

Building an Embedded Processor System on Xilinx NEXYS3 FPGA: A Tutorial

Introduction:

Modern FPGA's are equipped with a lot of resources that allow them to hold large digital systems on a single chip. FPGA vendors provide tools that allow the designer to build embedded systems on efficiently on FPGAs. One of the new concepts that you will learn in this experiments is the system-on-chip approach (SoC). In this methodology, a complete micro-processor system is implemented on a single chip. In our experiment we will use an FPGA board that is equipped with an FPGA from Xilinx a leading FPGA vendor. Xilinx provides a tool for building an embedded SoC on its FPGAs, and we are going to use this tool in this experiment. This tool is called Xilinx Embedded Development Kit (EDK). The EDK allows the designer to build the processor system based on an embedded processor from Xilinx called MicroBlaze. The tool provides a C/C++ compiler for that processor and an IDE based on Eclipse framework. In this tutorial we will build a processor system based on MicroBlaze using the EDK and run this system on Nexys 3 FPGA board.

Objectives:

1. Demonstrate the concept of SoC.
2. Familiar students with embedded soft processor systems on FPGA.
3. Build a soft processor system that will perform simple Input/Output operations.

Equipment and Tools

1. Nexys 3 FPGA board.
2. Xilinx Embedded Development Kit.

Detailed Steps

Part 1: Building the hardware system on FPGA

In previous labs you learned about the Xilinx ISE design flow and how to use it to map a design using VHDL on FPGA. In this tutorial we will use the Xilinx EDK to build a micro-processor system and write a simple program for that processor to perform simple I/O operations. The first part of the tutorial attempts to build the hardware system. The EDK is composed of two software components: i) Xilinx Platform Studio (XPS) which is used to build and configure the soft processor system on the FPGA. ii) Xilinx Software Development Kit (SDK), which is the IDE for software development. Perform the following steps to build the hardware system.

Step 1: We start by building the hardware using the XPS. Start XPS using the program menu as shown in Figure 1.

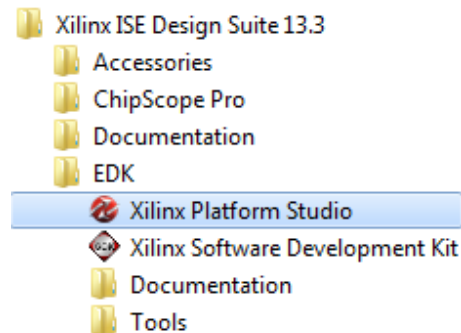


Figure 1. Starting EDK (Step 1)

When the XPS starts the main window will show the options shown in Figure 2. The “Getting Started” part gives you the options to create a new project (empty or based on a specific board), or open an old project that was created before.

Step 2: Select the option “Create New Project Base System Builder” which will allow you to build an embedded processor system based on the specification of specific board (In our case Nexys 3 board). The “New Project Wizard” will appear as shown in Figure 3.



Figure 2. XPS Window (Step 2)

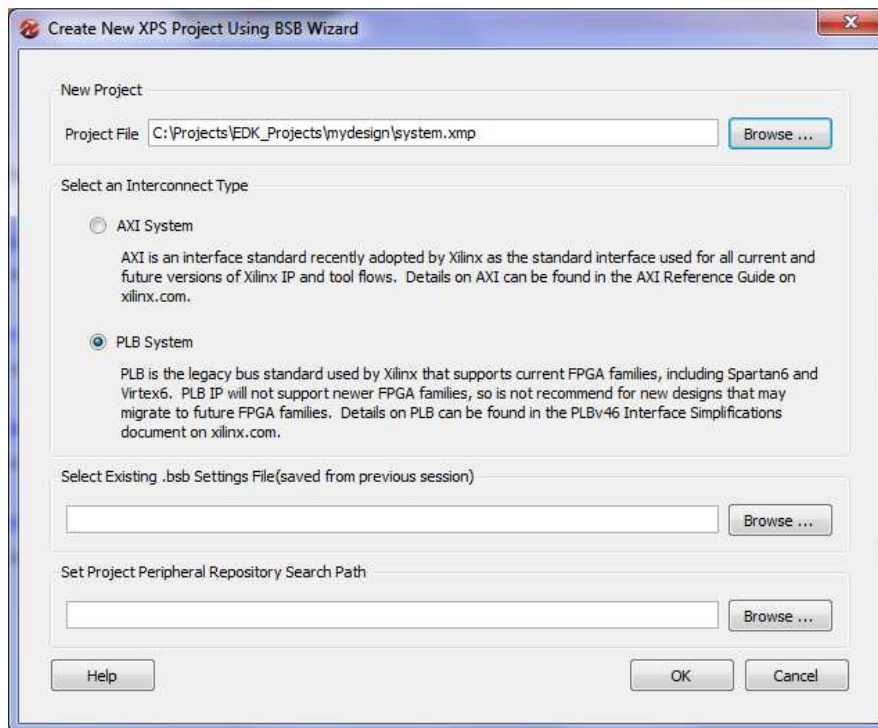


Figure 3. Building a New Design (Step 3)

Step 3: In the new project wizard window of Figure 3, select the location you want to save the files of your project. The project will include so many files in the future so try to keep this location specific and separate from other files so that you can refer back to it easily. Create a new folder on the hard drive and name it “Projects” to store all your projects, then create another folder called “EDK_Projects” to store all projects created using EDK. After that create a folder for this new project and name it for example “mydesign”. The project file (usually system.xmp) will be stored in that location (c:\Projects\EDK_Projects\mydesign\system.xmp) as shown in Figure 3.

The second option in this window is to select the interconnect type. The EDK supports two types of bus interface AXI, and PLB. Both are standard bus topology with various specifications. The differences between the two standards are beyond the scope of this article. For this experiment we will use the second option “PLB System”.

When done press “OK”. This will start the “Base System Builder” tool that will help us build and configure the hardware system as shown in Figure 4.

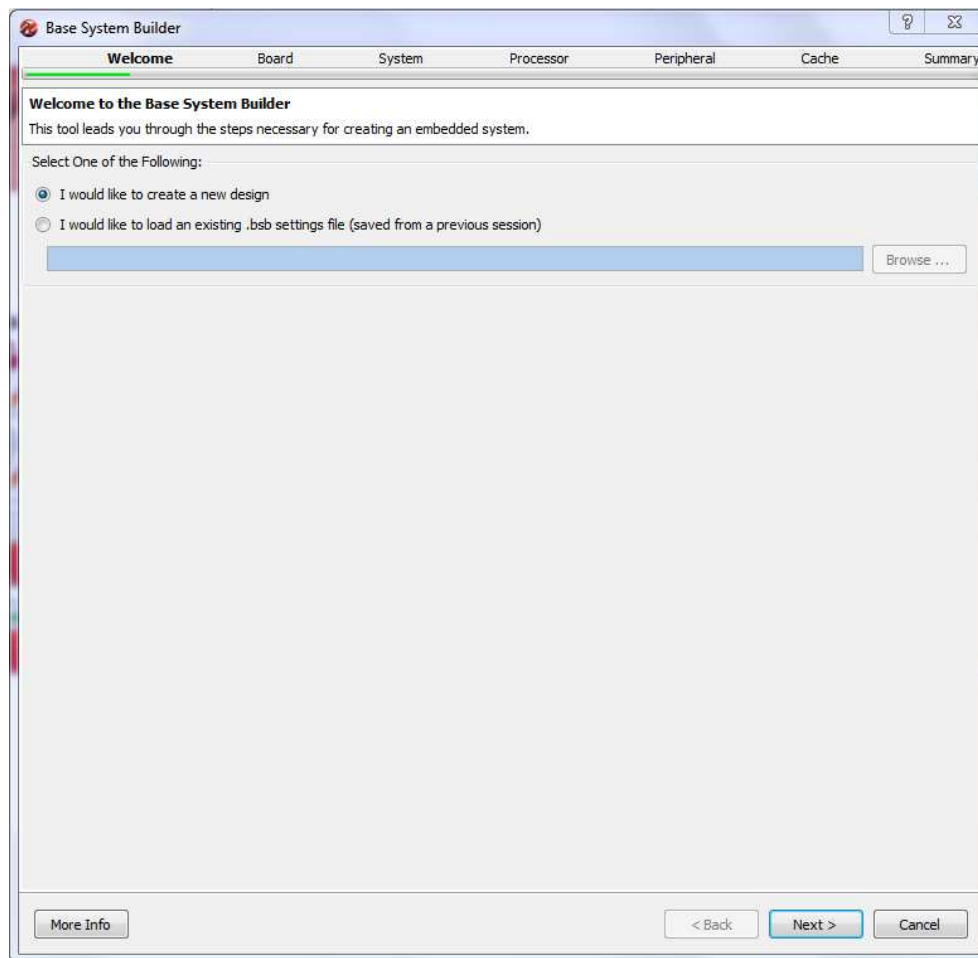


Figure 4. Base System Builder - Welcome Screen (Step 4)

Step 4: The welcome screen shown in Figure 4, is the first screen in the Base System Builder wizard that will guide you to build your system. In this screen select **“I would like to create a new design”** and then press next. This will let you create a new design.

