- Subject : Lab2 Adding IP cores in PL
- Created date : 2017. 06. 01
- Recently modified date : -
- Creator : Choi Jun-Ho
- Contact : peluza6332@gmail.com
- Worked dates to make working of subject : 2017. 06. 01
- Documentation status : finished
- Summary
  : Adding IP cores in PL 개인 한글화 문서

- Contents

- Introduction

This lab guides you through the process of extending the processing system you created in the previous lab by adding two GPIO (General Purpose Input/Output) IPs

- Objectives

After completing this lab, you will be able to:

• Configure the GP Master port of the PS to connect to IP in the PL

이 랩에서 PS와 PL 사이의 AXI GP Master port를 연결한다.

• Add additional IP to a hardware design
• Setup some of the compiler settings

이건 뭐지?! 뭔가 중요한 것 일수도

- 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 6 primary steps:
You will open the project in Vivado, add and configure GPIO peripherals in the system using IP Integrator, connect external ports, generate bitstream and export to SDK, create a TestApp application in SDK, and, finally, verify the design in hardware.

드디어 PS와 PL을 연결하기 위한 ports를 만든다.
그리고 PL을 구현했으므로 PL을 제어하기 위한 bitstream을 만든다.

- Design Description

The purpose of this lab exercise is **to extend the hardware design (Figure 1) created in Lab 1**

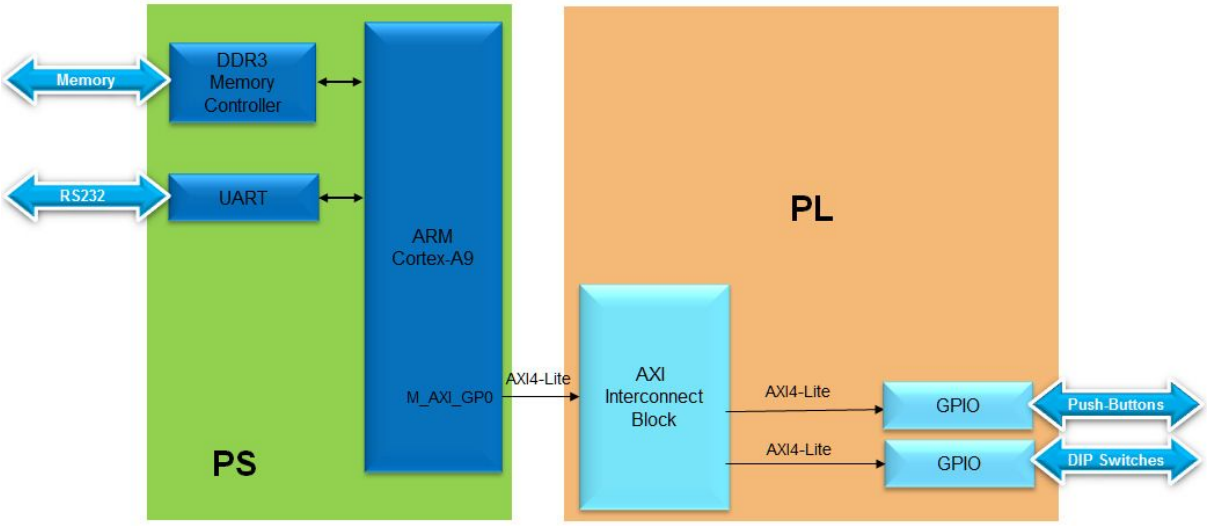
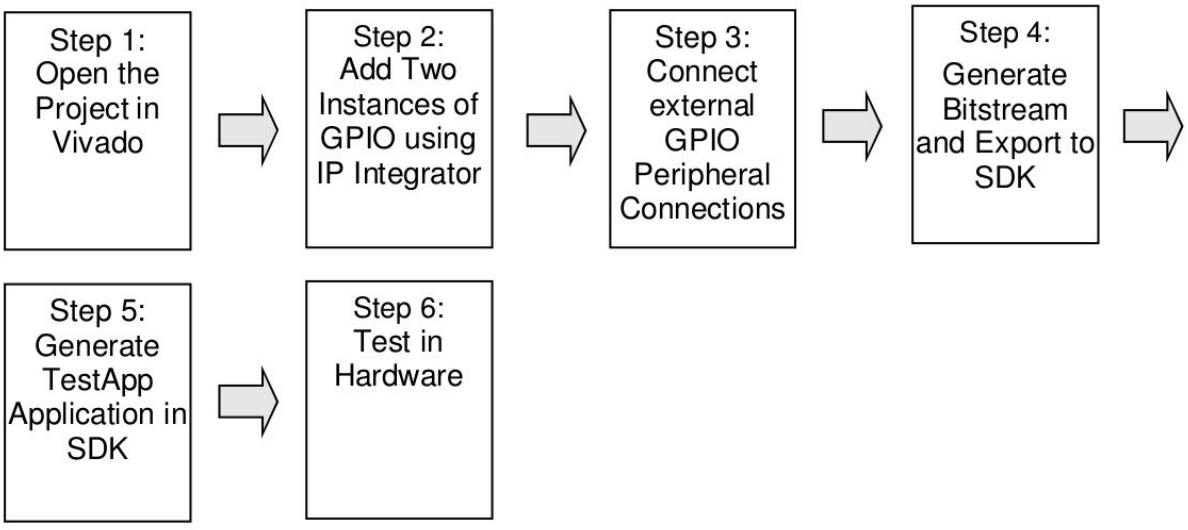



Figure 1. Extend the System from the Previous Lab

- General Flow for this Lab




이번에는 GPIO 인스턴스 2개를 IPI를 이용해 추가하고, 외부 GPIO 페리페럴 연결부와 연결한다.
Bitstream을 생성하고 SDK로 출력한다.

In the instructions below;
{sources} refers to: C:\xup\embedded\2015_2_zynq_sources
{labs} refers to : C:\xup\embedded\2015_2_zynq_labs
or for the Zybo refers to:

C:\xup\embedded\2015_2_zybo_labsolution

- 1. Open the Project

**Open the previous project, or the lab1 project from the {labsolutions} directory, and save the project as lab2. Open the Block Design.**

- 2. Add Two Instances of GPIO

In the Sources panel, expand system_wrapper, and double-click on the system.bd (system_i) file to invoke IP Integrator. (The Block Design can also be opened from the Flow Navigator)

Double click on the Zynq block in the diagram to open the Zynq configuration window.

