure 82. Design Hierarchy View

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 and click **OK**.

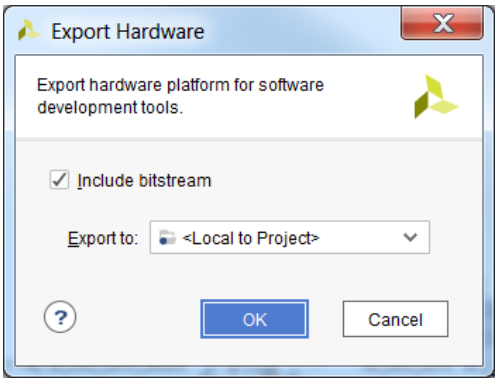


Figure 9. Exporting the hardware

4-1-2. 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.)

Generate an Application in SDK Step 5

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

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.

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

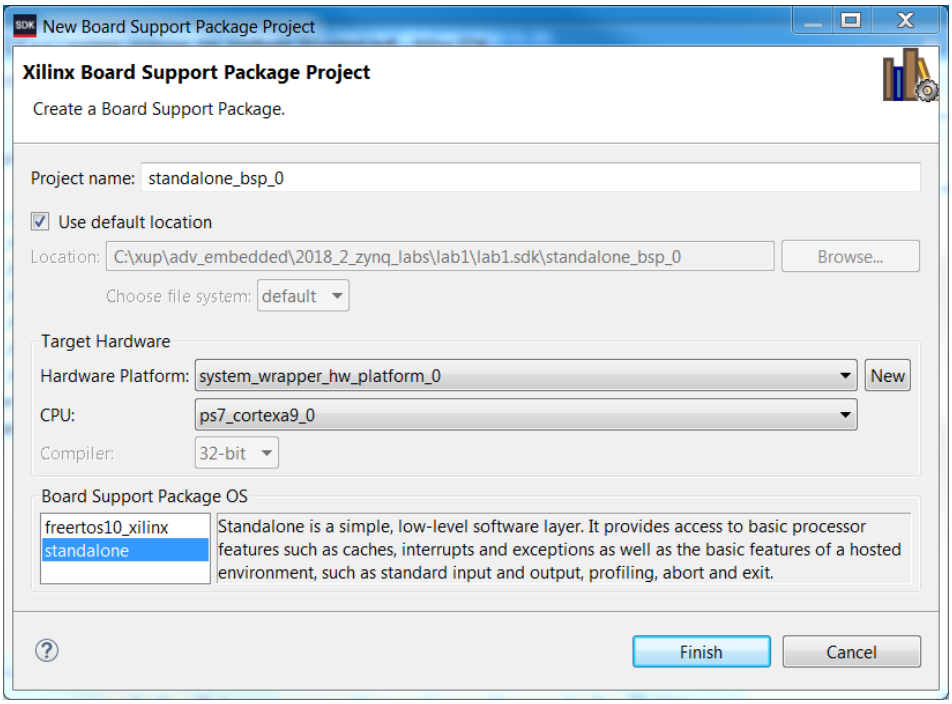


Figure 104. Create BSP

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 and then click **Next.**

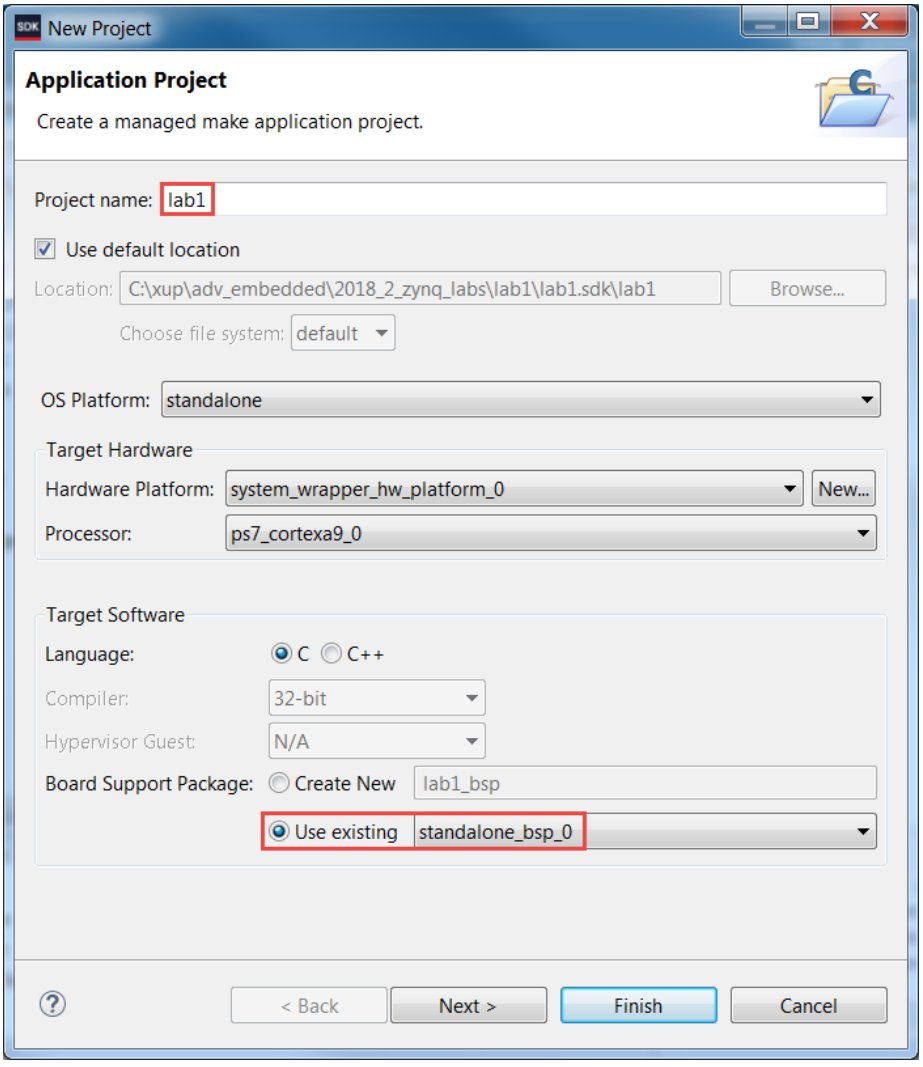


Figure 11. Create a Blank Application Project

5-2-4. Select the **Empty Application** template and click **Finish.**

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

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

5-2-6. Expand the **General** category and double-click on **File System.**

5-2-7. Browse to the **{sources}\lab1** folder.

5-2-8. Select the **lab1.c** source file and click **Finish.**

A snippet of the source code is shown in the following figure. Note the greyed out code will be used in Lab5. The code reads from the switches, and writes to the LEDs. The BTN is read, and written to the LED.

```
#include "xparameters.h"
#include "xgpio.h"
#ifdef MULTIBOOT
#include "xdevcfg.h"
#endif

//----------------------------------------------------
int main (void)
{

      XGpio sws, leds, btns;
      int i, sws_check, btns_check;

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

      // AXI GPIO switches Initialization
      XGpio_Initialize(&sws, XPAR_SWITCHES_DEVICE_ID);
      XGpio_SetDataDirection(&sws, 1, 0xffffffff);      // input