Step 5: The second screen of the wizard is shown in Figure 5. In this screen we select the board that we want to use to build the system. The XPS allow you to build a design based on a specific board or create a custom design that is generic. Select the first option **“I would like to create a system for the following development board”**. This will allow you to select a specific board. The board is specified by three options:

- Board Vendor: this is the manufacturer of the board. Select **“Digilnet”** which is the company that builds the Nexys 3 board.
- Board Name: this is the name of the board we want to use. Select **“Nexys 3 Board”**.
- Board Version: the board version. Select version **“B”**.

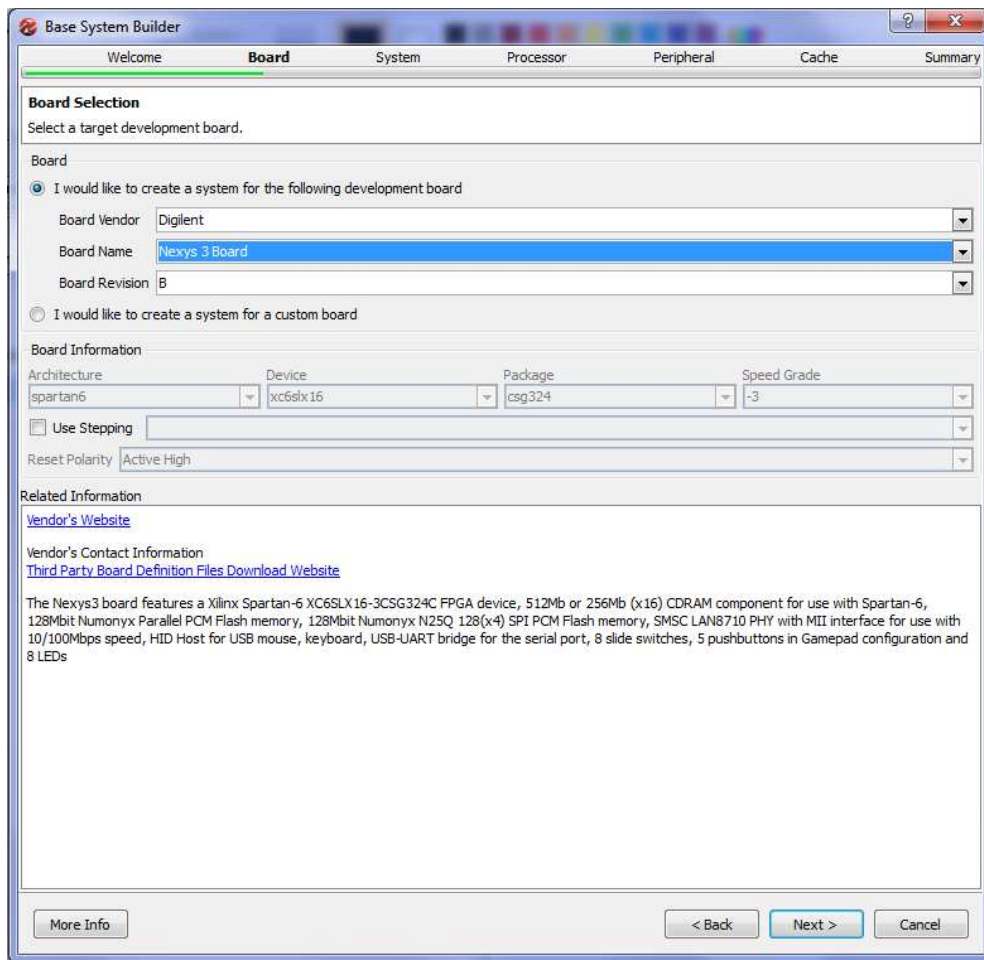


Figure 5. Board Selection (Step 5)

The Nexys 3 board is equipped with Spartan 6 FPGA chip which is a mid-size FPGA from Xilinx. The board has several peripherals that can be used with the FPGA to do several functions (See Figure 5):

- 512 MB CDRAM.
- 128 Mbit Flash Memory.
- 8 Switches and 8 LEDs.
- 5 Push Buttons.
- Ethernet Physical Interface for base 10/100 networking.
- USB Host for Keyboard and Mouse
- Serial Port

These peripherals allow you to build a small computer system on the board which we are going to do in this experiment (and the following ones).

When done selecting, press **“Next”** to move to the next screen.

Step 6: The following step is to select the system type. This is shown in Figure 6. The XPS allows us to choose between two types of system architecture:

- **Single Processor System:** This is the common system that you studied in many courses. The system is composed on one processor connected to several peripherals using a single common bus. The advantage of such system is the simplicity of the design and simplicity of programming such systems. The main disadvantage is that all peripherals are connected through a common bus, which will reduce the speed for fast peripherals to match the slower ones. This type of systems is good for applications that do not require high speed of data communication.
- **Dual Processor System:** This system architecture is composed of two processors; each is connected to a separate bus. One processor will be used for high speed peripherals and the other is used with lower speed peripherals. This increases the performance of the system compared to the single processor system. However, writing software for this system is more complex as it requires synchronization between the two processors.

Select **“Single Processor System”**. This will start building a single processor system. We choose to build a single processor system as this is simple to develop. However, you can later add more processors if required. When done press **“Next”** to move to the next screen shown in Figure 7.

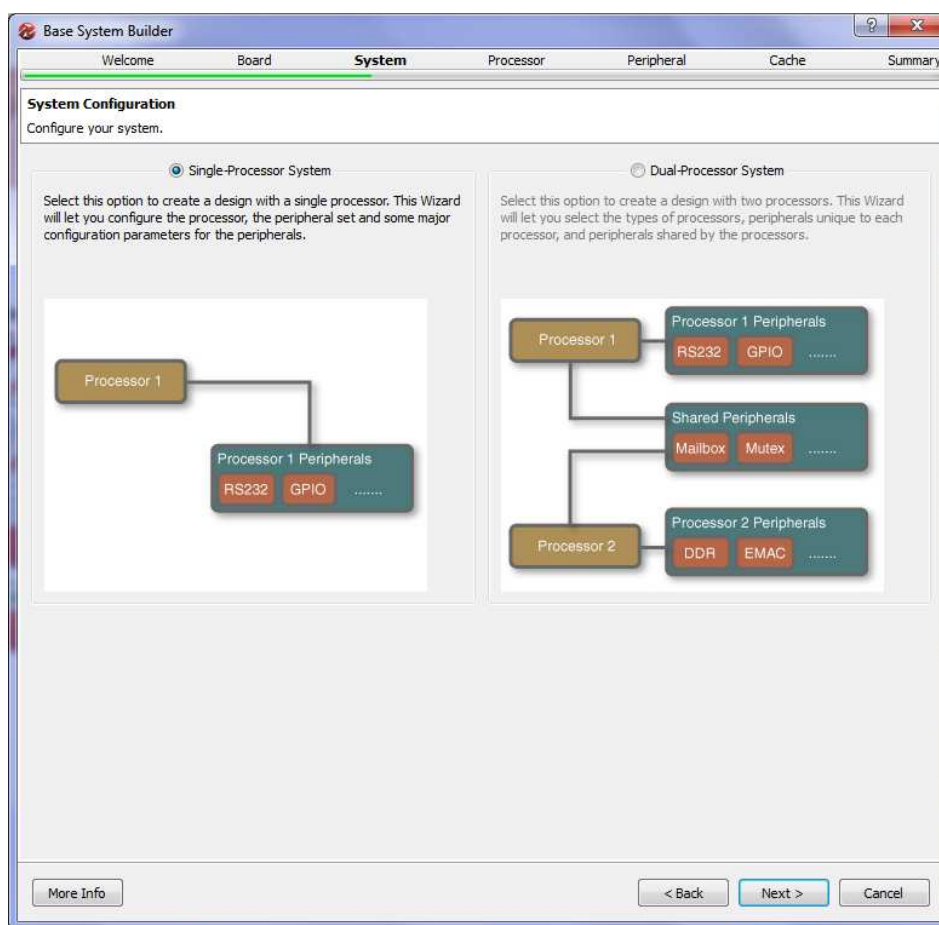


Figure 6. Select System Architecture (Step 6)

