



Adding EDK IP to an Embedded System Lab

Objectives

This lab demonstrates how to add and modify IP to an existing PowerPC system using Xilinx Platform Studio (XPS). The system from the previous lab will be used as the starting point. The lab will show

- How add an EDK IP
- How to connect to the existing system
- How to modify the peripheral options
- How to add constraints for the new IP
- How to create a new C application in SDK

Requirements

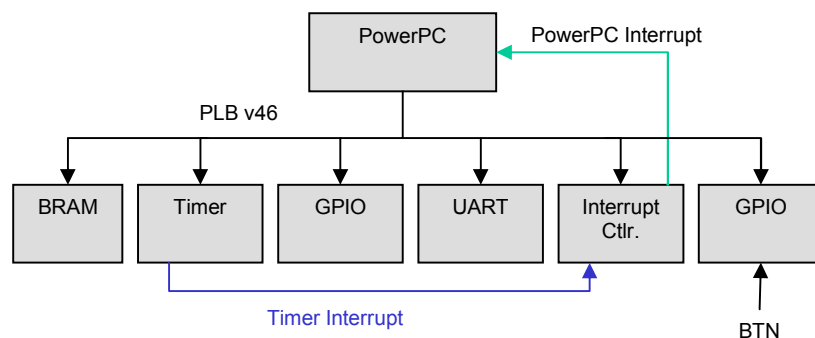
- ISE 9.2.04
- EDK 9.2.02
- ML403 Board
- Platform USB Cable
- Completion of the Creating an Embedded System Lab
- RS232 Cable

Reference

- Platform Studio Help
- Platform Studio SDK Help
- Embedded System Tools Reference Manual
- XPS GPIO datasheet

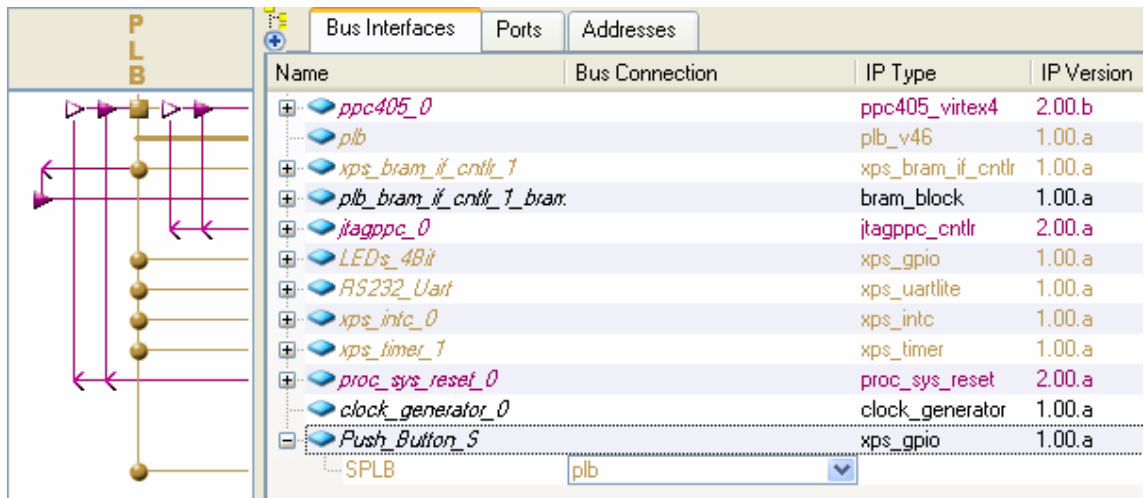
Overview

The lab is divided into three main steps: adding the new IP, writing code for the IP and testing the system on the board. The test application will reside in Block memory inside the FPGA. Below is a block diagram of the hardware platform. We will add a GPIO core to make use of push button GPIO_SW_S on the board.

