Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki
Zakład Inżynierii Komputerowej

LAB 2.

Adding IP to a Hardware Design

Introduction

This lab guides you through the process of adding additional IP to an existing processor system by using Xilinx Platform Studio (XPS). You will add GPIO peripheral from the IP Catalog tab to interface to the push buttons and DIP switches on the Spartan-3E Starter Kit. At the end of the lab, you will generate the bitstream and test the peripherals in hardware.

Objectives

After completing this lab, you will be able to:

- Add additional IP to a hardware design
- Update ucf file to support external ports of the added IP
- Setup some of the compiler settings

Design Description

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

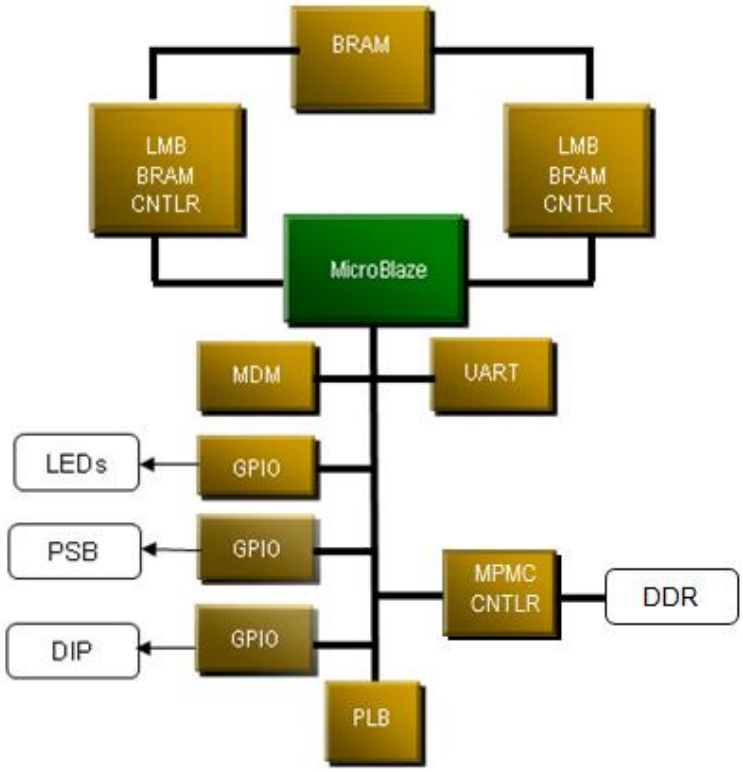


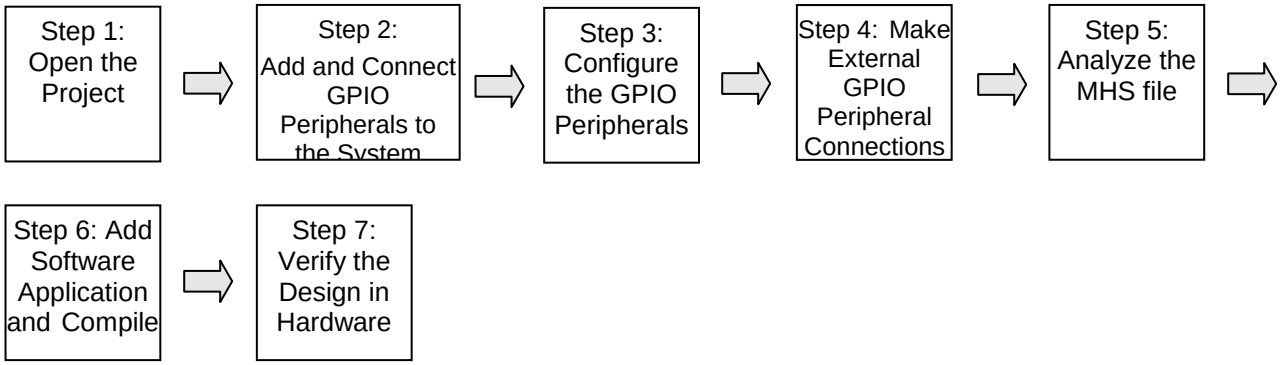
Figure 1. Extend the System from the previous lab

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 7 primary steps: You will open the project, add and connect GPIO peripherals in the system, configure the GPIO peripherals, make external GPIO connections, analyze the MHS file, add the software application and compile and, finally, verify the design in hardware.
Note: If you are unable to complete the lab at this time, you can download the original lab files for this module from the Xilinx University Program site at http://www.xilinx.com/university

General Flow for this Lab



Open the Project Step 1

1-1. Create a *lab2* folder and copy the contents of the *lab1* folder into the *lab2* folder. Launch Xilinx Platform Studio (XPS) and open the project file.

1-1-1. Create a *lab2* folder in the **D:** directory and copy the contents from *lab1* to *lab2*.

1-1-2. Open XPS by selecting **Start → All Programs → Xilinx → Xilinx ISE Design Suite 13.2 → EDK → Xilinx Platform Studio.**

1-1-3. Select **Open a recent project, Click OK** and browse to **D:\lab2**.

1-1-4. Click **system.xmp** to open the project.

Add and Connect GPIO Peripherals to the System Step 2

2-1. Add two instances of an XPS GPIO Peripheral from the IP catalog to the processor system via the System Assembly View.

XPS provides two methods for adding peripherals to an existing project. You will use the first method, the System Assembly View panel, to add most of the additional IP and connect them. The second method is to manually edit MHS file.