      // AXI GPIO leds Initialization
      XGpio_Initialize(&leds, XPAR_LEDS_DEVICE_ID);
      XGpio_SetDataDirection(&leds, 1, 0);              // output
      // AXI GPIO buttons Initialization
      XGpio_Initialize(&btns, XPAR_BUTTONS_DEVICE_ID);
      XGpio_SetDataDirection(&btns, 1, 0xffffffff);     // input

      xil_printf("-- Press any of BTN0-BTN3 to see corresponding output on LEDs --\r\n");
      xil_printf("-- Set slide switches to 0x03 to exit the program --\r\n");

      while (1)
      {
          btns_check = XGpio_DiscreteRead(&btns, 1);
          XGpio_DiscreteWrite(&leds, 1, btns_check);
          sws_check = XGpio_DiscreteRead(&sws,1);
          if((sws_check & 0x03)==0x03)
              break;
          for (i=0; i<9999999; i++); // delay loop
       }
      xil_printf("-- End of Program --\r\n");

#ifdef MULTIBOOT
      // Driver Instantiations
      XDcfg XDcfg_0;
      u32 MultiBootReg = 0;
      #define PS_RST_CTRL_REG   (XPS_SYS_CTRL_BASEADDR + 0x200)
      #define PS_RST_MASK   0x1 /* PS software reset */
      #define SLCR_UNLOCK_OFFSET 0x08

      // Initialize Device Configuration Interface
      XDcfg_Config *Config = XDcfg_LookupConfig(XPAR_XDCFG_0_DEVICE_ID);
      XDcfg_CfgInitialize(&XDcfg_0, Config, Config->BaseAddr);

      MultiBootReg = 0; // Once done, boot the master image stored at 0xfc00_0000
      Xil_Out32(0xF8000000 + SLCR_UNLOCK_OFFSET, 0xDF0DDF0D); // unlock SLCR
      XDcfg_WriteReg(XDcfg_0.Config.BaseAddr, XDCFG_MULTIBOOT_ADDR_OFFSET, MultiBootReg); // write to multiboot
      // synchronize
        __asm__(
            "dsb\n\t"
            "isb"
        );
      Xil_Out32(PS_RST_CTRL_REG, PS_RST_MASK);
#endif
      return 0;
}
```