한 마디로 이걸 띄워라

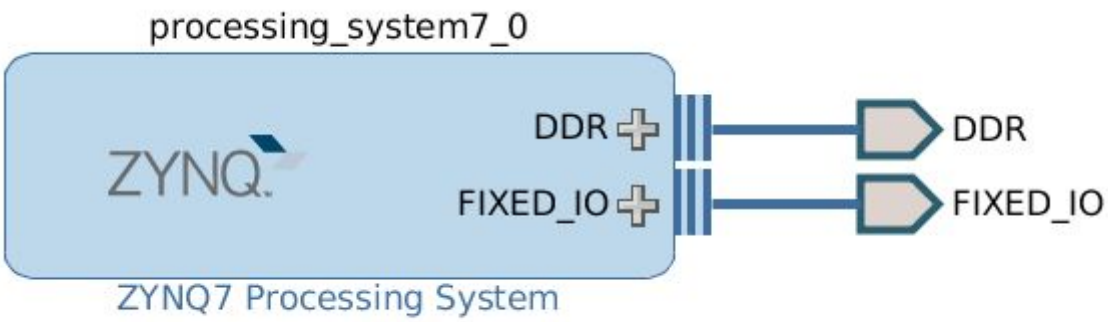



Figure 4. Zynq system with AXI and clock interfaces

그리고 연다.

Select PS-PL Configuration page menu on the left, or click 32b GP AXI Master Ports block in the Zynq Block Design view.

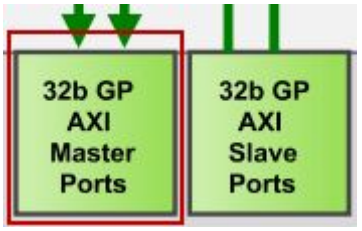



Figure 2. AXI Port Configuration

Expand AXI Non Secure Enablement > GP Master AXI Interfaces, if necessary, and click on Enable M_AXI_GP0 interface check box under the field to enable the AXI GP0 port.

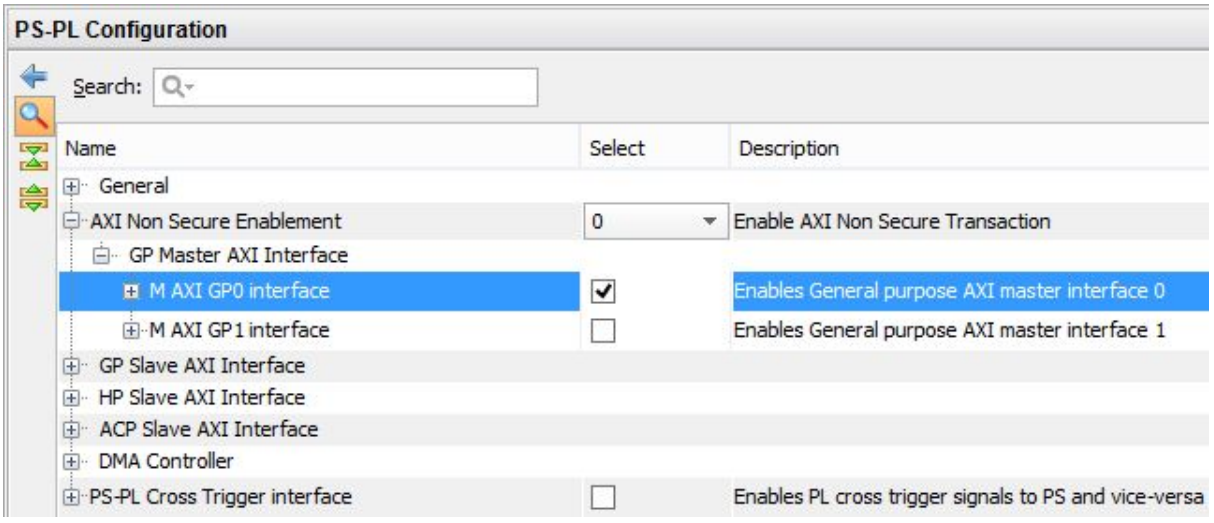
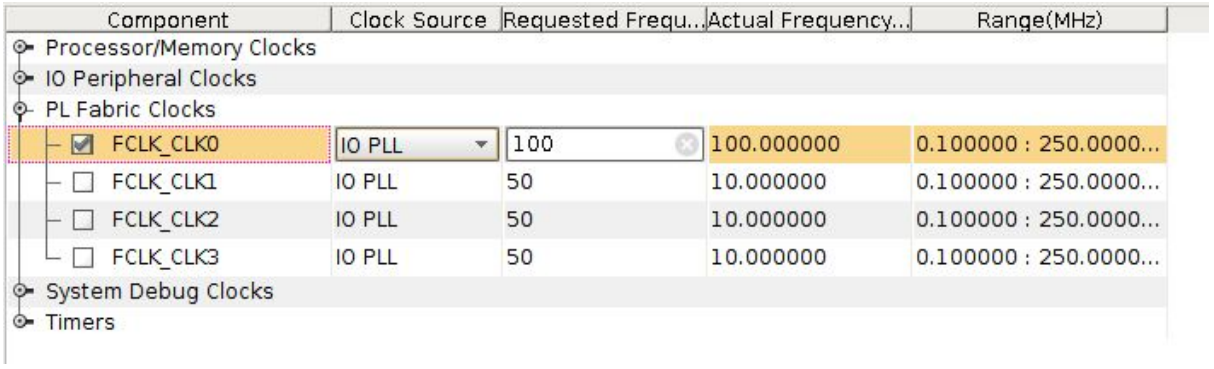


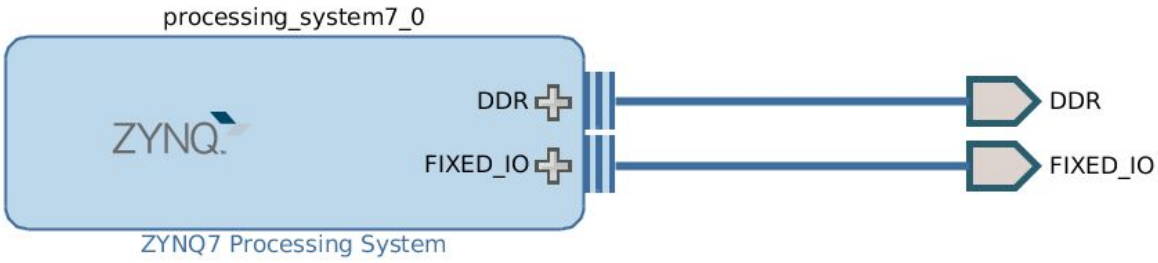

Figure 3. Configuration of 32b Master GP Block

Expand **General > Enable Clock Resets** and select the **FCLK_RESET0_N** option.

Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and select the **FCLK_CLK0** option (with requested clock frequency of 100.000000 MHz) and click **OK**.




Notice the additional **M_AXI_GPO interface**, and M_AXI_GPO_ACLK, FCLK_CLK0, and FCLK_RESET0_N ports are now included on the Zynq block. You can click the regenerate button ( ) to redraw the diagram.




에서

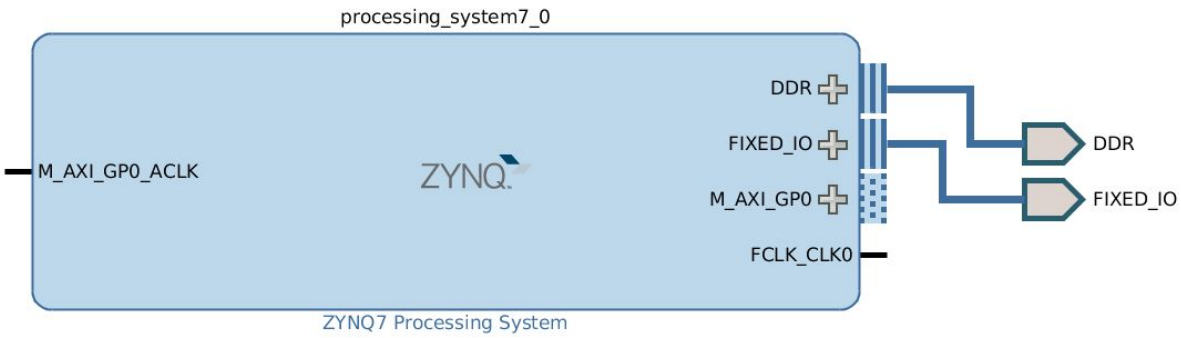
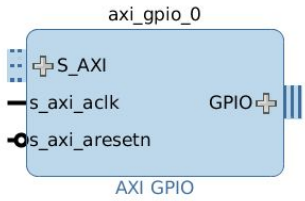


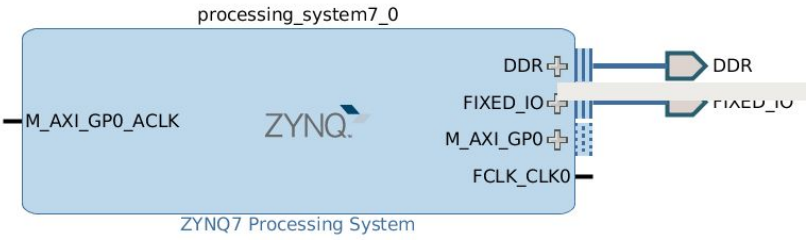

Figure 4. Zynq system with AXI and clock interfaces

으로

Click the Add IP icon and search for AXI GPIO in the catalog

Double-click the AXI GPIO to add the core to the design. The core will be added to the design and the block diagram will be updated.




Click on the AXI GPIO block to select it, and in the properties tab, change the name to switches

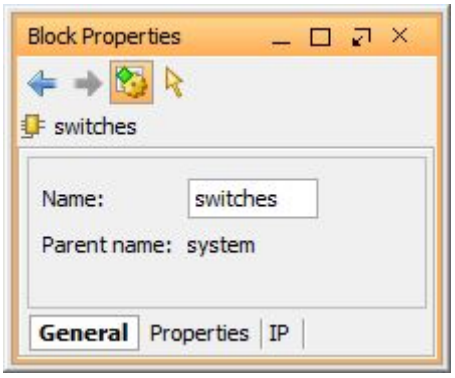



Figure 7. Change AXI GPIO default name

Double click on the AXI GPIO block to open the customization window.

From the Board Interface drop down, select sws 8bits for ZedBoard or sws 4bits for Zybo for GPIO.

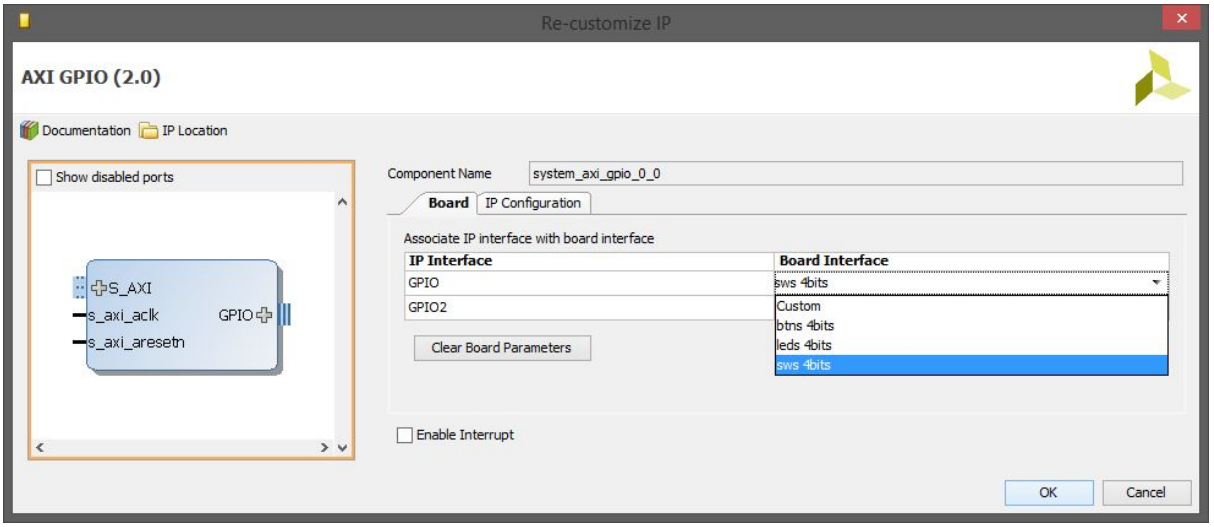



Figure 8. Configuring GPIO instance

**Click the IP configuration tab**, and notice the width has already been set to match the switches on the ZedBoard (8) or Zybo (4)

Notice that the peripheral can be configured for two channels, but, since **we want to use only one channel without interrupt**, leave the Enable Interrupt and Enable Dual Channel **unchecked**.