2-1-1. Select the **IP Catalog** tab in the left window and click on plus sign next to **General Purpose IO** entry to view the available cores under it **(Figure 2).**

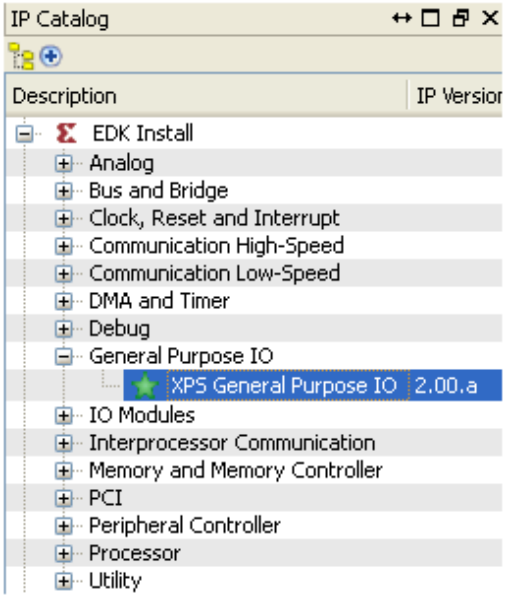


Figure 2. IP Catalog

2-1-2. Double-click on the **XPS General Purpose IO** core twice to add two instances to the System Assembly View, each time clicking OK to accept the default configuration (you can make changes to configuration settings, but we will do it later).

2-1-3. Change the instance names of the peripherals to **dip** and **push**, by clicking once in the name column, typing the new name for the peripheral followed by pressing Enter key.

At this point, the System Assembly View should look like the following **(Figure 3)**:

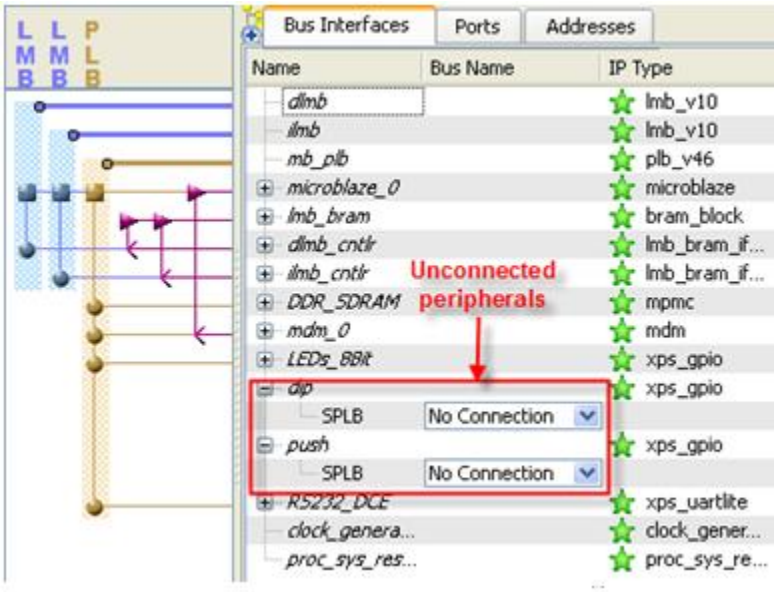


Figure 3. System Assembly View After Adding Peripherals

2-1-4. Click once in **Bus Connection** column for the **push** and **dip** instances to connect them as slave devices to the **PLB**.

At this point, the Bus Connections tab should look like the following **(Figure 4)**:

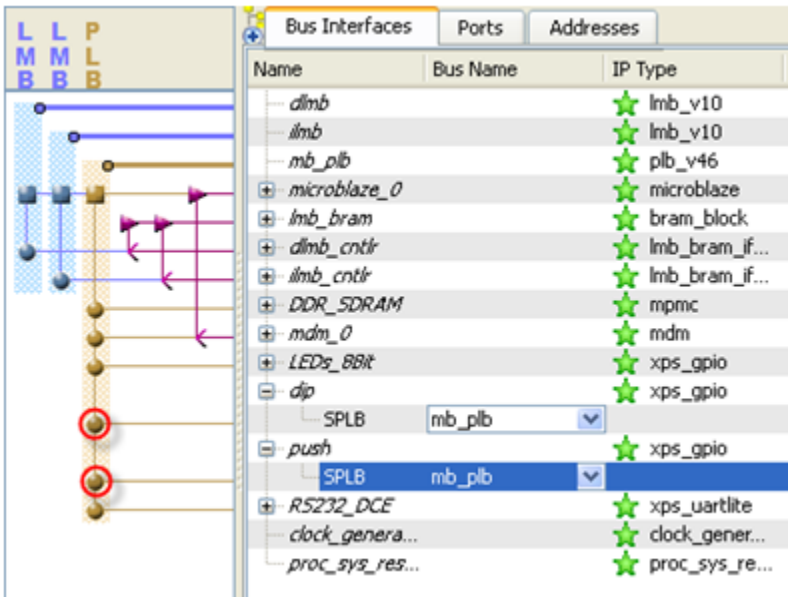
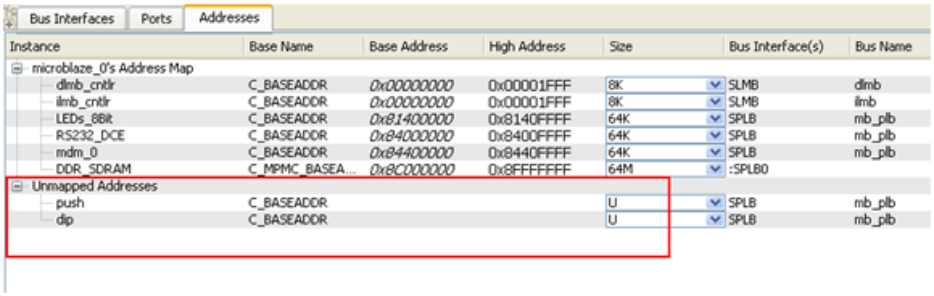


Figure 4. Bus Interfaces Tab showing Bus Connections to the Added Peripherals

2-1-5. Select the **Addresses** filter.

Note that there are few instances which are not assigned addresses.



You can manually assign the base address and size of your peripherals or have XPS generate the addresses for you.

2-1-6. Click **Generate Addresses** (located on the right most end of the tabs) to automatically generate the base and high addresses for the peripherals in the system. The base address and high addresses will change as shown in **Figure 5**.