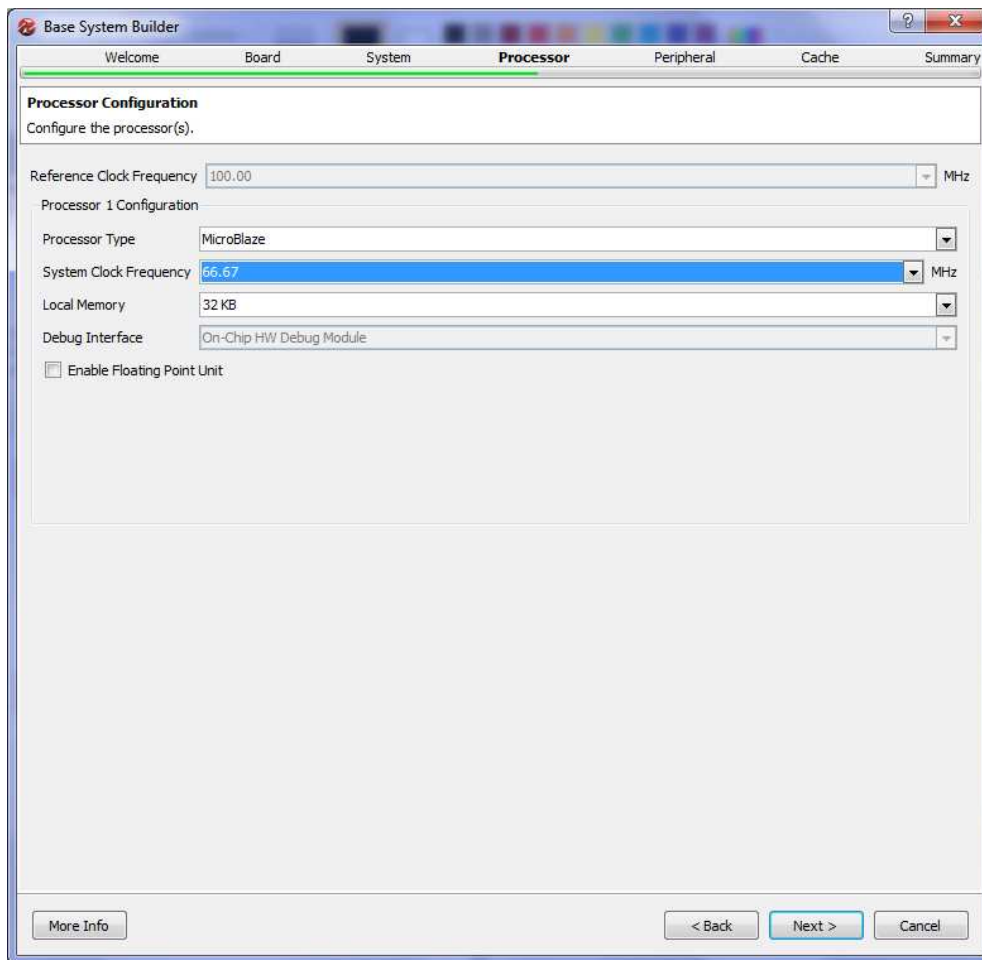


Figure 7. Processor Configuration (Step 7)

Step 7: The processor configuration screen is shown in Figure 7. As stated earlier, Xilinx provides a processor called MicroBlaze that can be implemented on its FPGAs. In this screen we configure the processor as follows:

- Processor Type: There is only one selection which is MicroBlaze.
- System Clock Frequency: This is the processor and common bus reference clock frequency. Most of the modern FPGA's are equipped with clock modules that are able of generating higher frequencies from a single fixed frequency. For example, Nexys 3 board has on board clock of 50 MHz, however, you can use higher frequencies using the FPGA clock modules. Select the desired frequency up to 83 MHz Select **"66.67 MHz"**.
- Local Memory: The FPGA contains several memory blocks up to 128 KB. This is different from the on board memories. You can connect up to 64KB of these block memories to the MicroBlaze as a local processor memory. You can connect more memory if required through the local bus. Select **"32KB"** which is enough for our project.
- Leave all other options and then press **"Next"** to move to the next scree.

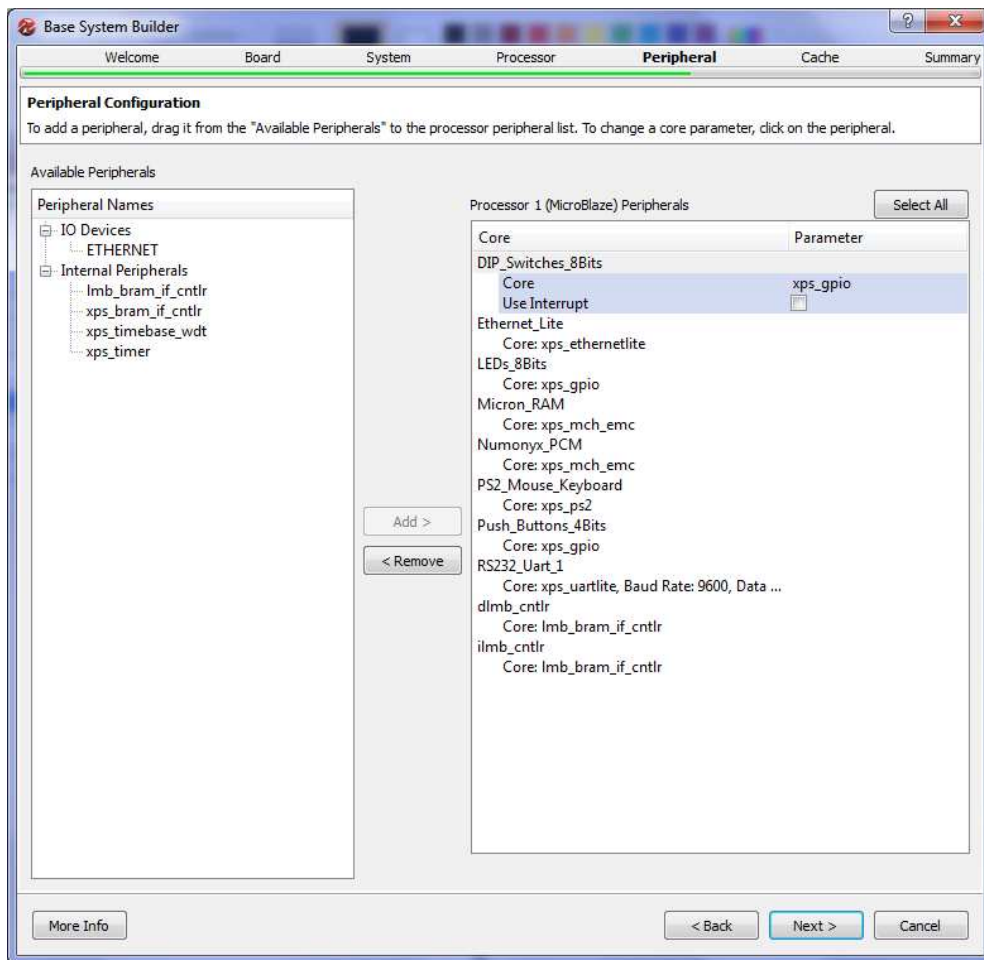


Figure 8. Peripheral Configuration (Step 8 - 1)

Step 8: After configuring the processor we are ready to configure the peripherals (Input/Output devices). The peripheral configuration screen is shown in Figure 8. In this screen a list of all the available peripherals is presented. From this list the designer can choose which peripheral to be connected to the processor. The screen is divided into two lists. The right-hand side list shows the peripherals that are not connected to the processor, while the left-hand side list shows all the peripherals that are currently connected to the processor. By default several peripherals will be connected to the processor as shown in Figure 8. In this project we do not need many of these devices so we will remove many of them as shown in Figure 9. Remove the following devices:

- PS2_Mouse_Keyboard
- Micron_RAM
- Numonyx_RCM

To remove a device, select the device then press **“Remove”**.

This leaves the Switches, Leds, Ethernet , Push Buttons, and Local Memory devices connected to the processor.