Click OK to save and close the customization window

Notice that **Designer assistance is available**. Click on **Run Connection Automation**, and **select /switches/S_AXI**

/switches/S_AXI가 뭐지? source 탭에서 말인가?

Click OK **when prompted to automatically connect the master and slave interfaces**

/switches/S_AXI라는 건 Run Connection Automation할 때 자동으로 붙일 것을 설정할 때 /switches 아래의 /S_AXI를 말하는 것.

switches는 아마도 GPIO IP의 이름을 저것으로 바꿔서 그런 것 같고(따라서 switches는 GPIO IP이다.)
이 GPIO Block 안에 GPIO 핀과 PS와 통신할 수 있는 S_AXI가 있다.
(아! S_AXI가 Slave AXI인 것 같고 따라서 Lab1에서 본 M_AXI는 Master AXI인듯.)

Notice two additional blocks, **Processor System Reset, and AXI Interconnect have automatically been added to the design**. (The blocks can be dragged to be rearranged, or the design can be redrawn.)

드디어 PS Reset과 AXI Interconnect가 자동으로 추가됨을 보긴 보았다.
이건 교재로 실습할 때도 본 것이긴 한데 어떤 역할 때문에 추가된 것인지는 정확하지 않다.
다만 추측할 수 있는 것은 AXI Interconnect는 여러 AXI 연결에 들을 컨트롤 해주는 것이므로 해당 기능 사용을 위해 자동 추가된 것 같다.

PS Reset은 왜 추가된 건지 잘 모르겠다. 어떤 이유가 있겠지.

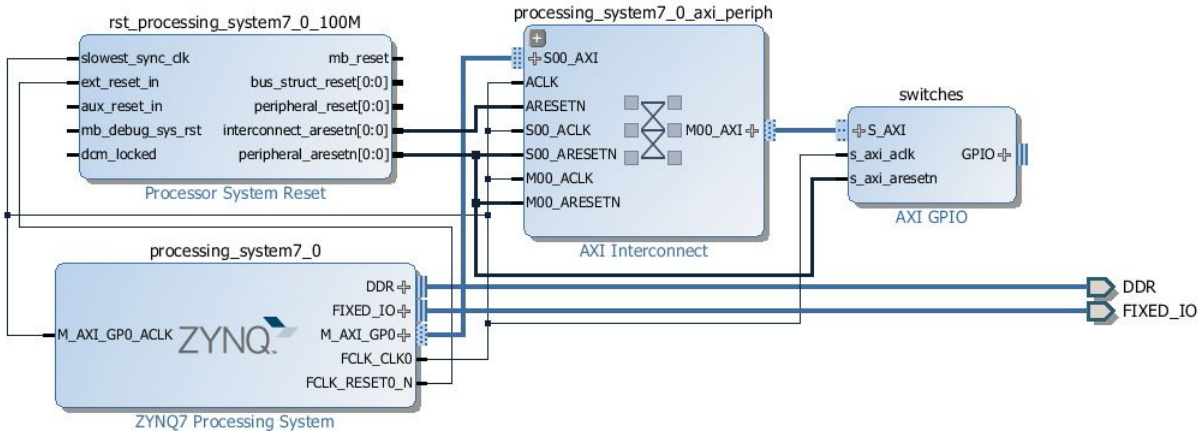


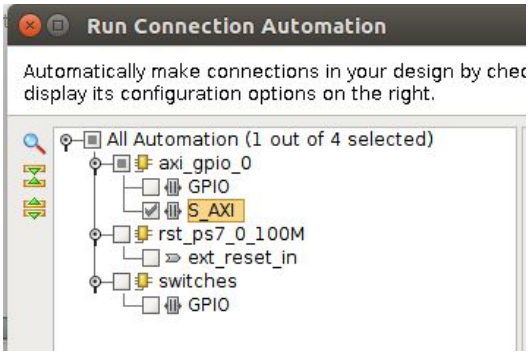

Figure 11. Design with switches automatically connected

흠… PS Reset Block에도 위 포트들 말고도 다른 많은 것들이 더 달릴 수 있다면,
아마도 이 Block의 역할은 초기 상태로 돌려주는 것을 말하는 것 같다.
그러니까, 내가 지금 추가한 GPIO 페리페럴과 AXI 버스, 그리고 자동 추가된 interconnect에 대한 reset도 보인다.