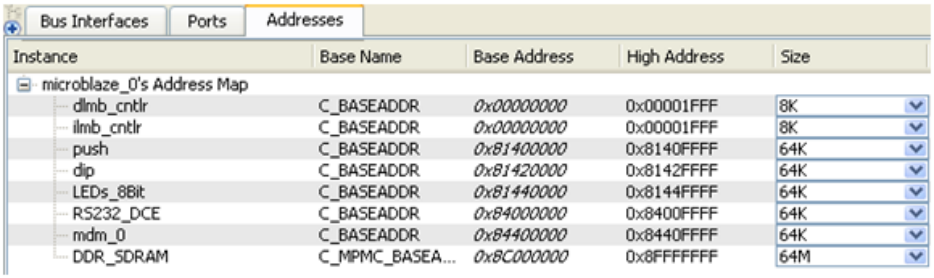


Figure 5. Peripherals Memory Map

Configure the GPIO Peripherals Step 3

3-1. There are four push buttons and four DIP switches on the Spartan-3E starter kit. You will first configure the push and dip instances according to their sizes and direction, and then make external pin connections.

3-1-1. Select the **Ports** filter in the toolbar of the System Assembly View.

3-1-2. Double-click on the **push** instance to access the configuration window.

Notice that the peripheral can be configured for two channels, but, since we want to use only one channel without interrupt, leave the **GPIO Supports Interrupts** and **Enable Channel 2** *unchecked*.

The settings for the Common parameters should be set according to **Figure 6** below.

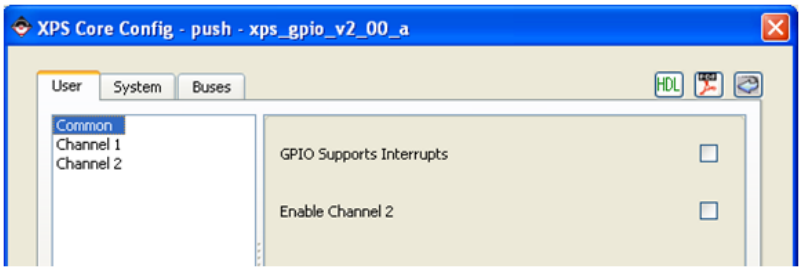


Figure 6. Configurable Common Parameters of GPIO Instance for Push Buttons

3-1-3. Next click **Channel 1**, click on the **GPIO Data Channel Width** down arrow and set it to **4**, you will use 4 push buttons on the Spartan-3E starter kit. Set **Channel 1 is input Only** to *True* **(Figure 7)**:

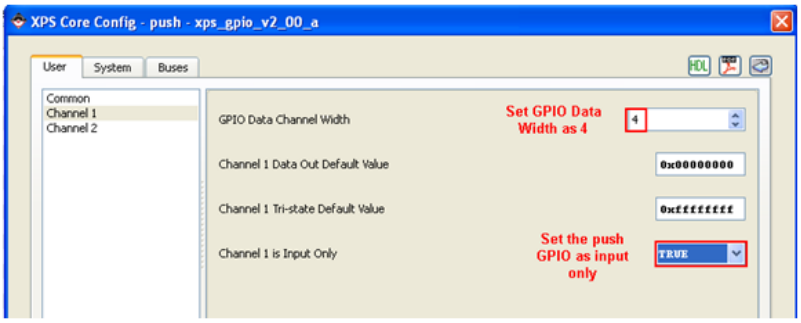


Figure 7. Setting Configurable Parameters for Push Buttons

3-1-4. Similarly, set the same parameters for the **dip** instance, as performed for the push buttons.

Make External GPIO Peripheral Connections Step 4

4-1. You will connect the push and dip instances to the push buttons and DIP switches on the Spartan-3E starter kit. In order to do this, you must establish the GPIO data ports as external FPGA pins and then assign them to the proper locations on the FPGA via the UCF file. The location constraints are provided for you in this section. Normally, one would consult the Spartan-3E starter kit user manual to find this information.

4-1-1. In the *Net* field of the**GPIO_IO_I** port of the **push** instance, make the **GPIO_IO_I** port as external by selecting **Make External**. You should see a new external net connection **(Figure 8)**.

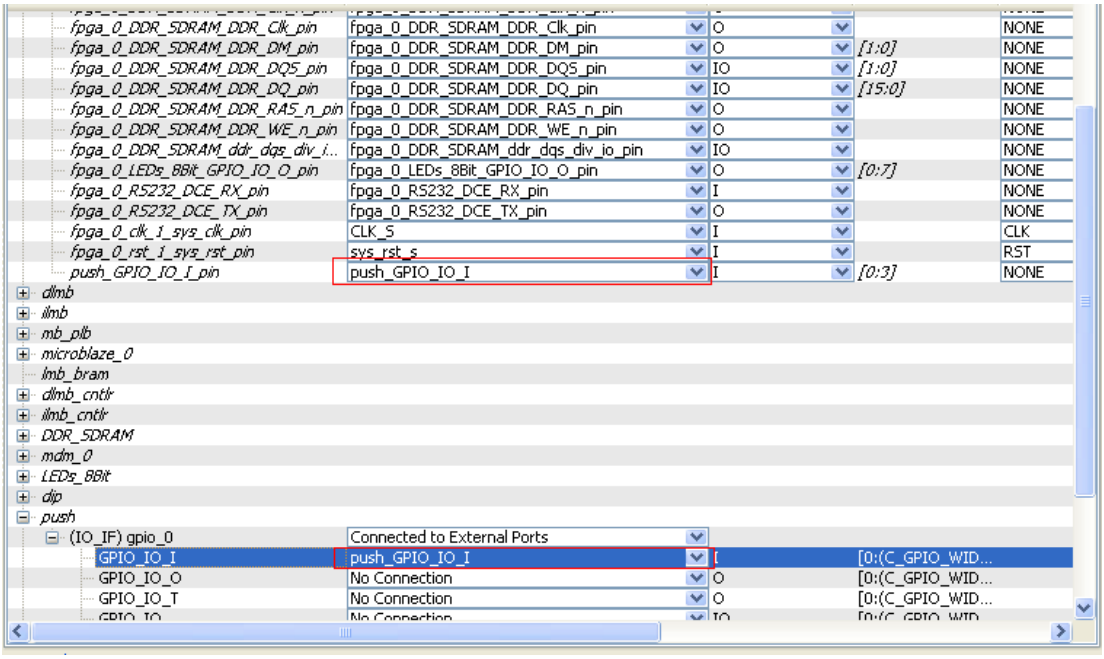


Figure 8. GPIO_in Port Connection Added to push Instance

4-1-2. Similarly, make the **GPIO_IO_I** port of **dip** as external in the net field of the **GPIO_IO_I** port of the **dip** instance.

The GPIO_IO_I ports of both **dip** and **push** are now connected externally on the FPGA (**Figure 9**).

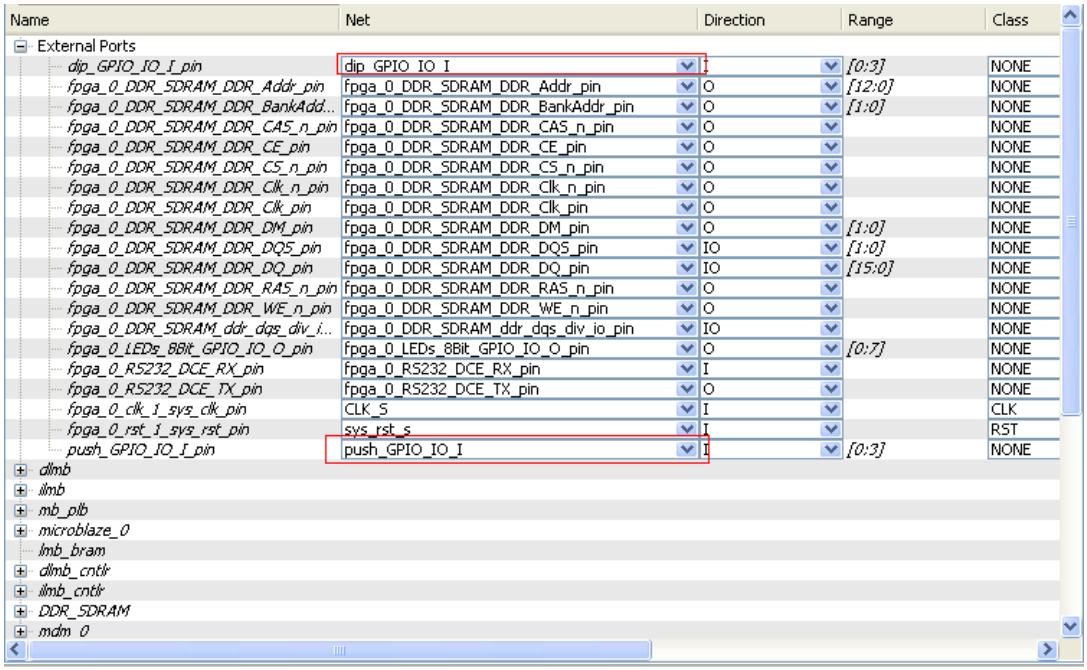


Figure 9. Push and DIP Instances External Ports

4-1-3. Click on the **system.ucf** file under the **Project** tab and add the following code to assign pins to push buttons (The constraints are provided in **lab2.ucf** file in **W:\A.Bukowiec\zajecia\SW\source**. Copy it from there and paste it in your ucf file).

```
#### Pin location constraints for the push buttons
NET push_GPIO_IO_I_pin<0> LOC=V4 | IOSTANDARD = LVTTL | PULLDOWN ; # North Push Button
NET push_GPIO_IO_I_pin<1> LOC=H13 | IOSTANDARD = LVTTL | PULLDOWN; # East Push Button
NET push_GPIO_IO_I_pin<2> LOC=D18 | IOSTANDARD = LVTTL | PULLDOWN; # West Push Button
Net push_GPIO_IO_I_pin<3> LOC=V16 | IOSTANDARD = LVTTL | PULLDOWN; # Center Push Button

#### Pin location constraints for the DIP switches
NET dip_GPIO_IO_I_pin<0> LOC=L13 | IOSTANDARD = LVTTL | PULLUP ; # Switch0
NET dip_GPIO_IO_I_pin<1> LOC=L14 | IOSTANDARD = LVTTL | PULLUP; # Switch1
NET dip_GPIO_IO_I_pin<2> LOC=H18 | IOSTANDARD = LVTTL | PULLUP; # Switch2
Net dip_GPIO_IO_I_pin<3> LOC=N17 | IOSTANDARD = LVTTL | PULLUP; # Switch3
```

Figure 10. UCF file (pin assignments).

4-1-4. Save the system.ucf and close it.

Analyze the MHS file Step 5

5-1. Open the system.mhs file, study its contents, and answer the following questions.

5-1-1. Double-click the **system.mhs** file to open it if it is not already open.

Study the external ports sections and answer the following questions:

Question 1

Complete the following:

Number of external ports: _____

Number of external ports that are output: _____

Number of external ports that are input: _____

Number of external ports that are bidirectional: _____

5-1-2. Review the entire MHS file.

Question 2

List the **instances** to which the clk_s is connected:

_____

_____

List the **instances** connected to the mb_plb bus:

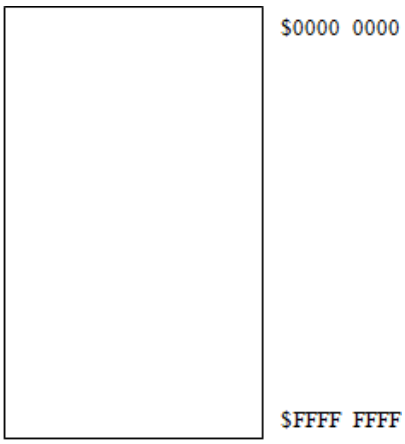
_____

_____

5-1-3. Review the memory map in the **Addresses** tab of the **System Assembly View.**

Question 3

Draw the **address map** of the system, providing instance names:



Add Software Application and Compile Step 6

6-1. Start SDK from XPS, generate software platform project with default settings and default software project name.

6-1-1. Start SDK by clicking **Project → Export Hardware Design to SDK** and click on **Export & Launch SDK** button with default settings.

Since we have not generated hardware bitstream and the default option is selected, a hardware bitstream will be generated and then SDK will be open.

6-1-2. Browse to **SDK\SDK_Export** folder of your project and click **OK**.

6-1-3. In SDK, select **File →New→Xilinx Board Support Package.**

6-1-4. Click **Finish** with default settings (with standalone operating system).

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

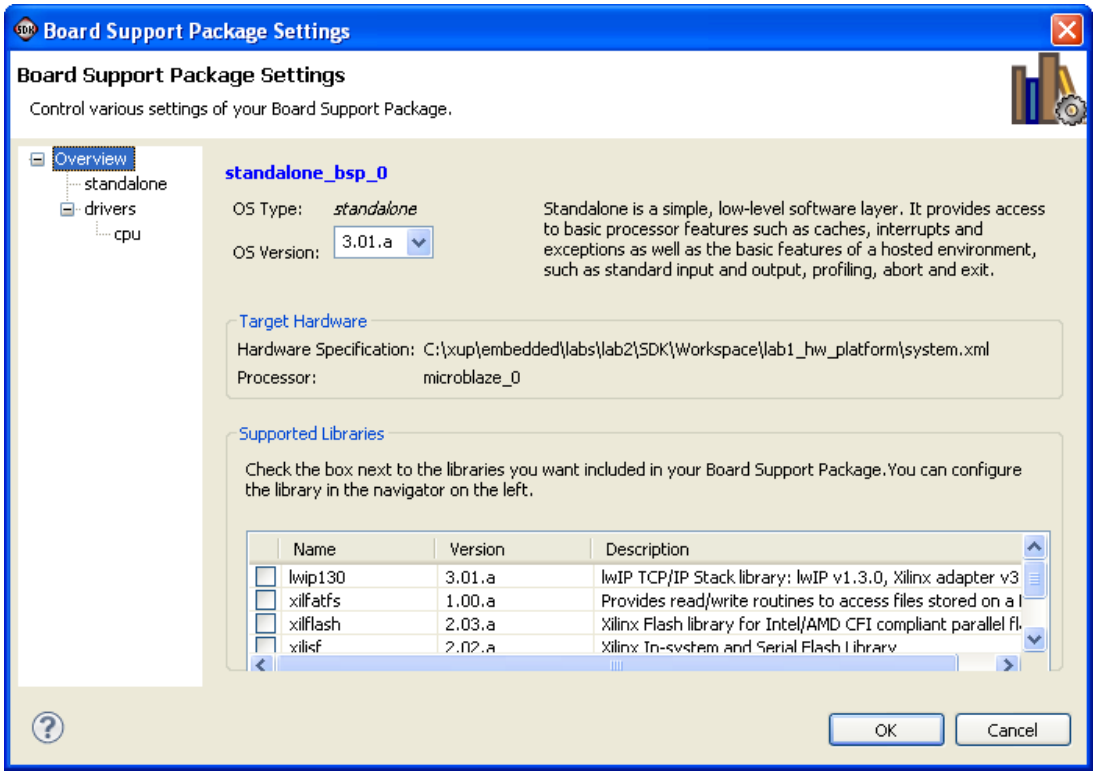


Figure 1. Board Support Package Settings

6-1-5. Click **OK** to accept the default settings, as we want to create a **standalone_bsp_0** software platform project without requiring any additional libraries support.

6-1-6. The library generator will run in the background and will create **xparameters.h** file in the **D:\lab2\SDK\SDK_Export\standalone_bsp_0\microblaze_0\include** directory.

6-2. Create an application project and import the provided lab2.c file.

6-2-1. Select **File → New → Xilinx C Project.**

6-2-2. Select **Empty Application** in the Select Project Template window, and enter **TestApp** as the **Project name** and click **Next.**

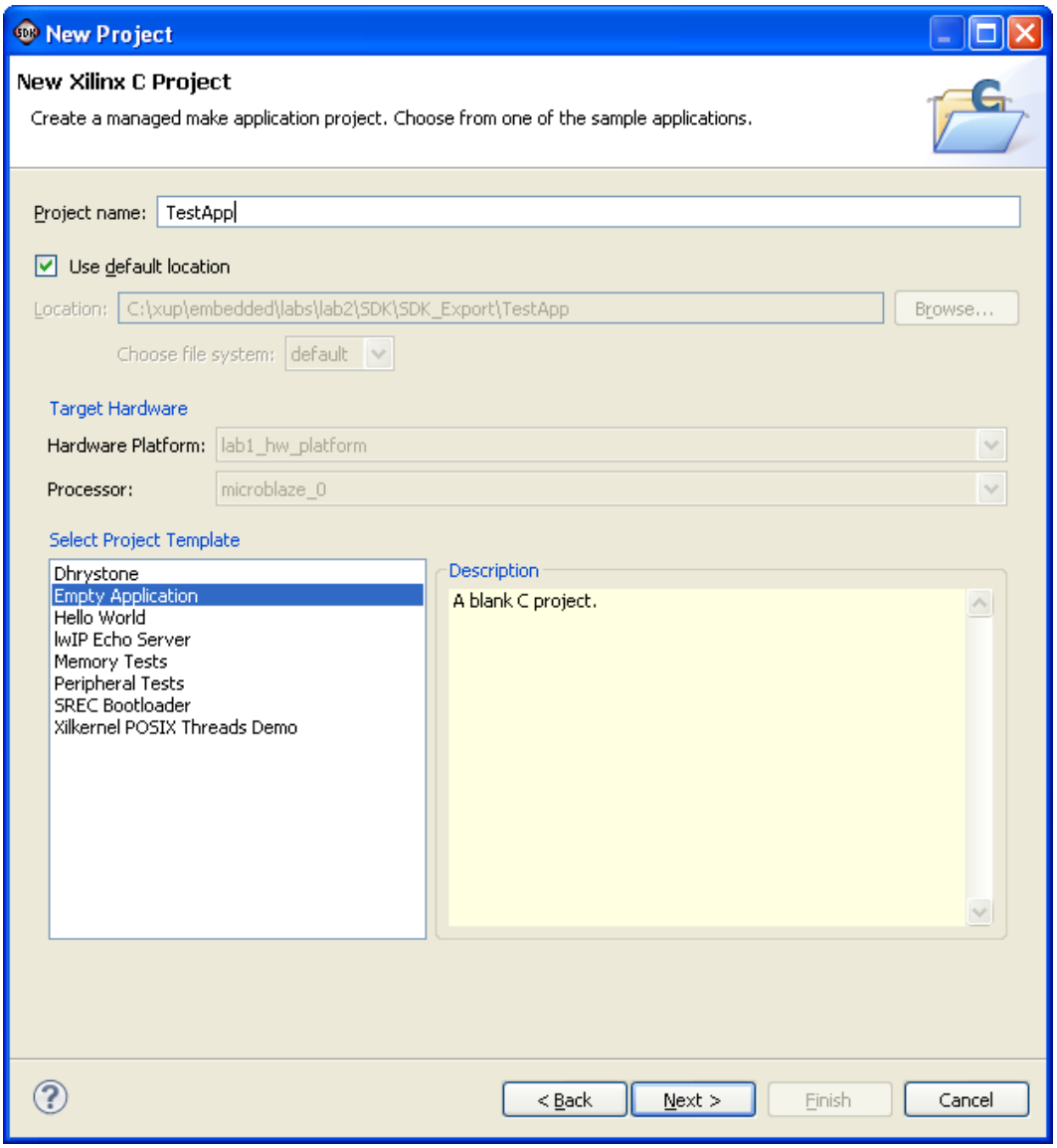


Figure 2. Create a blank C Project

6-2-3. Select **Target an existing Board Support Package** option, then select **standalone_bsp_0** and click **Finish.**

6-2-4. The TestApp project will be created in the Project Explorer window of SDK.

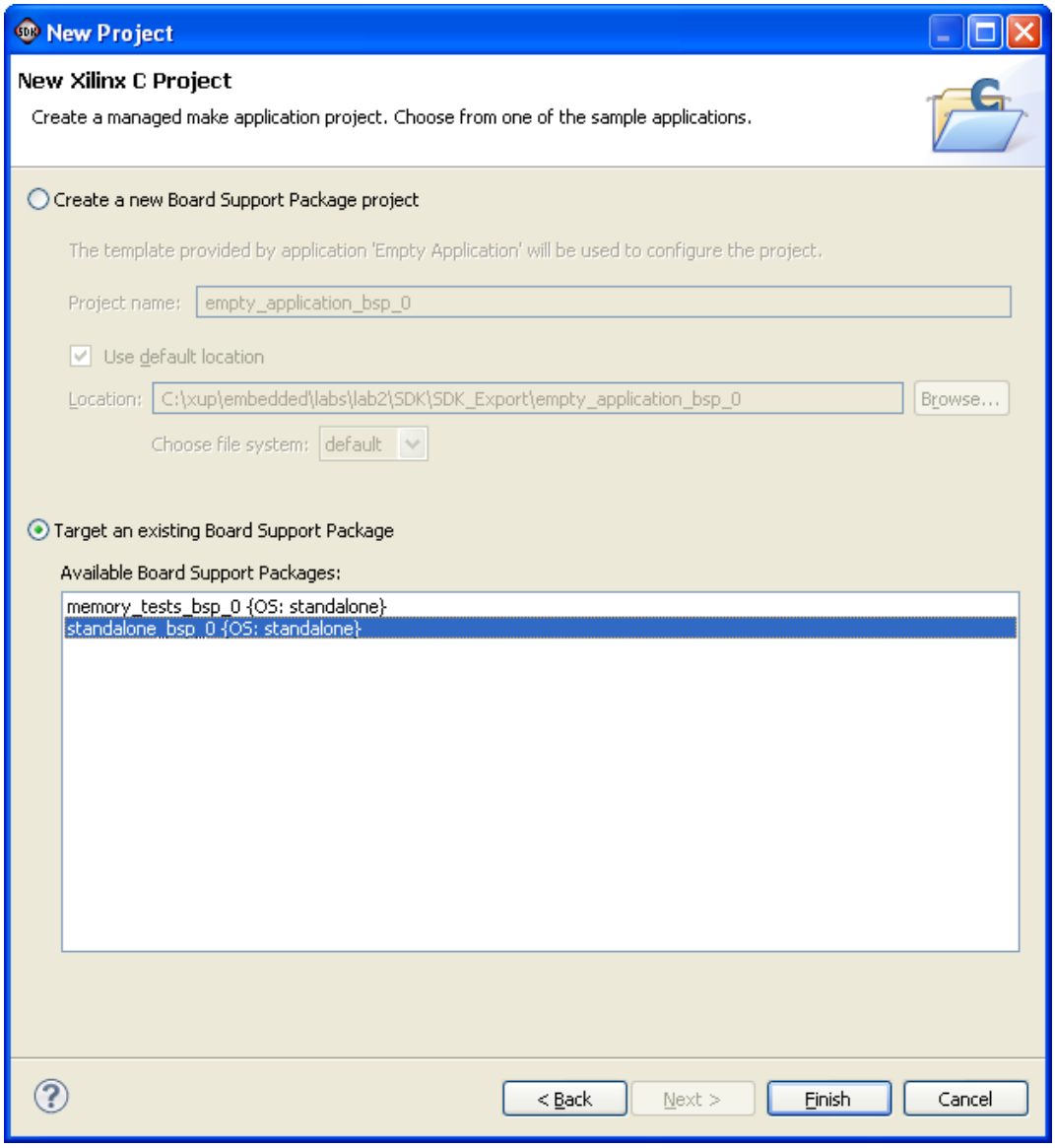


Figure 3. Use Existing Board Support Package

6-2-5. Select **TestApp** in the project view, right-click, and select **Import.**

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

6-2-7. Browse to **W:\A.Bukowiec\zajecia\SW\source**.

6-2-8. Select **lab2.c** and click **Finish.**

A snippet of the source code is shown in **Figure 4.**

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


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

int main (void)
{

   XGpio dip, push;
    int i, psb_check, dip_check;

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

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

    XGpio_Initialize(&push, XPAR_PUSH_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);

      for (i=0; i<999999; i++);
    }

}
```

Figure 4. Snippet of source code.

6-3. Set build setting to no optimization and generate linker script which targets the application to the ilmb and dlmb memories as well as have 400 bytes of heap and stack each.

6-3-1. Select **TestApp** project, right-click, and select **C/C++ Build Settings.**

6-3-2. Select **Optimization** option of the MicroBlaze gcc compiler in the **Tool Settings** tab and make sure that the Optimization Level is set to **None (-O0)** as we have a software loop acting as a delay loop and we do not want it to be optimized away.

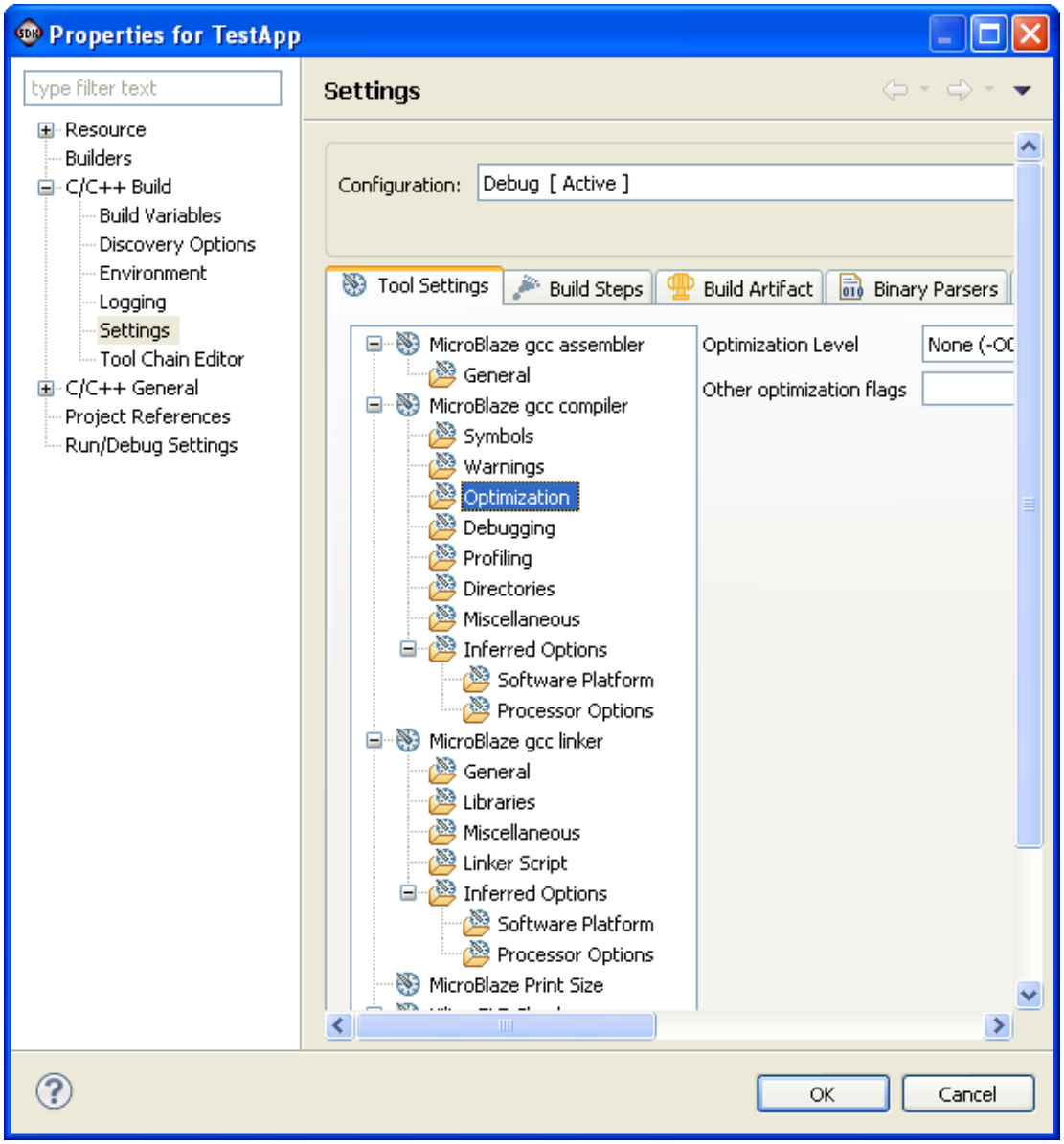


Figure 5. Setting the Compiler Settings

6-3-3. Select **TestApp**, right-click and select **Generate Linker Script.**

6-3-4. Target everything to ilmb and dlmb memories, set heap and stack to 400 bytes each, click **Generate** and click **Yes.**

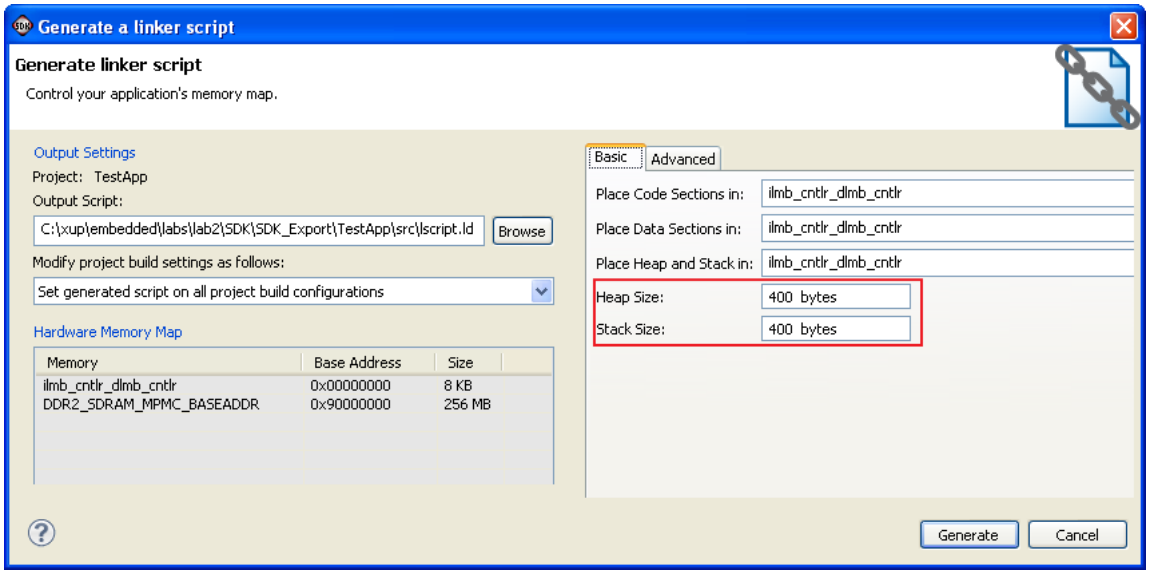


Figure 6. Setting Linker Script

Verify the Design in Hardware — Step 7

7-1. Download the bitstream to the Spartan-3E starter kit.

7-1-1. Connect and power up the Spartan-3E starter kit.

7-1-2. Start a PuTTY serial session with baud rate 115200.

7-1-3. Select **Xilinx Tools → Program FPGA in SDK.**

7-1-4. Click on **drop-down** button and select **Testapp.elf** file.

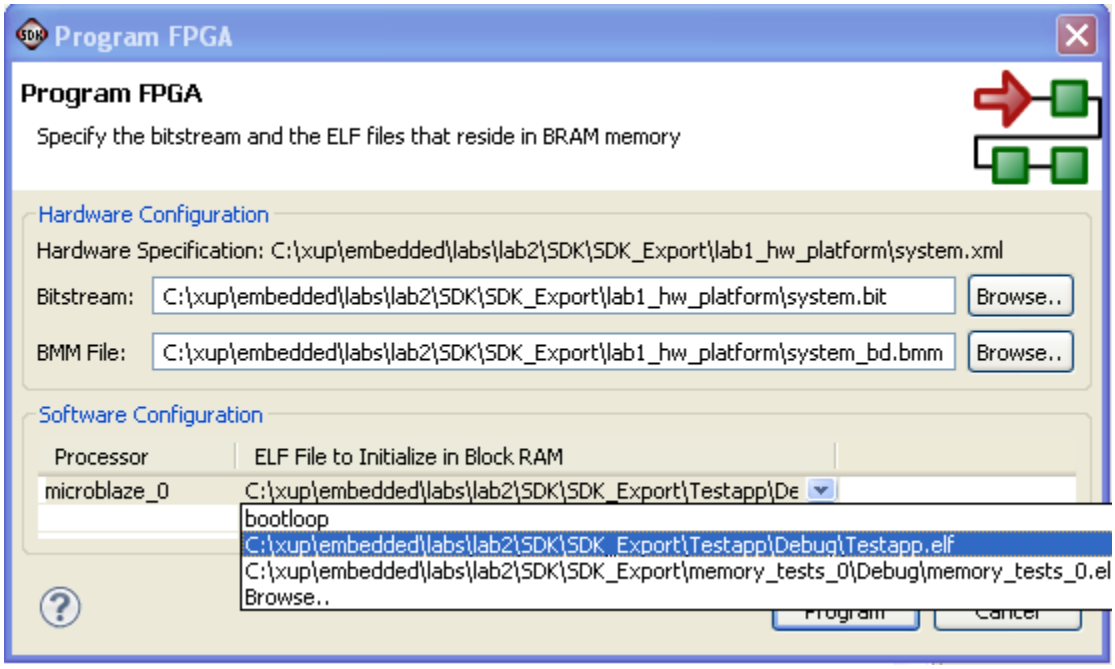


Figure 17. Selecting Application

7-1-5. Click **Program**.

The elf file and the system.bit file will be combined into download.bit file which will then be downloaded to program the FPGA.

Note: Once the bitstream is downloaded, you should see the DONE LED ON and a message displayed in PuTTY Terminal as shown in **Figure 18.**