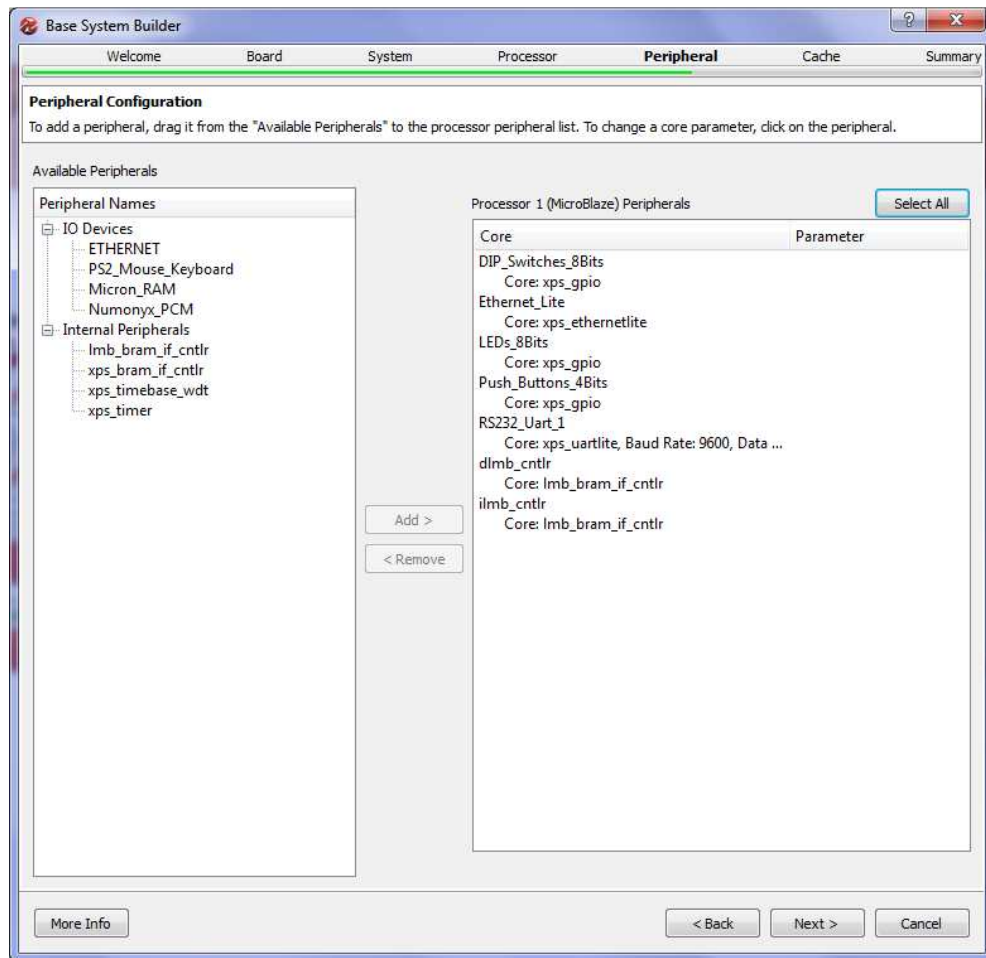


Figure 9. Peripheral Configuration (Step 8 - 2)

To add a device to the processor connection, select the required device and click add. Note that some devices may be implemented using different hardware. Example is the Ethernet Physical Layer. It is shown in the right-hand side as **“Ethernet_Lite”** and shown in the left-hand side as **“ETHERNET”**. The difference between the two is beyond the scope of this tutorial, so we will just use the module **“ETHERNET”** by selecting it and press **“Add”**, doing so will show a dialog box as shown in Figure 10. This dialog asks to replace **“Ethernet_Lite”** with **“ETHERNET”**. Press **“Yes”** to do the replacement.

Now we selected all the required devices. We have some input/output devices connected to the processor. After completing the hardware, we will write a software application to read from the switches and write to the LEDs.

Press **“Next”** to move to the next screen.

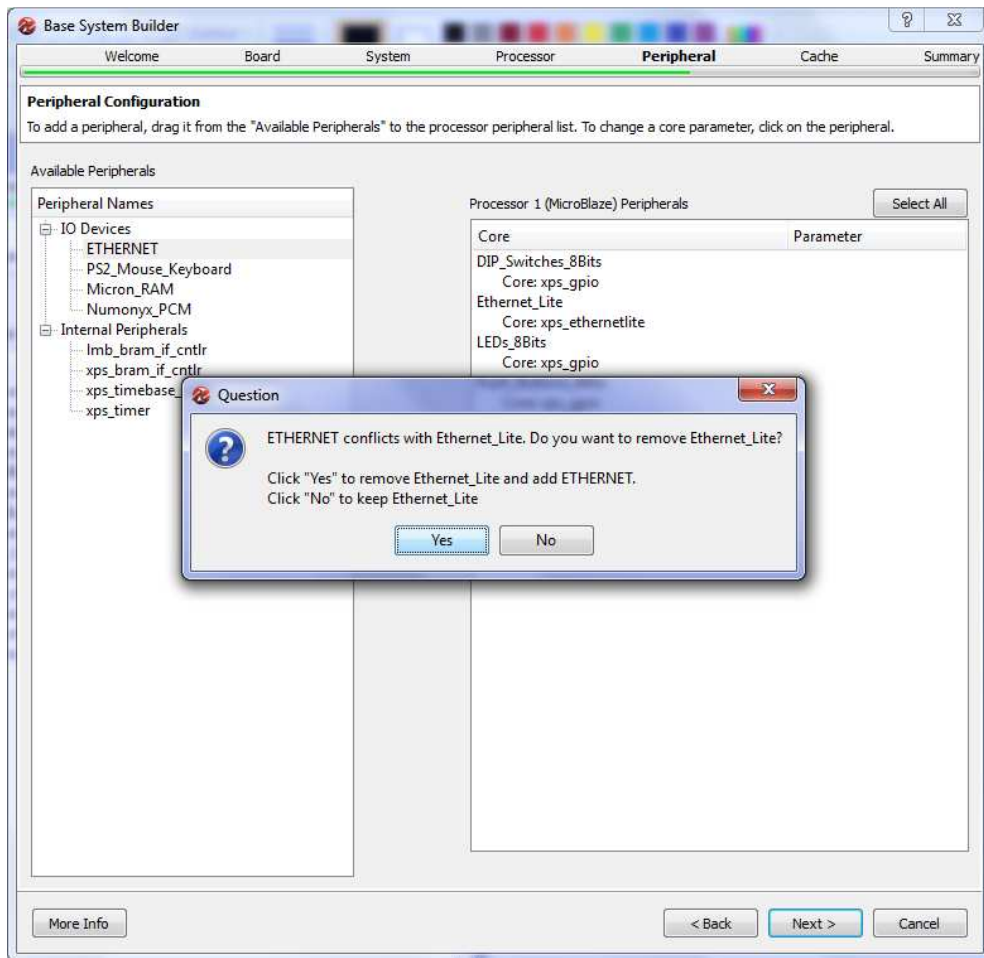


Figure 10. Peripheral Configuration (Step 8 - 3)

Step 9: Cache configuration screen is shown in Figure 11. If we have more than one memory types connected to the processor, we can use cache memory to speed up memory access. As we only have local memory connected to the processor, no cache memory can be configured. We will leave this screen unchanged and press “Next”.

Step 10: Now we are done configuring the hardware. The last screen of the “Base System Builder” is shown in Figure 12. This screen displays a summary of the system being built. Two types of information are displayed; system components and file allocation. The components list displays the name of each component and the address associated with it. The address assigned to each peripheral is unique and it is used by the processor to locate a specific device and communicate with it. For example, the device DIP_Switches_8Bits (the 8 switches on the board) is assigned the address 0x81440000 which means that the processor will use this address to read the value of the switches as digital input.