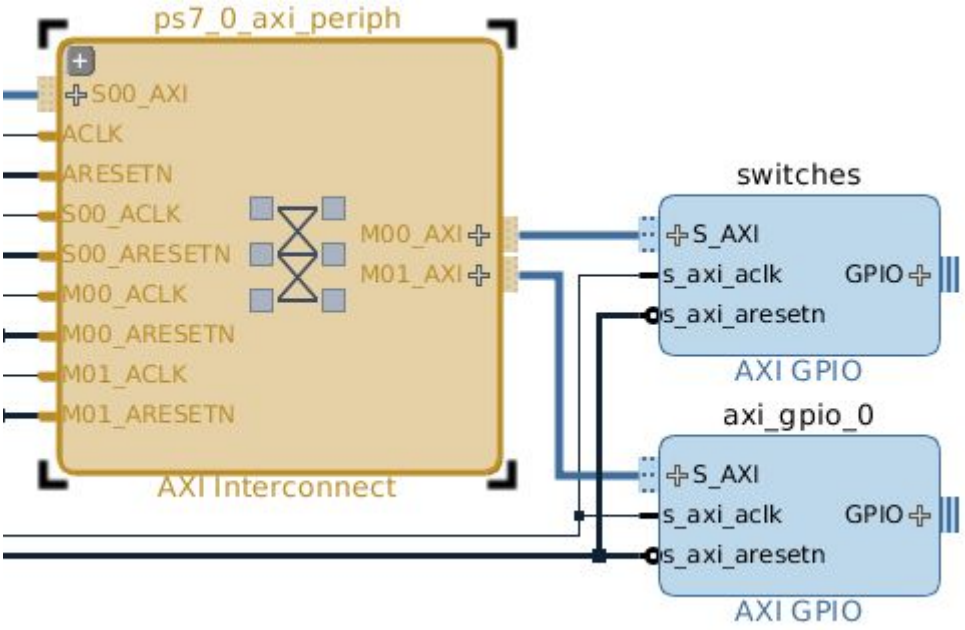
Add another instance of the GPIO peripheral (Add IP). Name it as buttons

Double click on the IP block, select the btns GPIO interface (~~btns_5bits for the Zedboard~~, btns_4bits for the Zybo) and click OK.

At this point connection automation could be run, or the block could be connected manually. **This time the block will be connected manually**. 꺅



Double click on the **AXI Interconnect** and change the Number of Master Interfaces to **2** and click **OK**




이건 잘못한 것이다.

위에서 Designer Assistance의 Connect Automation 눌러서 BTN GPIO의 S_AXI도 눌러줬기 때문에 문제가 있다.
따라서 뒤로 가서 다시 했다.


에서


으로

Click on the s_axi port of the buttons AXI GPIO block, and drag the pointer towards the AXI Interconnect block.

The message Found 1 interface should appear, and a green tick should appear beside the M01_AXI port on the AXI Interconnect indicating this is a valid port to connect to.

Drag the pointer to this port and release the mouse button to make the connection.

In a similar way, connect the following ports:
buttons s_axi_aclk -> Zynq7 Processing System FCLK_CLK0
buttons s_axi_aresetn -> Processor System Reset peripheral_aresetn
AXI Interconnect M01_ACLK -> Zynq7 Processing System FCLK_CLK0
AXI Interconnect M01_ARESETN -> Processor System Reset peripheral_aresetn

The block diagram should look similar to this:

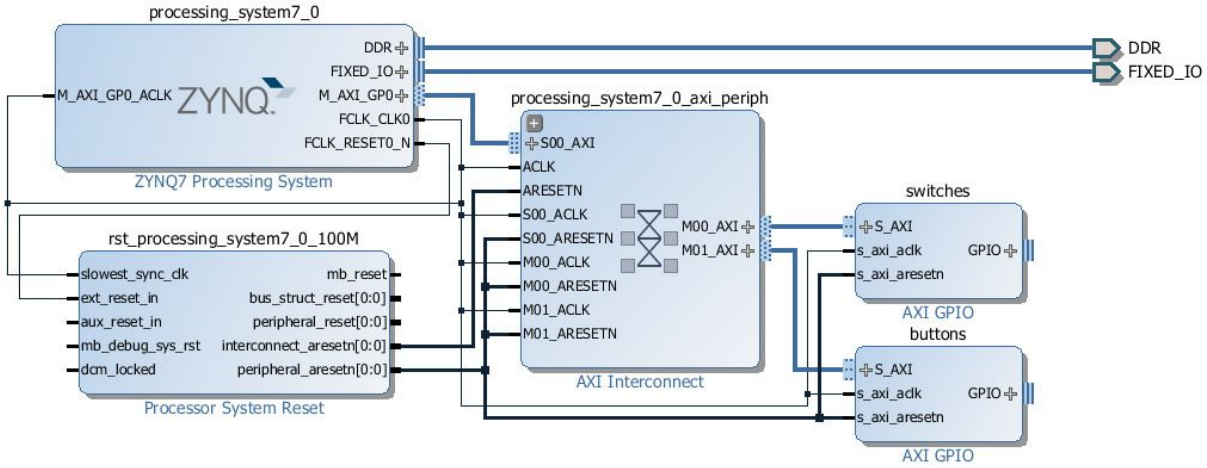
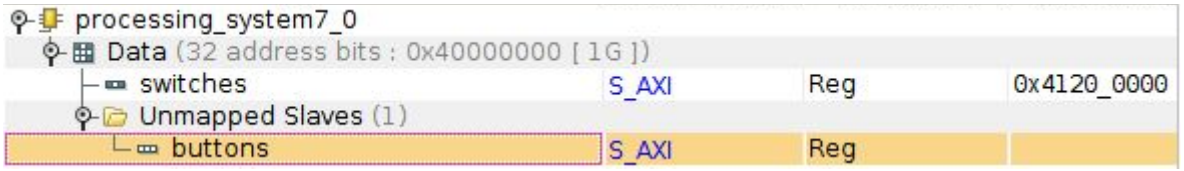



Figure 13. System Assembly View after Adding the Peripherals

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




buttons AXI GPIO는 수동으로 조작돼서 Unmapped Slaves가 있는듯.

Notice that switches has been automatically assigned an address, but buttons has not (since it was manually connected). **Right click on btns_4bit and select Assign Address or click on the button**.
맞네!