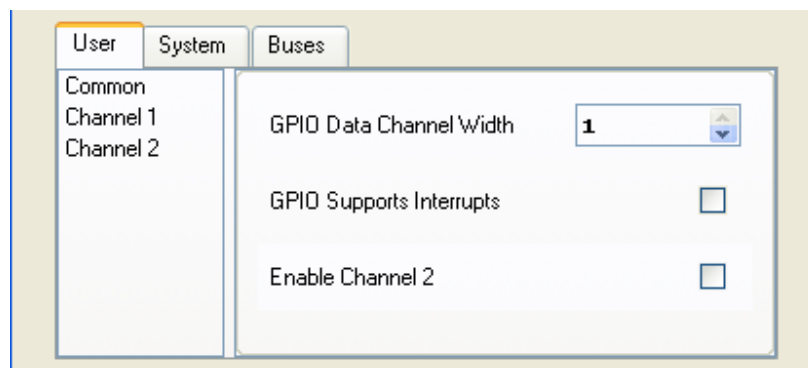


Lab Steps

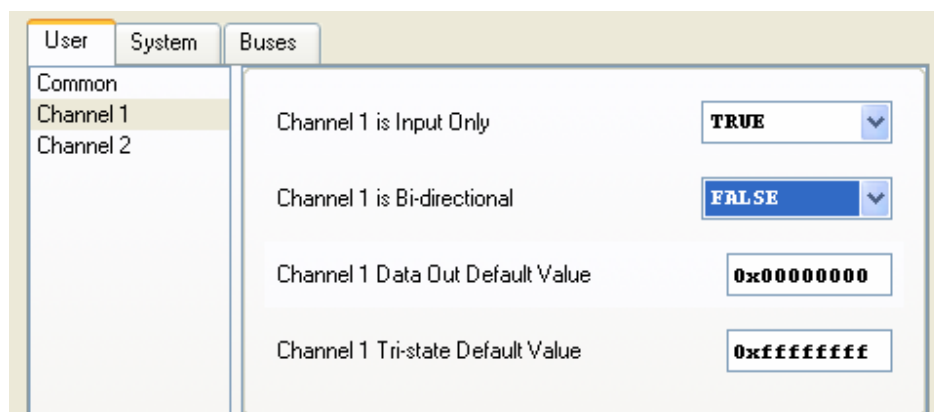
I. Adding New IP



6. **Double-click** on *Push_Button_S* to open the configuration options for the GPIO peripheral. **Change** the *GPIO Data Channel Width* from 32 to 1.



7. **Select** *Channel 1* from the list on the right side. **Change** *Bidirectional* to **FALSE** and *Input Only* to **TRUE**



8. Click **OK**.



- Click on the *Addresses* tab. The addresses view shows the address space for all the peripherals. The *Lock* box prevents the address for that peripheral from being changed when generating new addresses.

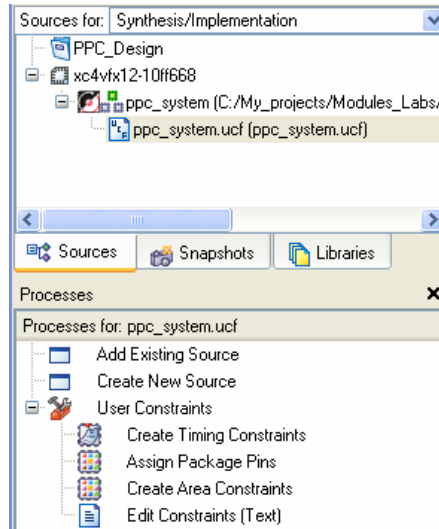
| Instance | Name | Base Address | High Address | Size | Bus Interface(s) | Bus Connection | Lock |
|---------------------|-----------------|--------------|--------------|------|------------------|----------------|--------------------------|
| xps_bram_if_cntlr_1 | C_BASEADDR | 0xffff0000 | 0xffffffff | 64K | SPLB | plb | <input type="checkbox"/> |
| Push_Button_S | C_BASEADDR | | | U | SPLB | plb | <input type="checkbox"/> |
| LEDs_4Bit | C_BASEADDR | 0x81400000 | 0x8140ffff | 64K | SPLB | plb | <input type="checkbox"/> |
| xps_intc_0 | C_BASEADDR | 0x81800000 | 0x8180ffff | 64K | SPLB | plb | <input type="checkbox"/> |
| xps_timer_1 | C_BASEADDR | 0x83c00000 | 0x83c0ffff | 64K | SPLB | plb | <input type="checkbox"/> |
| RS232_Uart | C_BASEADDR | 0x84000000 | 0x8400ffff | 64K | SPLB | plb | <input type="checkbox"/> |
| ppc405_0 | C_IDCR_BASEADDR | 0b0100000000 | 0b0111111111 | 256 | Not Connected | | <input type="checkbox"/> |