Figure 16. Snippet of Source Code

Test in Hardware — Step 8

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.

6-1-2. Select the Terminal tab. If it is not visible then select **Window > Show view > Other > Terminal > Terminal**.

6-1-3. Click on and select appropriate COM port (depending on your computer), and configure the terminal with the parameters as shown below.

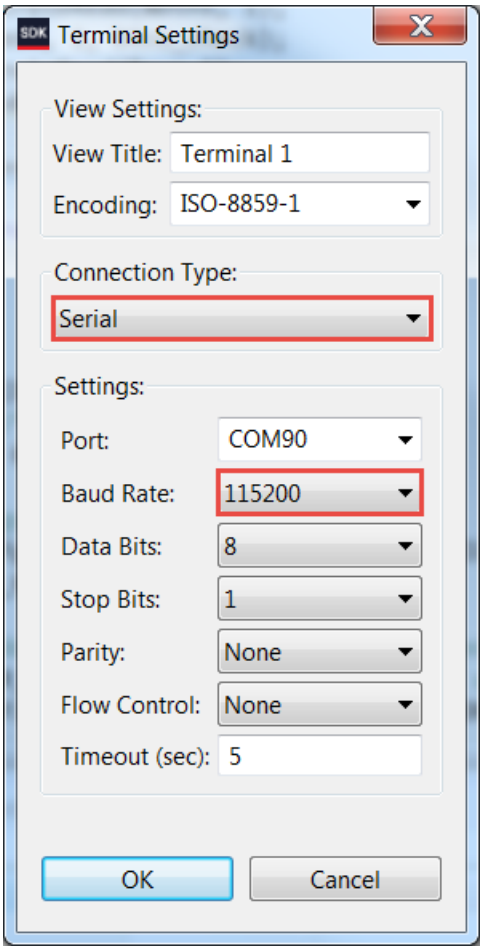


Figure 17. SDK Terminal Settings

6-1-4. Select **Xilinx** > **Program FPGA** and then click the **Program** button.

6-1-5. Make sure that the SW0-1 are not set to "11".

6-1-6. Select the **lab1** project in the *Project Explorer*, right-click and select **Run As > Launch on Hardware(System Debugger)** to download the application, execute ps7_init, and execute lab1.elf.

www.xilinx.com/university
xup@xilinx.com

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

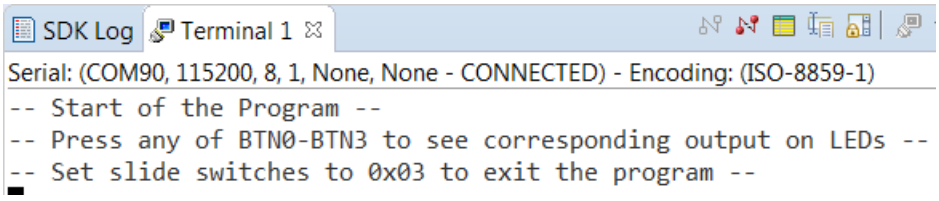


Figure 18. SDK Terminal Output

6-1-8. Press the BTN0-BTN3 (PYNQ-Z1, PYNQ-Z2) and see the corresponding LED light up.

6-1-9. Set the two slide switches on PYNQ-Z1 or PYNQ-Z2 to the ON position to exit the program.

6-1-10. Close SDK and Vivado programs by selecting **File > Exit** in each program.

6-1-11. 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 PYNQ-Z1/PYNQ-Z2 board. You instantiated the Xilinx standard GPIO IP to provide input and output functionality.

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.