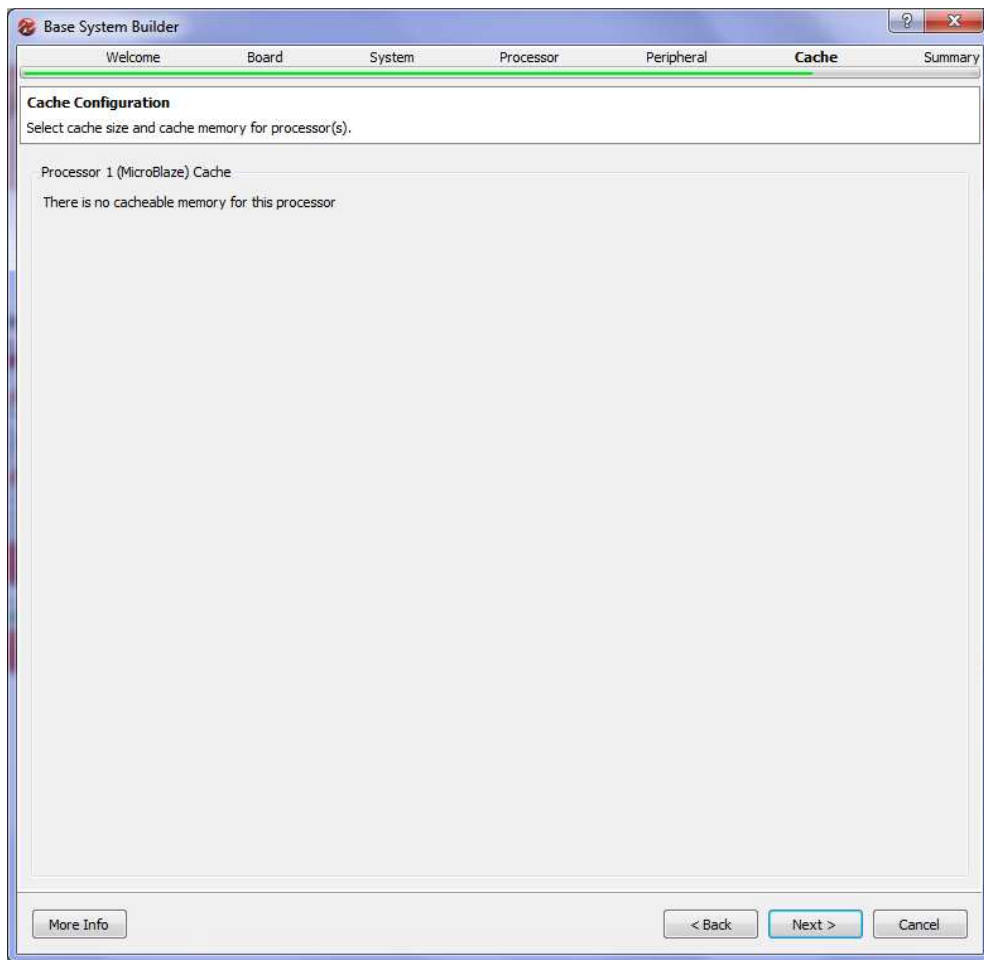


Figure 11. Cache Configuration (Step 9)

The file list includes all the files that are created by wizard to define the project. The wizard generates six files.

- System.xmp: this is the project file that is used by XPS to open the project for future use.
- System.mhs: this is the hardware description file. This is a text file that describes the hardware components of the system.
- System.ucf: user constraints file. This file defines the relation between the system input/output pins and the actual FPGA pins.
- Fast_runtime.opt, download.cmd, bitgen.ut: these three files are used build the bit file that is later downloaded to the FPGA to configure it to do the function of the system.

This is the final screen of the wizard; clicking **“Finish”** will end the wizard and generate the required files on the folder specified at the beginning of the wizard. Click **“Finish”** and then check the folder **“C:\Projects\EDK_Projects\mydesign”** it should look as shown in Figure 13.

When the **“Base System Builder”** is done the XPS window should look like Figure 14.

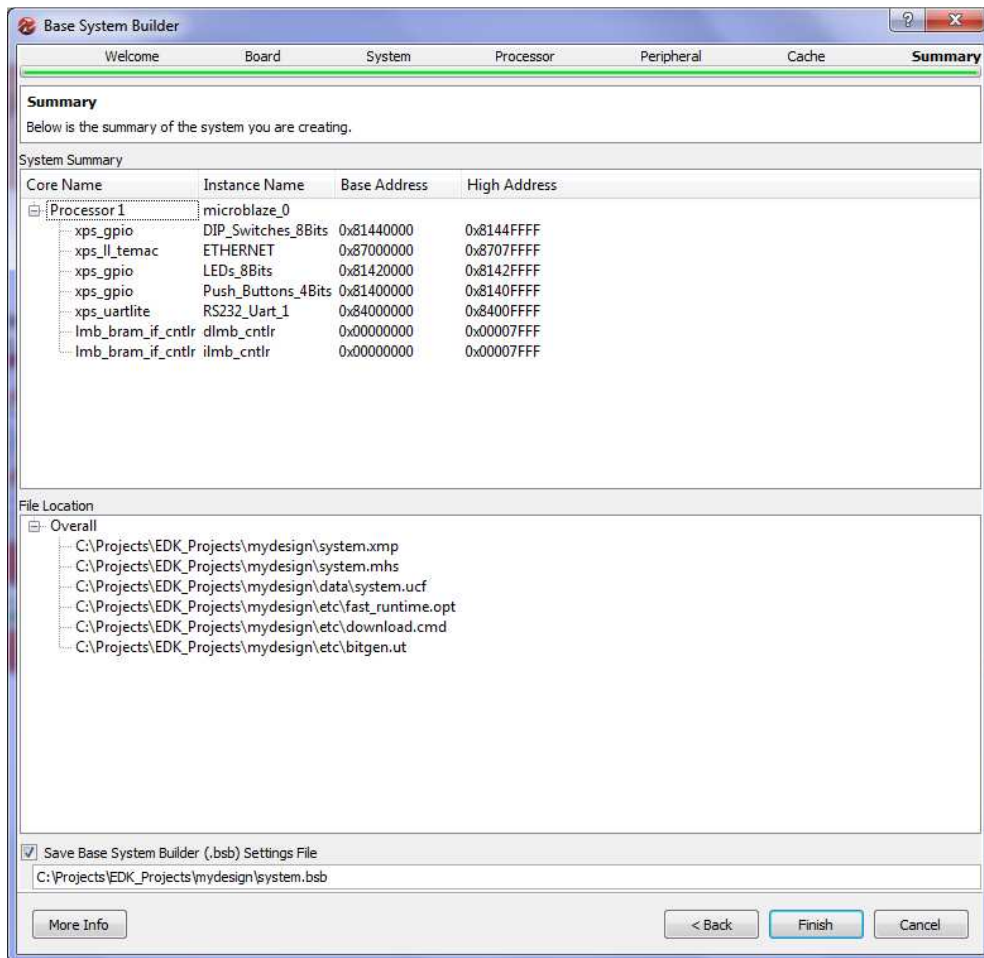


Figure 12. System Summary (Step 10)

__xps	05/03/2012 10:05 ...	File folder
data	27/02/2012 12:57 ...	File folder
etc	28/02/2012 3:00 PM	File folder
implementation	27/02/2012 1:05 AM	File folder
pcores	27/02/2012 12:00 ...	File folder
clock_generator_0	27/02/2012 12:57 ...	Text Document
system.bsb	27/02/2012 1:05 AM	BSB File
system	05/03/2012 10:05 ...	Text Document
system.make	05/03/2012 10:05 ...	MAKE File
system	27/02/2012 1:05 AM	MHS File
system	27/02/2012 1:05 AM	Xilinx Platform Stu...
system_incl.make	05/03/2012 10:05 ...	MAKE File
XpsGuiSessionLock	05/03/2012 10:05 ...	File

Figure 13. Directory Structure

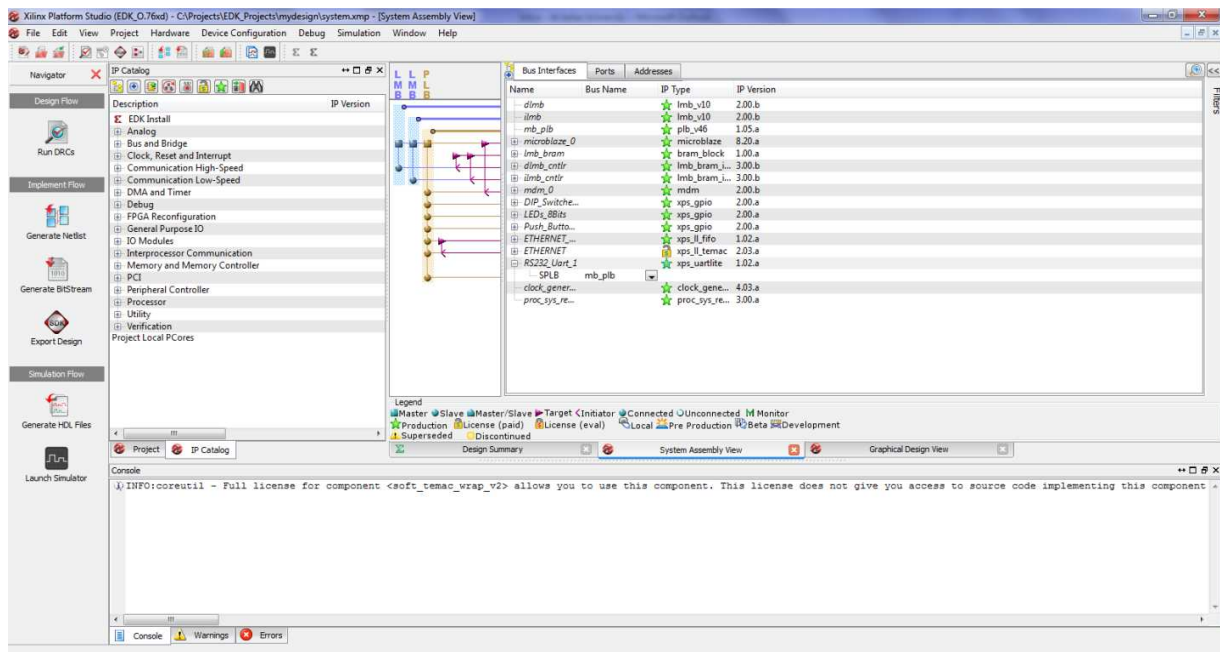


Figure 14. Xilinx Platform Studio - System Assembly View

Step 11: The XPS window (Figure 14) opens at the “System Assembly View” which shows the system components and their interconnection. The system assembly view has three tabs:

- **Bus Interface:** this shows the bus interconnection between components. As you can see there are three main buses:
 - ilmb : instruction local memory bus, used to connect the processor to the code memory. Code memory is the memory that holds the programs code (instructions).
 - dlmb: data local memory bus, used to connect the processor to the data memory. Data memory is the memory that holds the program data.
 - mplb: MicroBlaze peripheral local bus (plb), used to connect the processor to all the other peripherals.
- **Ports:** this tab lists all the ports of each peripheral/processor and displays its connection. In this project we will not need to modify this.
- **Addresses:** this tab lists the addresses assigned to each peripheral. You can modify these addresses through this tab. But for this project we will not do any address modification.