- Click on the *Generate Addresses* button to generate the address range for the new IP.
- Click on the *Ports* tab. The Ports view shows the internal connections between the peripherals as well as the external ports connections.
- Expand *Push_Button_S* from the list. It will show the connections available for the peripheral. Look at the GPIO datasheet for a description of each port.
- Select *GPIO_in* since the push button is only an input. Click on the *No Connection* drop-down list in the Net Column. Select *Make External* to add it the external ports list. Expand the *External Ports* to view the new connection.

| Name | Net | Direction | Range |
|------------------------------|--------------------------|-----------|----------------------|
| External Ports | | | |
| Push_Button_S_GPIO_in_pin | Push_Button_S_GPIO_in | I | [0:0] |
| sys_rst_pin | sys_rst_s | I | |
| sys_clk_pin | dcm_clk_s | I | |
| fpga_0_LEDs_4Bit_GPIO_IO_pin | fpga_0_LEDs_4Bit_GPIO_IO | IO | [0:3] |
| fpga_0_RS232_Uart_TX_pin | fpga_0_RS232_Uart_TX | O | |
| fpga_0_RS232_Uart_RX_pin | fpga_0_RS232_Uart_RX | I | |
| ppc405_0 | | | |
| plb | | | |
| xps_bram_if_cntlr_1 | | | |
| plb_bram_if_cntlr_1_bram | | | |
| jtagppc_0 | | | |
| LEDs_4Bit | | | |
| RS232_Uart | | | |
| xps_intc_0 | | | |
| xps_timer_1 | | | |
| proc_sys_reset_0 | | | |
| clock_generator_0 | | | |
| Push_Button_S | | | |
| GPIO2_t_out | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO2_d_out | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO2_in | No Connection | I | [0:(C_GPIO_WIDTH-1)] |
| GPIO2_IO_T | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO2_IO_0 | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO2_IO_I | No Connection | I | [0:(C_GPIO_WIDTH-1)] |
| GPIO2_IO | No Connection | IO | [0:(C_GPIO_WIDTH-1)] |
| GPIO_t_out | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO_d_out | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO_in | Push_Button_S_GPIO_in | I | [0:(C_GPIO_WIDTH-1)] |
| GPIO_IO_T | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO_IO_0 | No Connection | O | [0:(C_GPIO_WIDTH-1)] |
| GPIO_IO_I | No Connection | I | [0:(C_GPIO_WIDTH-1)] |



14. Click on the *Bus Interface* tab to go back to the system assembly view.
15. We need to update the constraint file to add the pinout information for the switch. In ISE, select the *ppc_system.ucf* file. Expand the User Constraints and double-click on *Edit Constraints (Text)*.



16. Look at the ML40x user guide to find the FPGA pin for push button GPIO_SW_S.

Table 7: User Push Button Connections

| Reference Designator | Label/Definition | FPGA Pin |
|----------------------|--------------------|----------|
| SW3 | GPIO Switch North | E7 |
| SW5 | GPIO Switch East | F10 |
| SW4 | GPIO Switch South | A6 |
| SW7 | GPIO Switch West | E9 |
| SW6 | GPIO Switch Center | B6 |

17. In the UCF, add the line:
 - NET Push_Button_S_GPIO_in_pin<0> LOC = A6 | IOSTANDARD = LVCMOS25;
18. Make sure to leave a space between A6 and |. **Save** and **Close** the file
19. **Select** the *ppc_system* source
20. **Double-click** on **Update Bitstream with Processor Data** to update the bit file with the new peripheral.

II. Writing Code for the New IP

To test the new peripheral we will create a new software application in Platform Studio SDK and use the GPIO device drivers.

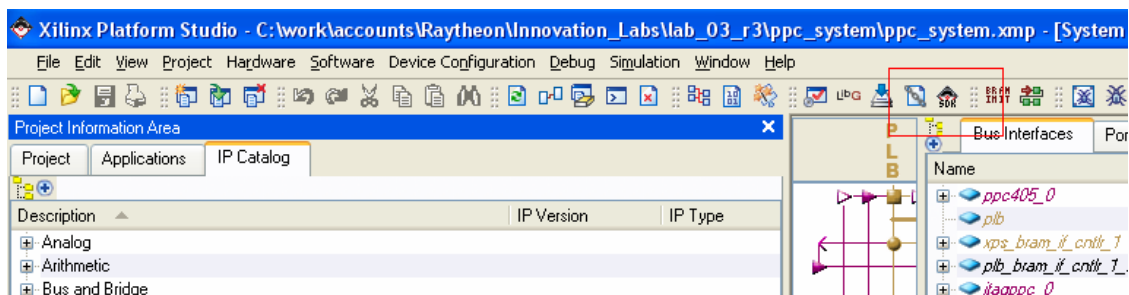


1. Open the GPIO datasheet to view the GPIO register map. In XPS, right-click on the *Push_Button_S* instance in the system assembly view and select *View PDF Datasheet*. The GPIO Data Register is located at the base address of the peripheral, which is 0x8140_0000.