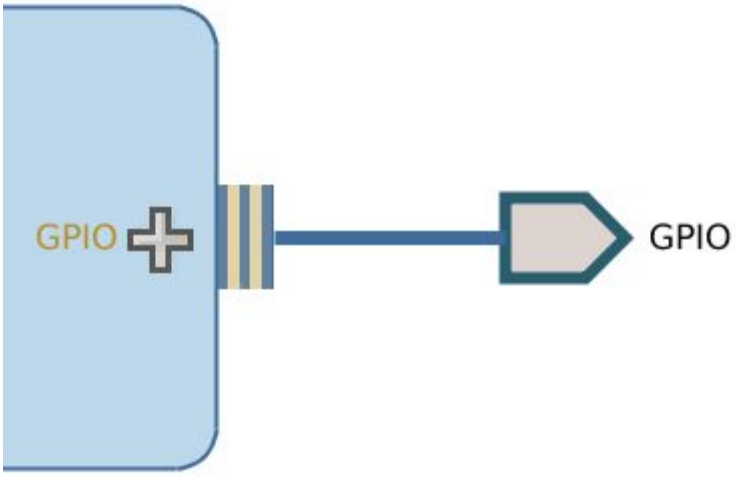
Note that both peripherals are assigned in the address range of 0x40000000 to 0x7FFFFFFF (GP0 range).
/// TODO
0x40000000 ~ 0x7FFFFFFF까지가 GP0 범위구나… Datasheet에서 확인해 볼 것

- 3. Make GPIO Peripheral Connections External

**The push button and dip switch instances will be connected to corresponding pins on the board. <u>This can be done manually, or using Designer Assistance</u>. Normally, one would consult the board's user manual to find this information.**
인스턴스는 그에 맞는 핀과 연결되는데 이에 대한 것은 해당 개발 보드의 사용자 메뉴얼을 참고하도록 한다.

In the Diagram view, notice that Designer Assistance is available. We will manually create the ports and connect.
디자이너 어시스턴스를 쓸 수도 있지만 여기서는 수동으로 포트를 만들고 연결하도록 한다.

Right-Click on the GPIO port of the switches instance and select Make External to create the external port. **This will create the external port named gpio and connect it to the peripheral**. Because Vivado is "board aware", the pin constraints will be automatically applied to the port.




외부 포트를 만들기 위한 Make External을 선택하면
GPIO라는 이름의 외부 포트가 만들어지고 자동으로 페리페럴에 연결된다.
이는 Vivado가 핀의 제약 사항에 대해 알고 있기 때문에 해당 포트에 자동으로 연결된 것이다.

Select the gpio port and change the name to switches in its properties form.


The width of the interface will be automatically determined by the upstream block.
인터페이스의 폭이 upstream block에 의해 자동으로 결정된다는데
이 인터페이스의 폭은 어디서 볼 수 있으며 upstream block은 또 뭔지 모르겠다.

For the buttons GPIO, click on the Run Connection Automation link.

In the opened GUI, ~~select btns_5bits (for ZedBoard) or~~ btns_4bits (for Zybo) under the options section.

Click OK.

Select the created external port and change its name as buttons
자동으로 생성됨을 나타내는 의미로 그대로 둔다.

Run Design Validation (Tools -> Validate Design) and verify there are no errors. The design should now look similar to the diagram below

**Synthesize the design, open the I/O Planning layout, and check the constraints using the I/O planning tool.**

In the Flow Navigator, click **Run Synthesis**. (Click Save if prompted) and when synthesis completes, select **Open Synthesized Design** and click OK In the shortcut Bar, **select I/O Planning** from the Layout dropdown menu

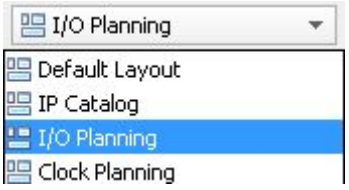
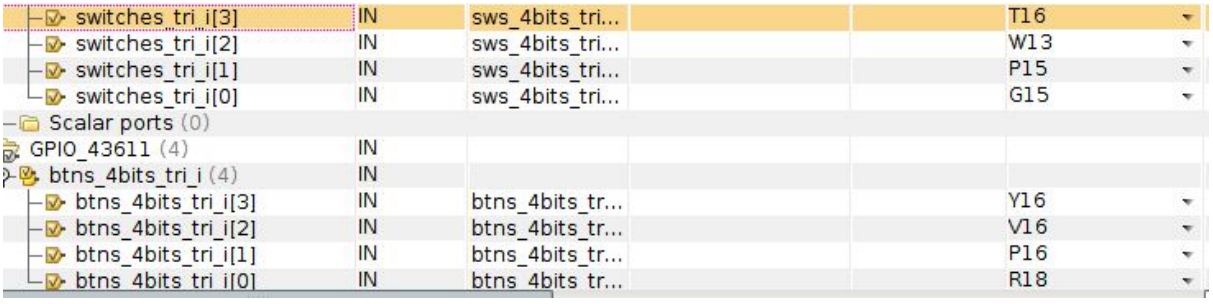



Figure 16. Switch to the IO planning view

In the I/O ports tab, **expand the two GPIO icons**, and **expand buttons_tri_i**, and **switches_tri_i**, and notice that **the ports have been automatically assigned pin locations**, along with **the other Fixed IO ports in the design**, and an I/O Std of ~~LVCMOS25 (for ZedBoard)~~ and LVCMOS33 (for Zybo) has been applied. <u>If they were not automatically applied</u>, **pin constraints can be included in a constraints file**, or **entered manually or modified through <u>the I/O Ports tab</u>**.



| | | | | | |
|---|---|---|---|---|---|
| switches_tri_i[3] | IN | sws_4bits_tri... | | | T16 |
| switches_tri_i[2] | IN | sws_4bits_tri... | | | W13 |
| switches_tri_i[1] | IN | sws_4bits_tri... | | | P15 |
| switches_tri_i[0] | IN | sws_4bits_tri... | | | G15 |
| Scalar ports (0) | | | | | |
| GPIO_43611 (4) | IN | | | | |
| btns_4bits_tri_i (4) | IN | | | | |
| btns_4bits_tri_i[3] | IN | btns_4bits_tr... | | | Y16 |
| btns_4bits_tri_i[2] | IN | btns_4bits_tr... | | | V16 |
| btns_4bits_tri_i[1] | IN | btns_4bits_tr... | | | P16 |
| btns_4bits_tri_i[0] | IN | btns_4bits_tr... | | | R18 |

- 4. Generate Bitstream and Export to SDK

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

Click on Generate Bitstream, and click Yes if prompted to Launch Implementation (Click Yes if prompted to save the design)

Click Cancel

Export the hardware by clicking File > Export > Export Hardware and click OK. This time, thereis hardware in Programmable Logic (PL) and a bitstream has been generated and should be included in the export to SDK.

Click Yes to overwrite the hardware module.

Start SDK by clicking **File > Launch SDK** and click OK

- 5. Generate TestApp Application in SDK

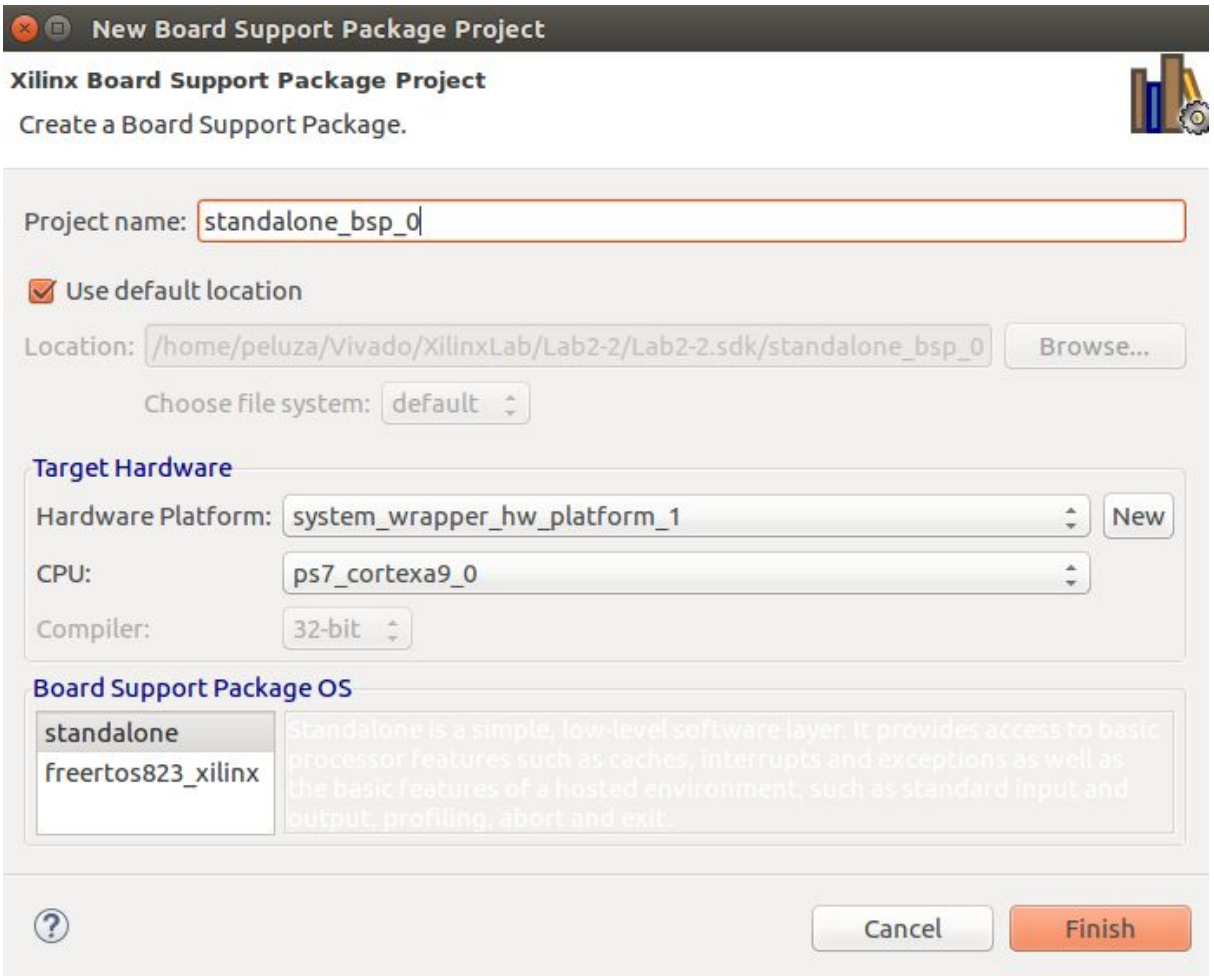
**Close the projects from the previous lab. Generate software platform project with default settings and default software project name (standalone_0).**

In SDK, **right click on the mem_test project** from the previous lab and **select Close Project**

<u>Do the same for mem_test_bsp and system_wrapper_hw_platform_0</u>

From the File menu select **File > New > Board Support Package**

Click Finish with the standalone OS selected and default project name as **standalone_bsp_0**




Click **OK to generate** the board support package **named standalone_bsp_0**

From the File menu select **File > New > Application Project**

**Name the project TestApp**, **select** Use existing board support package, **select standalone_bsp_0** and click **Next**

**Select** Empty Application and click **Finish** This will create a new Application project **using the created board support package**.

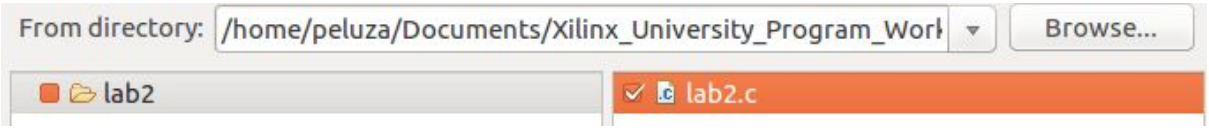
The library generator will run in the background and **will create the xparameters.h** file in the lab2\lab2.sdk**standalone_bsp_0**\ps7_cortexa9_0\include directory

**Expand** TestApp in the project view, and **right-click on the src folder**, and **select** Import

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

Browse to the **{sources}\lab2** folder

**Select** lab2.c and click **Finish**




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

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


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

int main (void)
{

    XGpio dip, push;
    int psb_check, dip_check;

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

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

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

    while (1)
    {
      psb_check = XGpio_DiscreteRead(&push, 1);
      xil_printf("Push Buttons Status %x\r\n", psb_check);
      dip_check = XGpio_DiscreteRead(&dip, 1);
      xil_printf("DIP Switch Status %x\r\n", dip_check);

      sleep(1);
    }

}
```

Figure 21. Snippet of source code

- 6. Test in Hardware

**Connect the board with a micro-usb cable(s) and power it ON. Establish the serial communication using SDK's Terminal tab.**

Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power

Select the tab. If it is not visible then select Window > Show view > **Terminal**

**Click on and if required**, select appropriate COM port (depends on your computer), and configure it with the parameters as shown. (These settings may have been saved from previous lab)

**Program the FPGA by selecting Xilinx Tools > Program FPGA and assigning system.bit file. Run the TestApp application and verify the functionality**

Select Xilinx Tools > Program FPGA

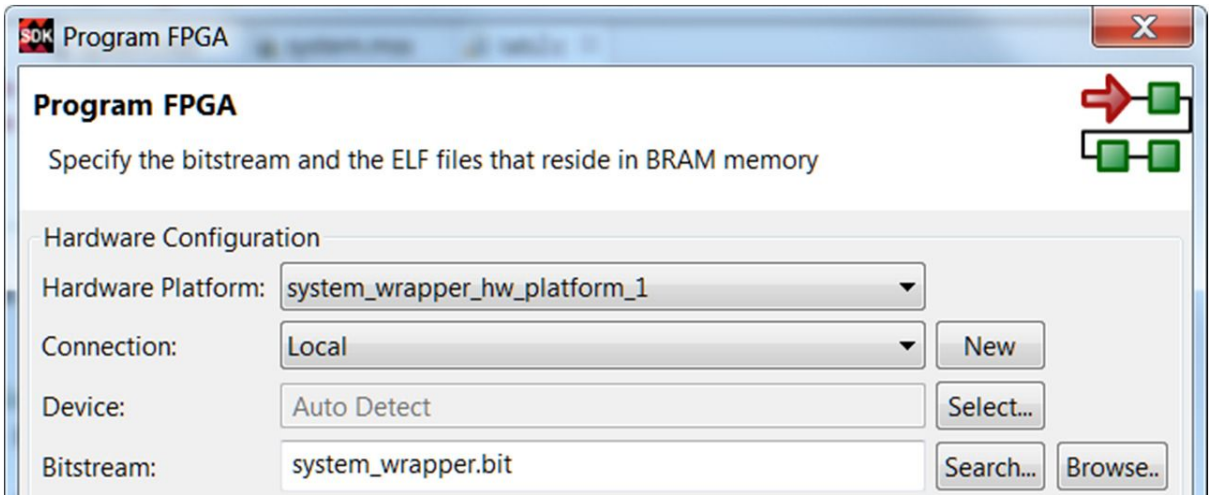
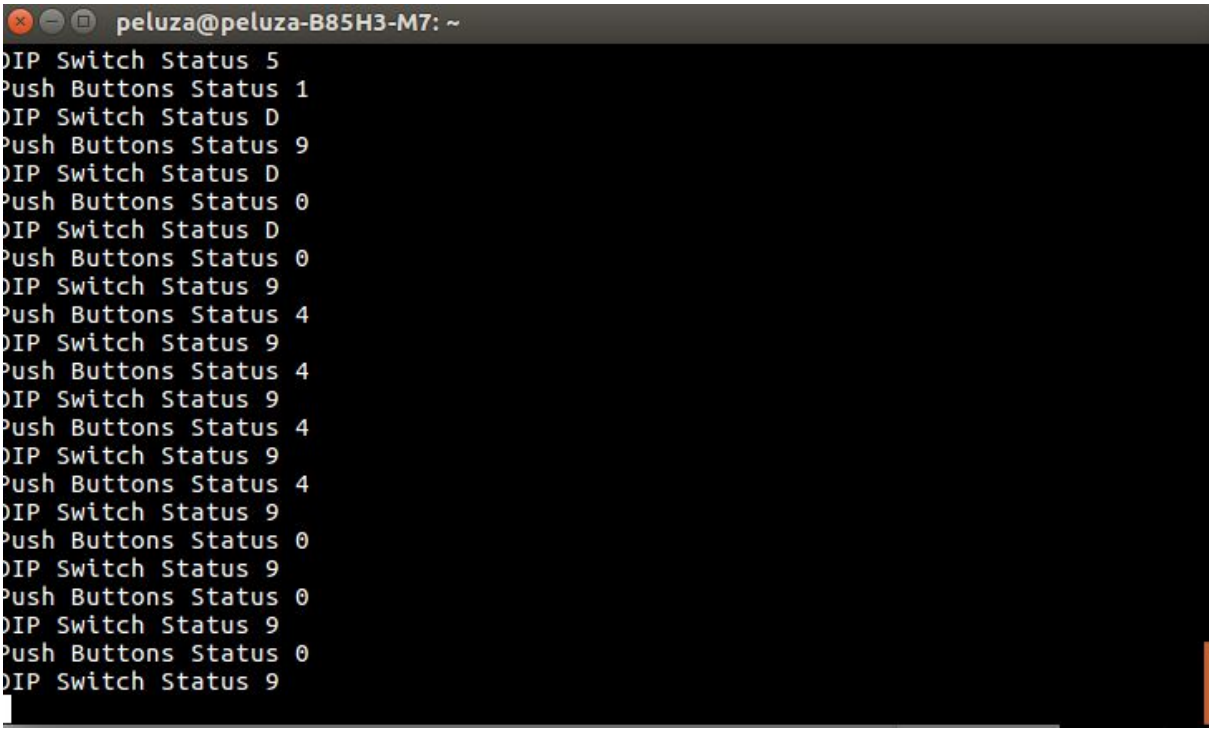



**Figure 22. Program FPGA**

Click Program to download the hardware bitstream. When FPGA is being programmed, the DONE LED (green color) will be off, and will turn on again **when the FPGA is programmed**

**Select TestApp in Project Explorer, right-click and select Run As > Launch on Hardware (GDB)** **to download the application**, execute ps7_init, and execute TestApp.elf




You should see the something similar to the following output on Terminal console

```
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
```

Figure 23. SDK Terminal output

Select Console tab and click on the **Terminate button ( ) to stop the program**

**Close SDK and Vivado programs by selecting File > Exit in each program**

**Power OFF** the board
Run 정지, SDK 종료하고 나서 Power OFF

- Conclusion

GPIO peripherals were added from the IP catalog **and connected to the Processing System through the 32b Master GP0 interface**. **The peripherals were configured and external FPGA connections were established**. A TestApp application project was created and the functionality was verified after downloading the bitstream **and executing the program**.