In the “Bus Interface” tab you can modify the system by adding more peripherals/processors, removing peripherals, or changing the configuration of any peripheral. To illustrate this, we will delete the peripheral “Push_Buttons_4Bits”:

- Right click the peripheral “Push_Buttons_4Bits” a menu will pop up as shown in Figure 15. Select “Delete Instance”.
- A dialog box will show up with four options. Select the first option “Delete instance and all its connections” and press “OK”. This will remove the instance from the system.

- In the project tab in the middle of the XPS window, locate the file “data/system.ucf” and open it. Locate the following four lines and delete them:

```
Net fpga_0_Push_Buttons_4Bits_GPIO_IO_I_pin<0> LOC=D9 | IOSTANDARD = LVCMOS33;
Net fpga_0_Push_Buttons_4Bits_GPIO_IO_I_pin<1> LOC=C9 | IOSTANDARD = LVCMOS33;
Net fpga_0_Push_Buttons_4Bits_GPIO_IO_I_pin<2> LOC=C4 | IOSTANDARD = LVCMOS33;
Net fpga_0_Push_Buttons_4Bits_GPIO_IO_I_pin<3> LOC=A8 | IOSTANDARD = LVCMOS33;
```

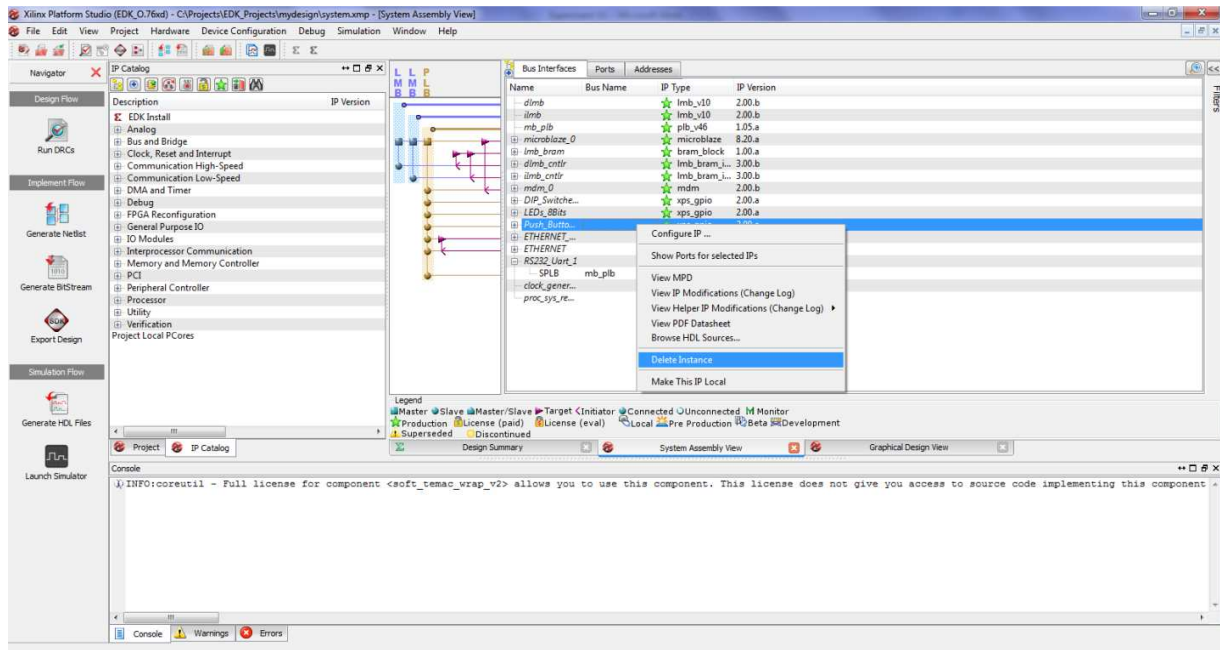


Figure 15. Deleting Peripheral – 1

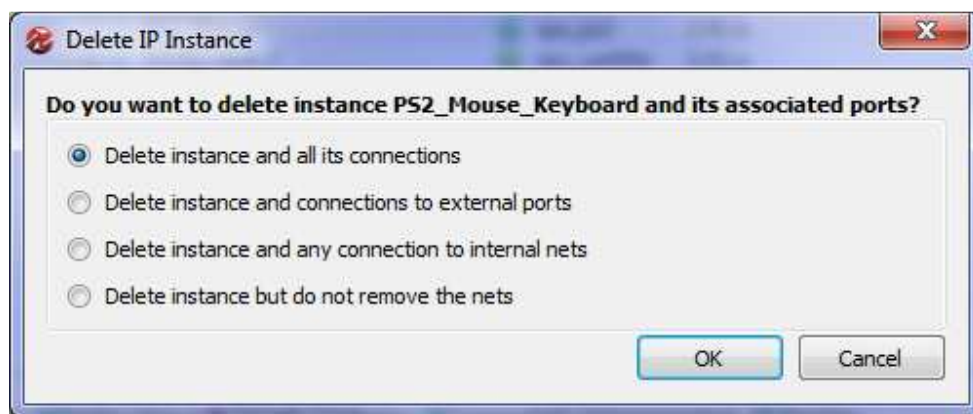


Figure 16. Deleting Peripheral – 2

Step 12: The last step to build the hardware is to generate the bit-file that is used to configure the FPGA. XPS generates all the necessary files to do so. Each component of the system is defined in VHDL (or other hardware language) and the tool will compile all these files to build a single design file to be downloaded on the board. This process is composed of several tasks:

- **Synthesis:** the compilation process. It will compile all hardware description language (HDL) files of the system and check for any errors. Synthesis will convert HDL into logical blocks.
- **Mapping:** transform the logical blocks generated from synthesis into FPGA blocks.
- **Placement:** specify each component of the system and its location on the FPGA.
- **Routing:** connect all components of the system on the FPGA.
- **Generate bit-file:** the result of all the previous steps is stored in a single configuration file to be downloaded on the FPGA.

The navigator tool-bar is displayed on the left hand side of the XPS window as shown in Figure 17.

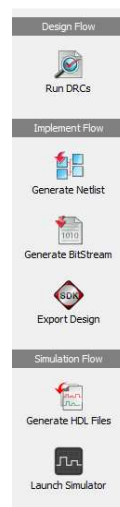


Figure 17. Flow Navigator

In the navigator toolbar under the implementation flow you will find a button labeled **“Generate BitStream”**. Click this button to start the hardware building process which will take some time to perform all the tasks listed before. A lot of information will be displayed in the **“Console”** window in the bottom of the XPS window. Watch the progress of the work until the **“Console”** window displays the message shown in Figure 18. This message states that the tool has created the file **“System.bit”** which contains all the required configuration bits. At this point we are done creating the hardware. The hardware will have no function without software running on the processor to do some function. This what will be done in the second part of the experiment.

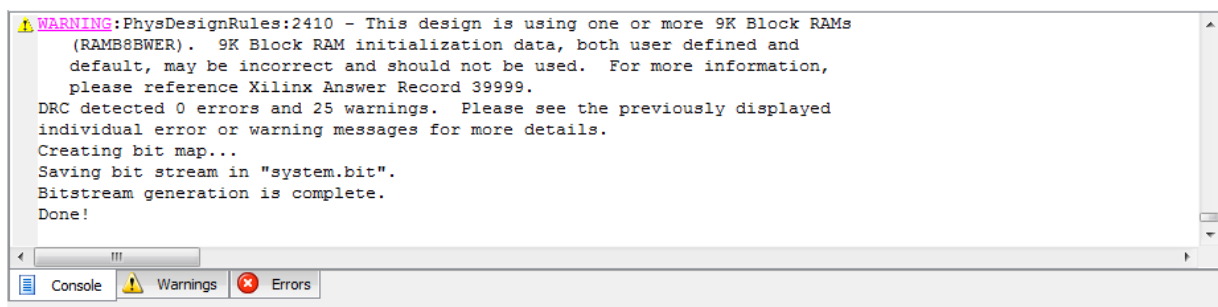


Figure 18 Hardware Build Done

Part 2: Software Development

In the first part of the tutorial we were able to build the hardware part of the system. In the second part we will write a simple C application and compile it for MicroBlaze processor that we built in part one. The process of building software application and running it is composed of the following tasks:

- Create a software workspace
- Create a board specification project to link the hardware to the software platform
- Create C project and add a source file to it
- Compile the C project to create ELF (embedded executable file)
- Merge the ELF file to the bit-stream created in part 1 (system.bit) to generate a complete bit file that contains both hardware and software (download.bit).
- Download the final bit file (download.bit) to the FPGA to run the application on the processor system.