Table 4: XPS GPIO Registers

| Register Name | Description | PLB Address | Access |
|---------------|-------------------------------------|-------------------|------------|
| GPIO_DATA | Channel 1 XPS GPIO Data Register | C_BASEADDR + 0x00 | Read/Write |
| GPIO_TRI | Channel 1 XPS GPIO 3-state Register | C_BASEADDR + 0x04 | Read/Write |
| GPIO2_DATA | Channel 2 XPS GPIO Data register | C_BASEADDR + 0x08 | Read/Write |
| GPIO2_TRI | Channel 2 XPS GPIO 3-state Register | C_BASEADDR + 0x0C | Read/Write |

2. In XPS, click on the **Launch Platform Studio SDK** button on the toolbar.



3. Select **Create a New SDK C Application Project**. Click on **Next**.
4. Name the project *Lab_Test* and click **Finish**. A new file *main.c* will be created.
5. Expand the processor instance *ppc405_0* from *ppc405_0_sw_platform* in the *Navigator* window. Expand the *include* directory. Double-click on the *xparameters.h* file to view the hardware parameters for the system. Using the macros isolate the software from the actual hardware parameters.

6. Modify the existing code to:

```
#include "xparameters.h"
#include "stdio.h"
#include "xbasic_types.h"

//=====

int main (void) {

    print("-- Entering main() --\r\n");

    return 0;

}
```

7. Save the *main.c* file. The application will be compiled when saved. The *Project* menu gives options to change the behavior for building the application.
8. Create a Linker Script for the new application. Right-click on the *Lab_Test* project and select **Generate Linker Script...** All the sections are targeted for internal BRAMs. Click on **Generate**. Click **OK**.
9. We will add code after the print statement to turn-on the LEDs when the push button is pressed.



10. Inside the expanded *include* directory for *ppc405_0* are all the driver header files for the different peripherals. The *_l.h* denotes a low level driver. **Double-click** on *xgpio_l.h* to view the GPIO low level functions.
11. **Click** on *Xgpio_mGetDataReg* in the *Outline* window to view the format to read the GPIO data register. We will also use the function *XGpio_mSetDataReg* to write to the LEDs. The BaseAddress for the device can be found in the *xparameters.h* file.
12. Write code to read from channel 1:
 - Include the low level driver header for the GPIO after the other include statements

```
#include "xgpio_l.h"
```
 - Declare a new global variable before *int main (void) {*

```
Xuint32 Push_Read;
```
 - Add code to set the GPIO direction for the LEDs to output after the print statement and before *return 0:*

```
XGpio_mSetDataDirection(XPAR_LEDS_4BIT_BASEADDR, 1, 0x00000000);
```
 - Add code to read from the push button after the previous command:

```
while (1) {
    Push_Read = XGpio_mGetDataReg(XPAR_PUSH_BUTTON_S_BASEADDR, 1);
```
 - Add code to write to the LEDs when the button is pressed:

```
if (Push_Read == 0x00000001) {
    XGpio_mSetDataReg(XPAR_LEDS_4BIT_BASEADDR, 1, 0x0000000F);
}
else {
    XGpio_mSetDataReg(XPAR_LEDS_4BIT_BASEADDR, 1, 0x00000000);
}
}
```
13. **Save** and **close** the file. The application will be compiled automatically.
14. To view the changes made to *main.c*, **right-click** on *main.c* in the *Navigator* window and select **Compare With Local History...** Click **OK**.
15. Go to **Device Configuration > Program Hardware Settings...** Select **Lab_Test.elf** and click **Save**. The Lab_Test application will be loaded in the BRAMs during configuration.
16. The system is ready to be downloaded to the board.

III. Test the Generated System with the Sample Application

1. Connect a serial cable to the RS232 port and connect the USB platform cable.
2. Power-up the board.
3. Open Hyperterminal and connect at 9600, no parity.
4. Download to the FPGA from SDK: **Device Configuration > Program Hardware**.
5. Verify that the print statement appears on the Hyperterminal window.
6. Pressing the Push Button (GPIO_SW_S) should turn on the LEDs.
7. Close SDK.