```
DIP Switch Status 0
Push Buttons Status 0
DIP Switch Status 0
Push Buttons Status 0
DIP Switch Status 0
Push Buttons Status 0
DIP Switch Status 0
Push Buttons Status 0
DIP Switch Status 0
```

Figure 18. Screen Shot after the Bitstream Downloading

7-1-6. After pressing the **buttons** and toggling the **switches**, and you should see the corresponding values being displayed on the PuTTY Terminal **(Figure 19).**

```
DIP Switch Status 12
Push Buttons Status 2
DIP Switch Status 12
Push Buttons Status 2
DIP Switch Status 12
Push Buttons Status 2
DIP Switch Status 12
```

Figure 19. Push button and DIP switch status displayed on PuTTY Terminal

7-1-7. Disconnect and close the Hyper Terminal window, and also close SDK and XPS.

Task 1

Modify the **main** function of the **lab2.c** source file to display DIP Switch a Push Buttons Statuses in decimal, hexadecimal, octal and binary formats.

Conclusion

GPIO peripherals were added from the IP catalog and connected to a MicroBlaze system that was created in the first lab. The peripherals were configured and external FPGA connections were established. Pin location constraints were made in the UCF file to connect the peripherals to push buttons and DIP switches on the Spartan-3E starter kit.

In future labs in this course, you will learn how to add user cores, add software to the system, debug the software, and verify the functionality of the completed design by using a Spartan-3E starter kit.