Step 1: We perform all these tasks using another tool called Xilinx Software Development Kit (SDK). We start the SDK using the button “**Export Design**” in the Navigation toolbar. This button copies the bit-file (system.bit) and some other files necessary to link the hardware to a directory called “**SDK\SDK_Export**” under the projects folder. Then it will ask if you want to start the SDK. Click “**Export Design**” which starts the window shown in Figure 19. In which you have two options; “**Export Only**” to just copy the hardware file (used when the SDK is already running) or “**Export & Launch SDK**” which will copy the files and starts the SDK; click this button to start the SDK.



Figure 19. Export to SDK (Step 1)

Step 2: When the SDK starts the work space launcher window shown in Figure 20. The workspace as a software environment which is used to collect several projects in one entity. As the SDK is based on eclipse platform, those who are familiar with eclipse will find it easy to deal with the SDK. Change the workspace path as shown in Figure 20.

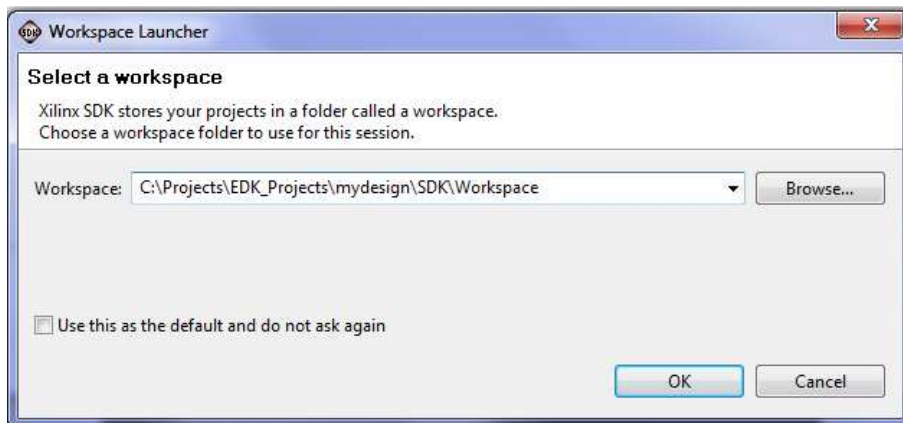


Figure 20. Workspace Launcher (Step 2)

Step 3: The SDK main window will start then as shown in Figure 21. The SDK IDE development area is divided into several areas. The left-hand side is the Project Explorer window which shows one project “mydesign_hw_platform” which represents the hardware created in part one. You will notice it includes the file “system.bit”. The middle area is the code area. By default the file “system.xml” which lists the components of the system and information about each of them.

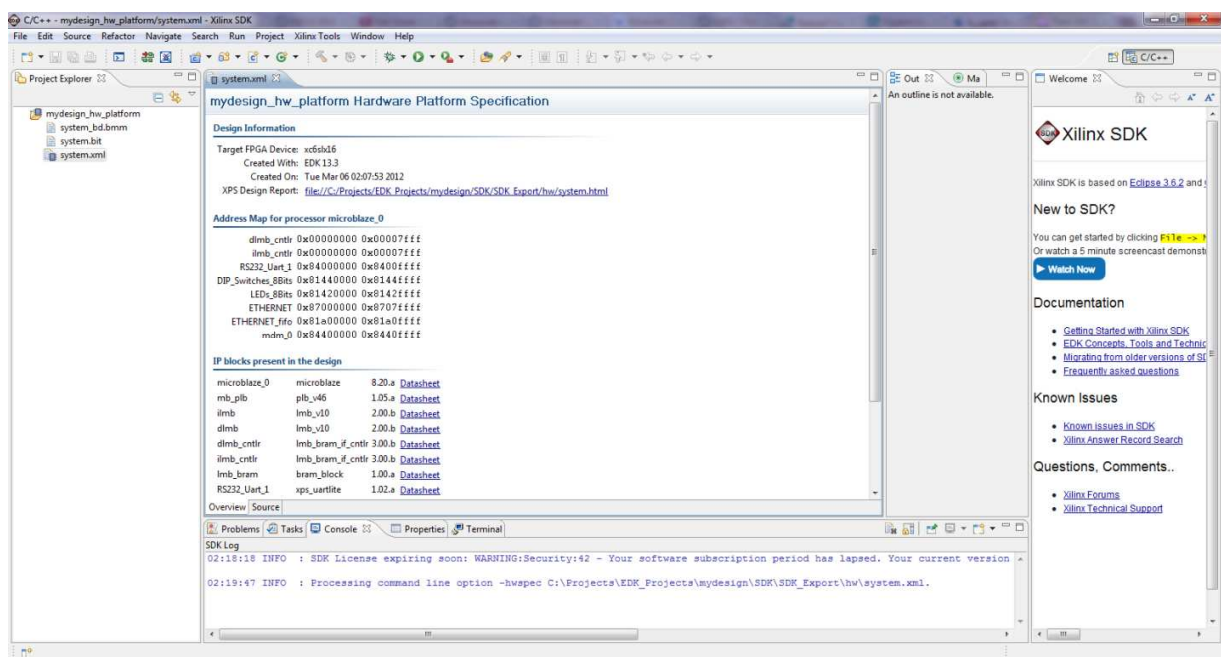


Figure 21. SDK Main Window (Step 3)

Step 4: The following step is that we create a C project using the SDK. In the project explorer window, right click the window a menu will show up; select New->Project. A new project window will appear as shown in Figure 22. A list of projects that can be created is shown. Select “Xilinx C Project” which will start a Wizard to create a simple C program. Select “Xilinx C Project” then click “Next” to continue.

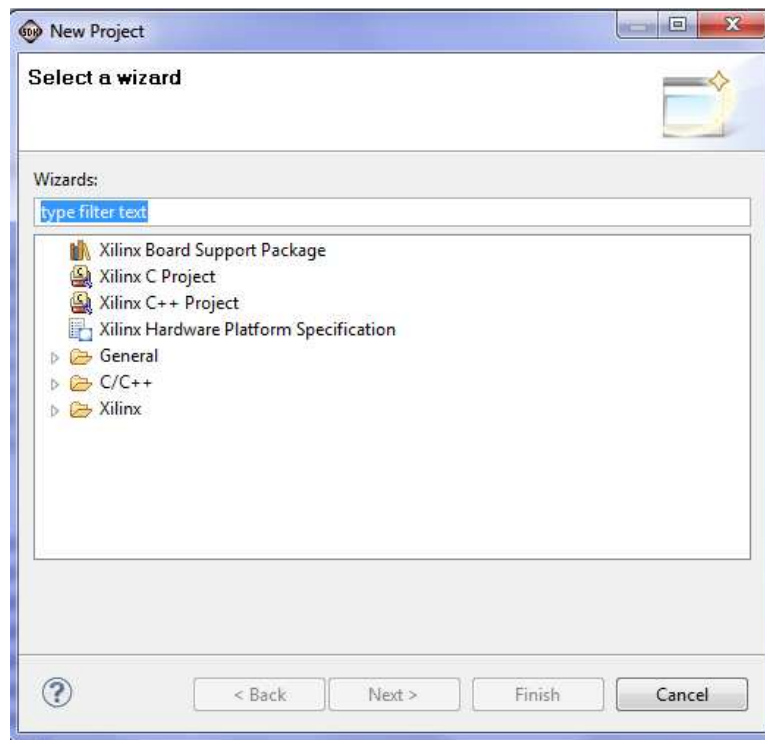


Figure 22. New Project (Step 4)

Step 5: The second screen of the New Project Wizard is shown in Figure 23. This screen allows you to select a template for the project you want from a predefined list. Select **“Hello World”** which is a template for a simple application that will send some text through the serial port. We will modify this application later. Select **“Hello World”**, keep everything else unchanged, and then press next.

Step 6: The last screen of the new project wizard is shown in Figure 24. In this screen we create another project called the **“The Board Support Package Project”**. Each component of the hardware is associated with a software program that is responsible for operating the device called the driver. The source of each driver is copied from the EDK directories to Workspace to be compiled with the project. We will use these drivers to access the hardware, so it is important to create the **“Board Support Package Project”**. This screen, Figure 24, creates that project and names it **“Hello_World_bsp0”**.

Press **“Finish”** to end the wizard and create the C project **“hello_world_0”** and **“The Board Support Package Project”** named **“Hello_World_bsp0”**. The Project Explorer Window should look like Figure 25. Three projects will be listed in the explorer.

Step 7: Expand the folder **“src”** in the **“hello_world_0”** project. You will notice several C files. Double click the file **“hello_world.c”** to open it. The code should look like Figure 26. The code is a simple C application that will print the word **“Hello World”** on the output device. We will modify this code now to read from the switches and put the value on the LEDs.

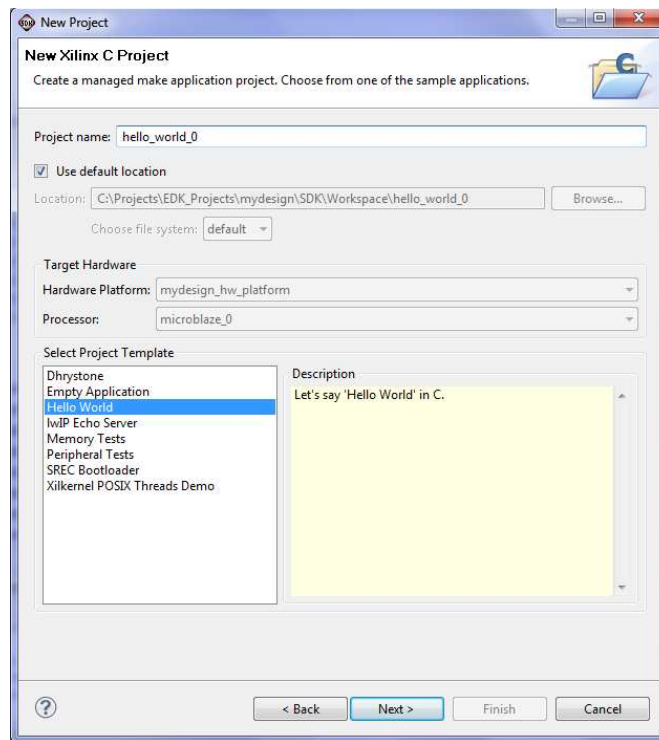


Figure 23. New Project (Step 5)

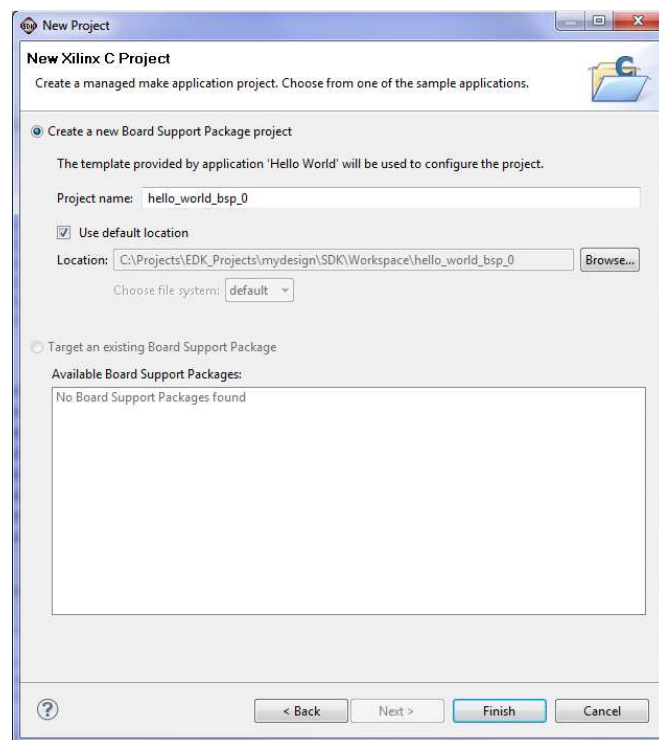


Figure 24. New Project (Step 6)

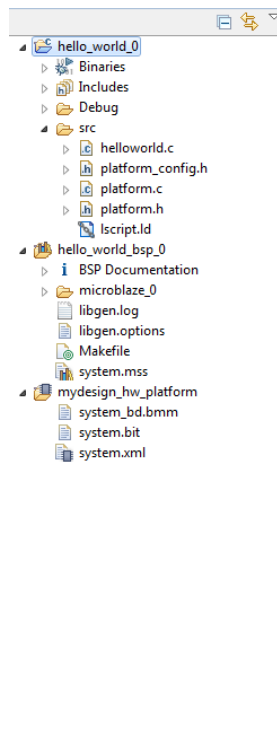


Figure 25. Project Explorer (Step 6)

```
#include <stdio.h>
#include "platform.h"

void print(char *str);

int main()
{
    init_platform();

    print("Hello World\n\r");

    cleanup_platform();

    return 0;
}
```

Figure 26. Initial Hello World “Source Code” (Step 7)

Step 8: Modify the source code as shown in Figure 27 . When done save the file. Note that once you are done, the SDK will start compiling your code automatically and generate the output files (in this case it is the download.bit file). This code performs the following tasks:

- Initialize the drivers for both the switches and LEDs.
- Starts an infinite loop
- Reads the input port connected to the switches and writes it back to the LEDs.

```

#include <xparameters.h>
#include <xgpio.h>

#include "platform.h"

int main()
{
    init_platform();

    char swval = 0;    // a variable to store the switch value

    XGpio dipsw, leds; // GPIO Access Variables these are used by the driver
                      // dipsw = Switches variable
                      // leds = LEDs variable

    XGpio_Initialize (&dipsw, XPAR_DIP_SWITCHES_8BITS_DEVICE_ID); // Driver initialisation for
                                                                // switches variable
    XGpio_Initialize (&leds , XPAR_LEDS_8BITS_DEVICE_ID);        // Driver initialisation for
                                                                // LEDs variable

    XGpio_SetDataDirection (&leds , 1, 0);                       // Set LEDs direction as output
    XGpio_SetDataDirection (&dipsw , 1, 0xFFFFFFFF);            // Set switches direction as input

    while (1)                                                    // Loop forever
    {
        swval = XGpio_DiscreteRead(&dipsw,1);                    // Read the value of the switches
        XGpio_DiscreteWrite(&leds,1,swval);                      // Write the switches value to the LEDs
    }

    cleanup_platform();|
    return 0;
}

```

Figure 27. Modified Source Code (Step 8)

Those who worked with micro-controllers will notice great similarity with programming micro-controllers. When done modifying the code and saving, the SDK will compile the code automatically and link it to the hardware. You will be able to download the design on the FPGA at this point.

Step 9: The final step is to download the design on FPGA. Select the menu item **“Xilinx Tools”** from the menu bar and then select **“Program FPGA”**. The **“Program FPGA”** window of Figure 28 will appear asking you to select the executable (ELF) file to be used. Beside **“microblaze_0”** select the second option **“..\mydesign\SDK\Workspace\hello_world_0\Debug\hello_world_0.elf”**, then press **“Program”**. This will start the programming process on the board. Of course you need to connect the board at this point for the programming process to complete. As soon as the programming process complete the system will start working and you will notice the output of the system.

Questions:

- 1- What is the output of the system?
- 2- How can you change the output to display some moving patterns on the LEDs?

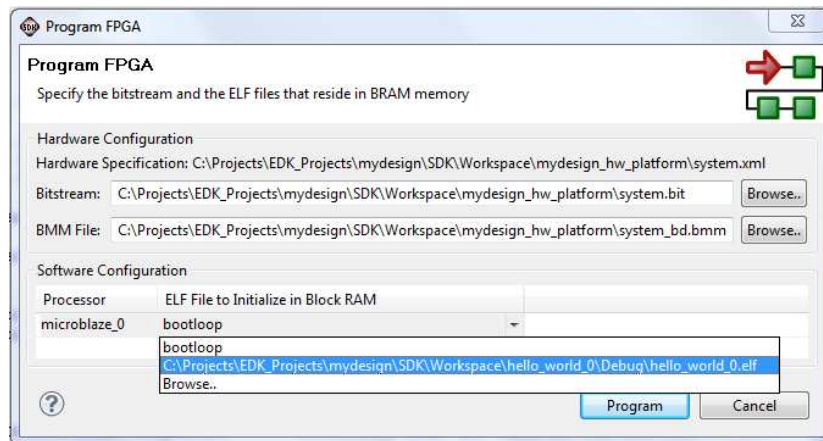


Figure 28. Program FPGA (Step 9)

Notes:
