Lab 3 - PowerPC Processor

Adding Custom IP to an Embedded System Lab:

Creating and Adding Custom IP to an Embedded System Lab: PowerPC Processor

Introduction

This lab guides you through the process of creating and adding a custom OPB peripheral to a processor system by using the Create and Import Peripheral Wizard.

Objectives

After completing this lab, you will be able to:

- Create an IP
- Add the custom IP to a real system, develop an application, and generate a bitstream
- Download the bitstream and verify the functionality in an actual hardware

Procedure

The purpose of this lab exercise is to complete the hardware design started in Lab 1 and extended in Lab 2. Lab 1 included the PPC, PLB bus, JTAG_PPC, proc_Sys_Reset, DCM, PLB2OPB, RS232_Uart_1, PLB RAM controller, and PLB BRAM components. Lab 2 added the remaining IP, except for an MYIP instance for the LED, to extend the hardware design.

In this lab, you will use the Create and Import Peripheral Wizard of Xilinx Platform Studio (XPS) to create a user peripheral from an HDL module, add an instance of the imported peripheral, and modify the system.ucf file to provide an interface to the on-board LED module.

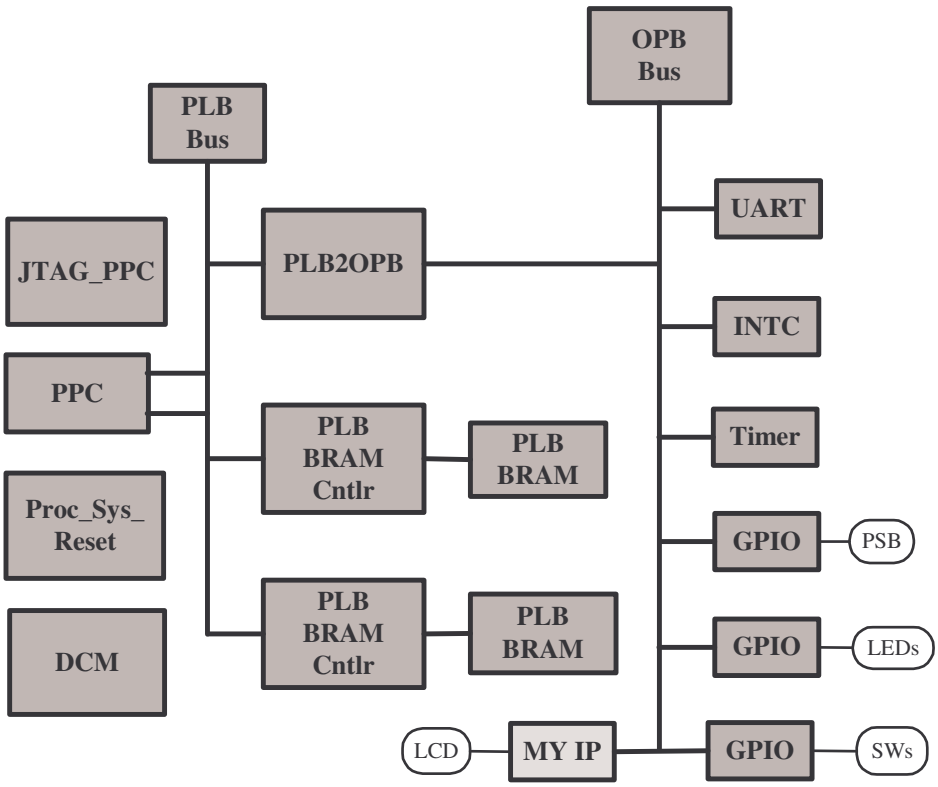


Figure 3-1. Completed Design

This lab comprises several steps involving the creation of OPB custom IP (a simple 4-bit output to drive LEDs) and addition of a custom OPB peripheral. Although the change to the hardware is simple, this lab illustrates the integration of a user peripheral through the Create and Import Peripheral Wizard. This lab also illustrates the use of an existing peripheral to provide the OPB bus interface.

Below each general instruction for a given procedure, you will find accompanying step-by-step directions and illustrated figures providing more detail for performing the general instruction. If you feel confident about a specific instruction, feel free to skip the step-by-step directions and move on to the next general instruction in the procedure.

Creating a Custom IP — Step 1

Create a *lab3* folder and copy the contents of the *lab2* folder into the *lab3* folder if you wish to continue with the design you created in the previous lab. Launch the **Create – Import Peripheral** wizard. Name the peripheral as **my_led** and let it be for OPB bus.

❶ If you wish to continue using the design that you created in Lab 2, create a *lab3* folder in the *C:\xup\embedded\ppc\labs* directory and copy the contents from *lab2* to *lab3*

❷ Open **XPS** by clicking **Start → Programs → Xilinx Platform Studio 7.1i → Create-Import Peripheral**

❸ Click **Next** to continue

❹ In the **Select Flow** panel, select **Create templates for a new peripheral** and click **Next**

❺ In the **Repository or Project** panel, select **To an existing XPS project**, browse to **C:\xup\embedded\ppc\labs\lab3** and click **Next**

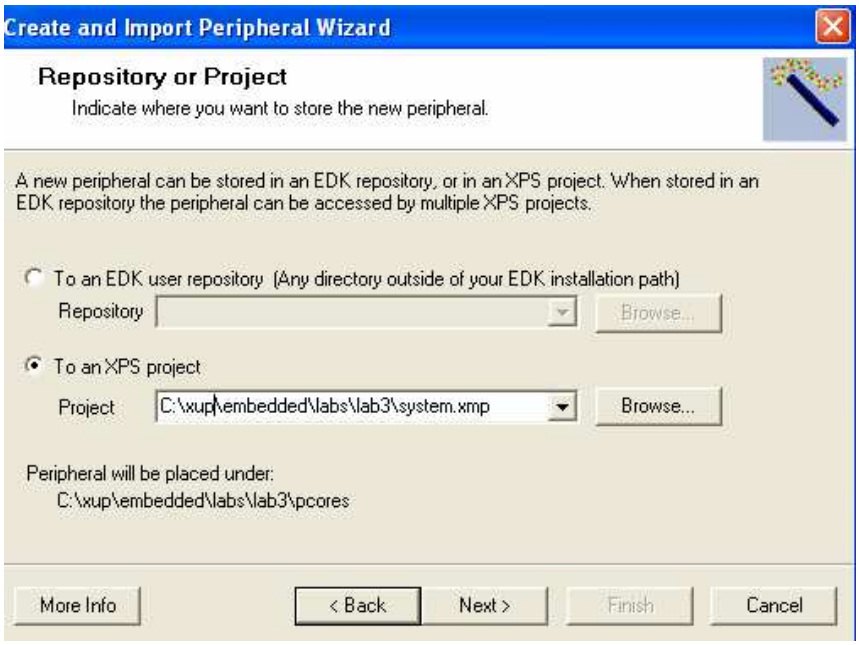


Figure 3-2. Repository or Project Dialog Box

❻ In the **Name and Version** panel, enter **my_led** as the peripheral name, accept default versions, and click **Next**

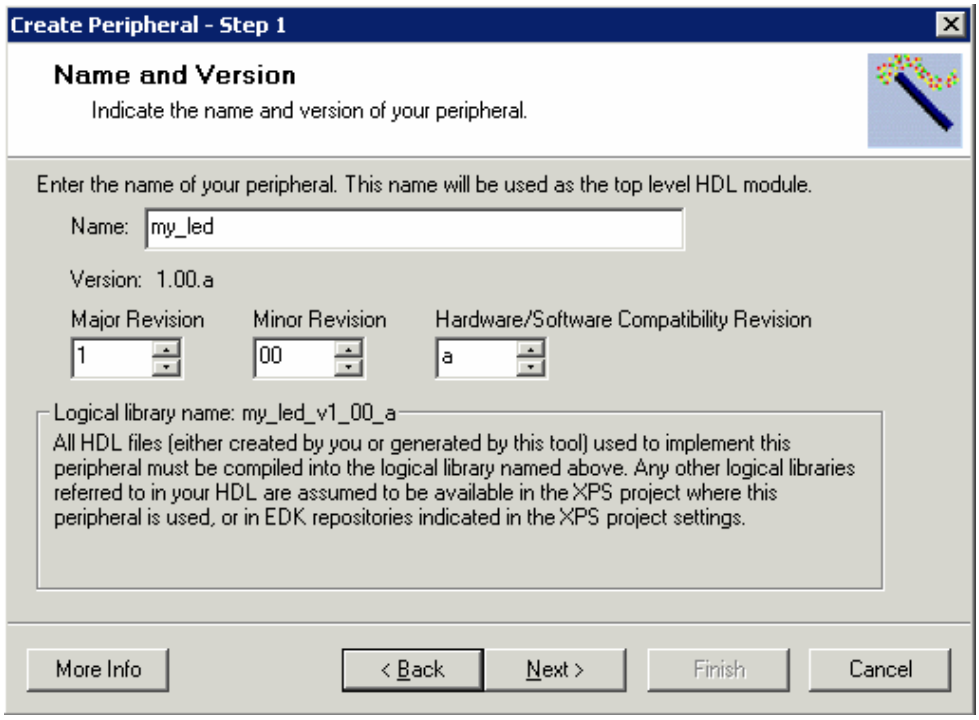


Figure 3-3. Name and Version Dialog Box

❼ In the **Bus Interfaces** panel, select **On-chip Peripheral Bus (OPB)**, and click **Next**

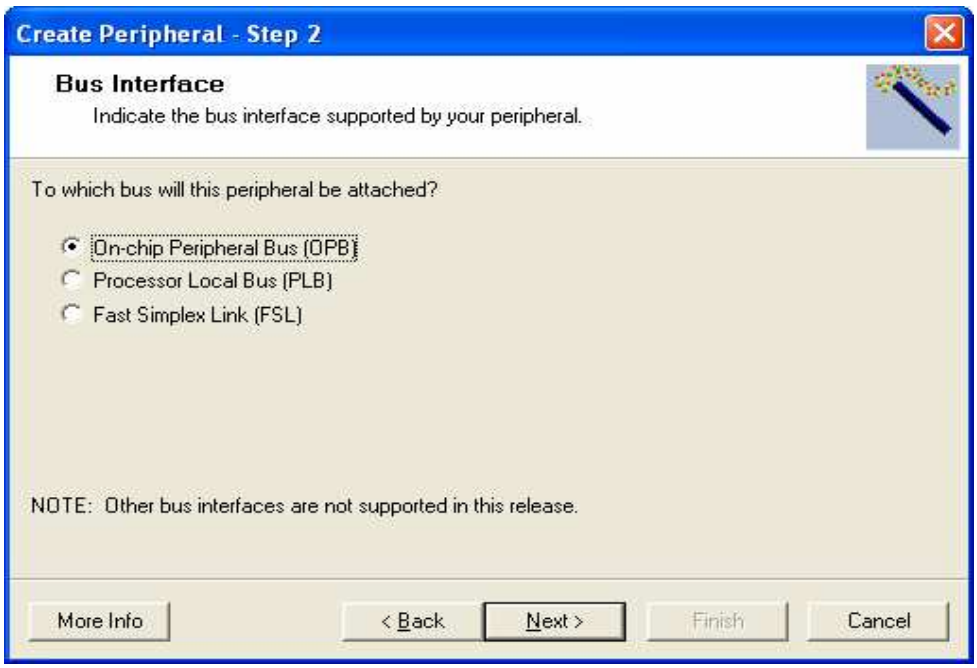


Figure 3-4. Bus Interface Dialog Box

Continuing with the wizard, select **RST/MIR** and **User Logic S/W Register** support. Select only **one** software accessible register of **32**-bit width. Generate template driver files. Browse to the **C:\xup\embedded\ppc\labs\lab3** directory and answer the questions at the end of this step

❶ In the **IPIF Services** panel, select **S/W Reset and Module Information register (RST/MIR)** and **User Logic S/W Register Support**

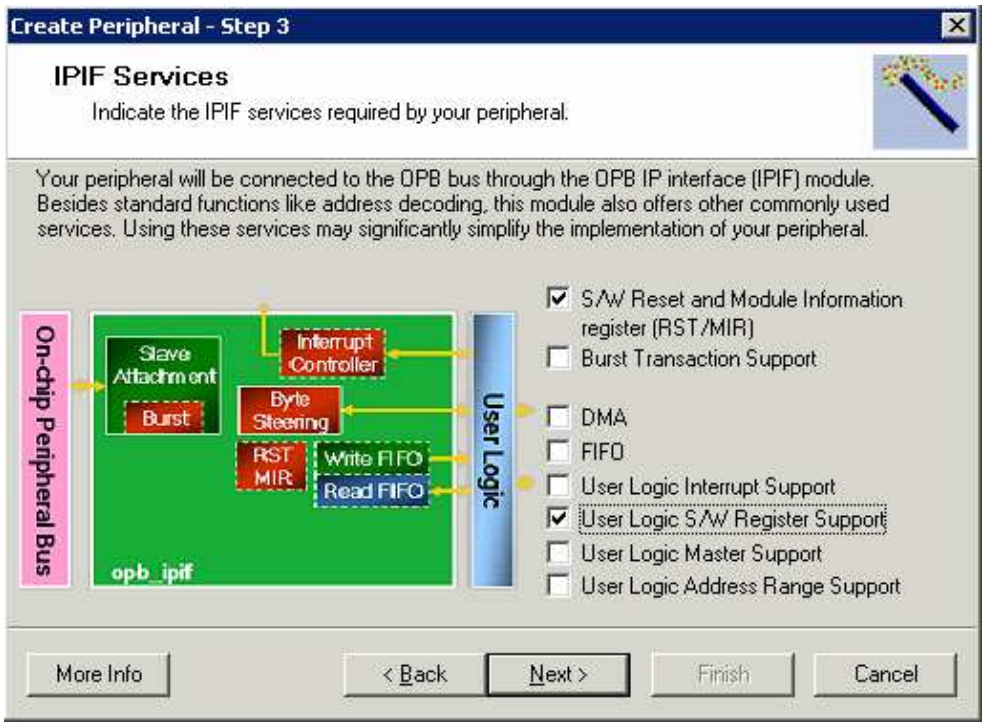


Figure 3-5. IPIF Services Dialog Box

❷ Click **Next**

❸ In the **User S/W Register** panel, click **Next** to accept the default values as will have only one register that will control the LEDs

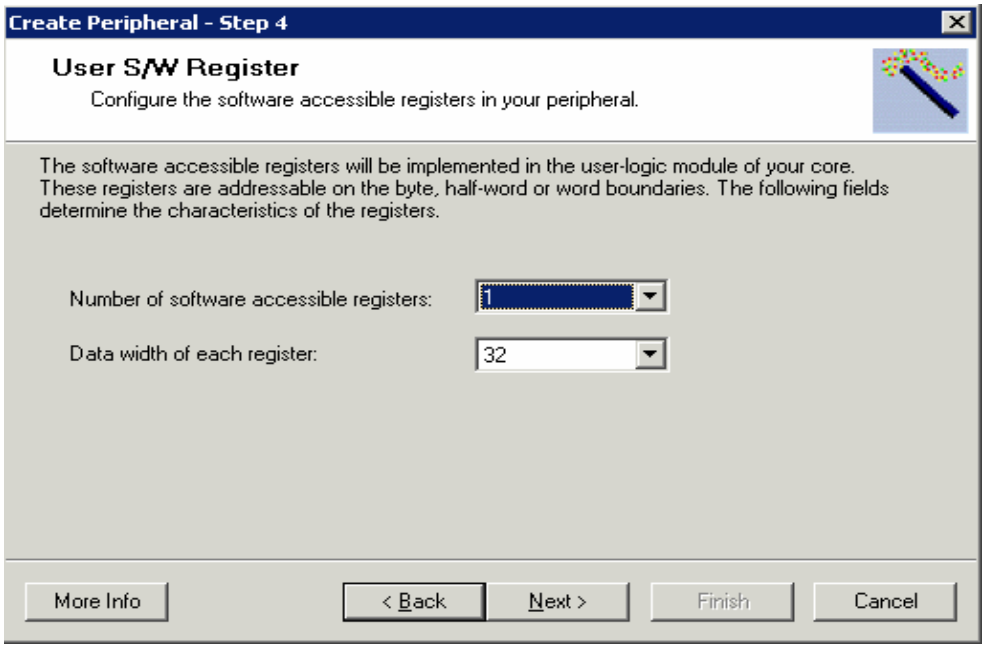


Figure 3-6. User S/W Register Dialog Box

❹ Scroll through the **IP Interconnect (IPIC)** panel, which displays the default IPIC signal which are available for the user logic based on the previous selection, click **Next**

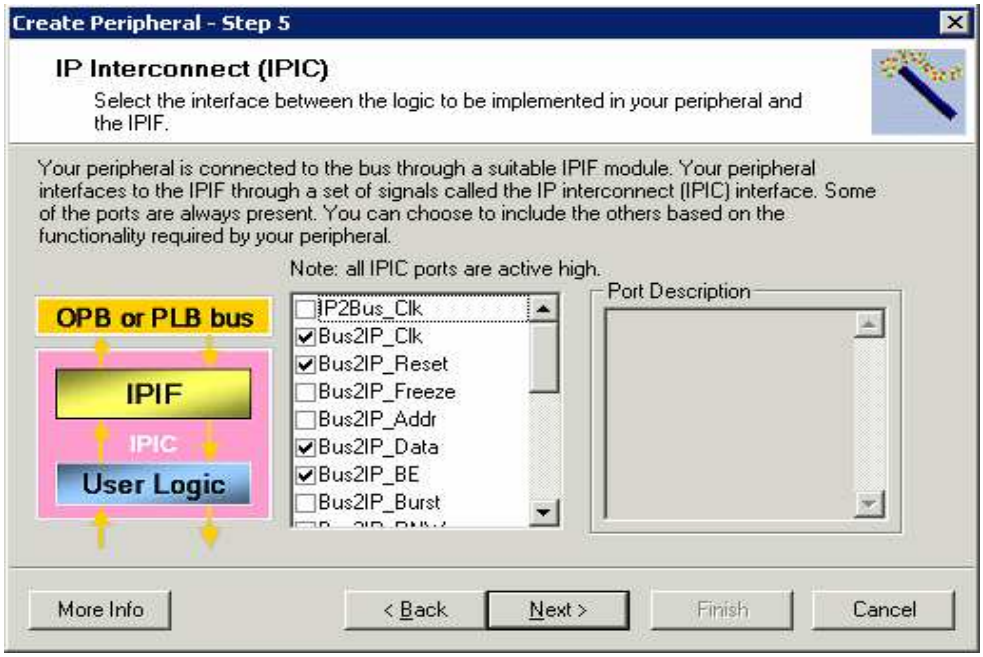


Figure 3-7. IP Interconnect (IPIC) Dialog Box

❺ In the **(OPTIONAL) Peripheral Simulation Support** panel, uncheck **Generate BFM simulation platform** inorder not to generate the BFM simulation associated files and directories, and click **Next**

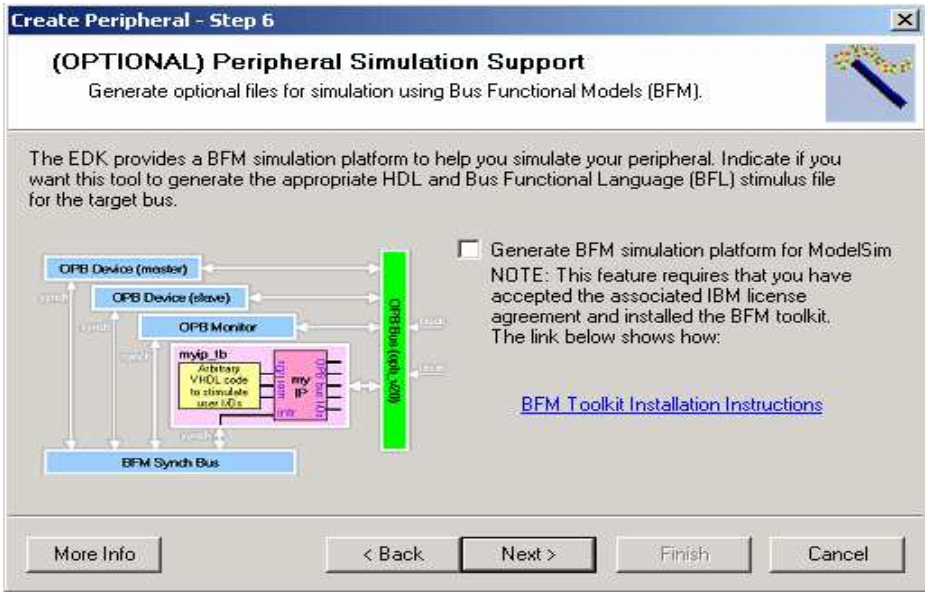


Figure 3-8. Peripheral Simulation Support Dialog Box

❻ In the **(OPTIONAL) Peripheral Implementation Options** panel, uncheck **Generate ISE and XST project files to help you not to implement the peripheral using XST flow** and check **Generate template driver files to help you to implement software interface**

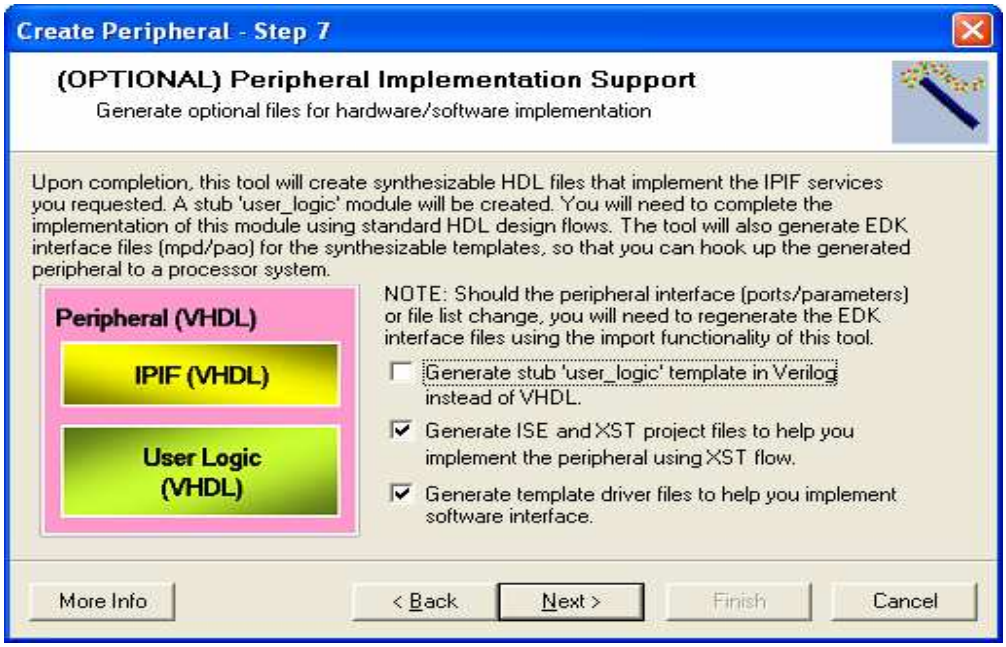


Figure 3-9. Peripheral Implementation Options Dialog Box

❼ Click **Next**, and you will see the summary information panel

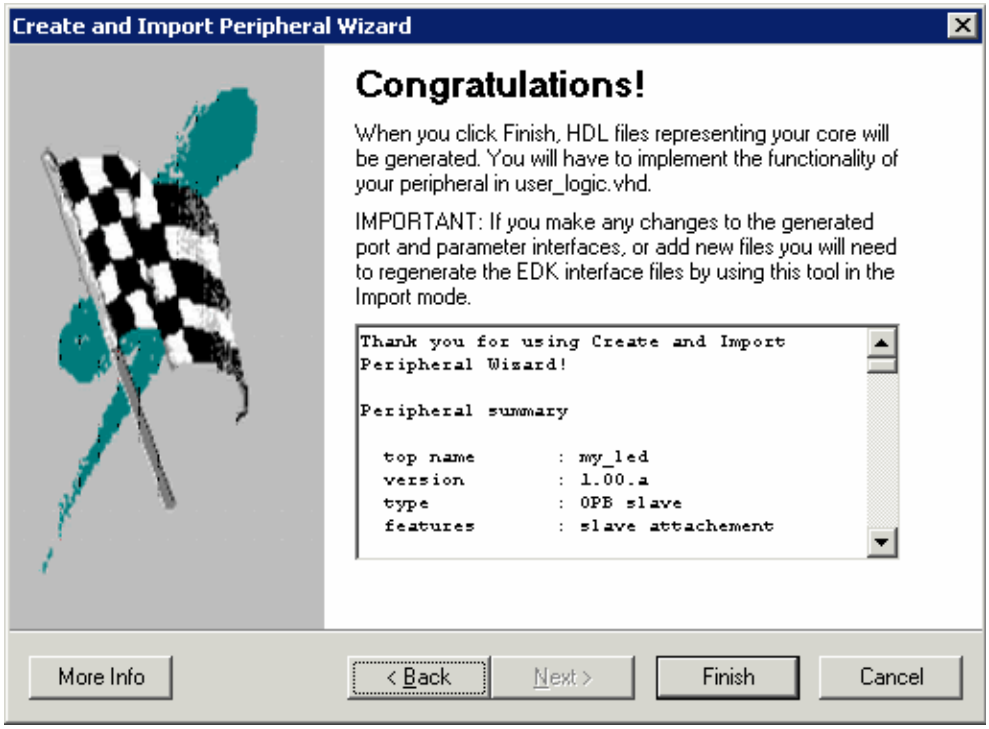


Figure 3-10. Congratulations Dialog Box

❽ Click **Finish** to close the wizard

Add **LED** port in **my_led_v2_1_0.mpd** file generated by the wizard in **C:\xup\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\data** directory

❶ Using **Windows Explorer**, browse to **C:\xup\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\data** directory

❷ Open **my_led_v2_1_0.mpd** file using an editor

❸ Add the following line before the **OPB_Clk** port under the **Ports** section

PORT LED = " ", DIR = O, VEC = [0:3]

This is necessary for the port to appear in Add/Edit Cores… (Dialog)

❹ Save the file and close

Open the my_led.vhd and user_logic.vhd files in the Text Editor window from **C:\XUP\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\hdl\vhdl** directory. Add necessary declarations and logic in my_led.vhd and user_logic.vhd files.

❶ Browse to **C:\xup\embedded\labs\lab3\pcores\my_led_v1_00_a\hdl\vhdl** directory.

❷ Right click on **my_led.vhd** file and open it with **text editor** program.

❸ Add user port **LED** under **USER ports added here** token

```
109  entity my_led is
110        generic
111        (
112              -- ADD USER GENERICS BELOW THIS LINE ---------------
113              --USER generics added here
114              -- ADD USER GENERICS ABOVE THIS LINE ---------------
115
116              -- DO NOT EDIT BELOW THIS LINE ----------------------
117              -- Bus protocol parameters, do not add to or delete
118              C_BASEADDR      : std_logic_vector   := X"00000000";
119              C_HIGHADDR      : std_logic_vector   := X"0000FFFF";
120              C_OPB_AWIDTH    : integer := 32;
121              C_OPB_DWIDTH    : integer := 32;
122              C_USER_ID_CODE  : integer := 3;
123              C_FAMILY   : string  := "virtex2p"
124              -- DO NOT EDIT ABOVE THIS LINE ----------------------
125        );
126        port
127        (
128              -- ADD USER PORTS BELOW THIS LINE ------------------
129              --USER ports added here
130              LED : out std_logic_vector (0 to 3);
131
132              -- ADD USER PORTS ABOVE THIS LINE ------------------
```

Figure 3-11. Add the User Port LED

❹ Search for next **--USER** and add port mapping statement

```
393       ---------------------------------------
394       -- instantiate the User Logic
395       ---------------------------------------
396       USER_LOGIC_I : entity my_led_v1_00_a.user_logic
397            generic map
398            (
399                 -- MAP USER GENERICS BELOW THIS LINE ---------------
400                 --USER generics mapped here
401                 -- MAP USER GENERICS ABOVE THIS LINE ---------------
402
403                 C_DWIDTH => USER_DWIDTH,
404                 C_NUM_CE => USER_NUM_CE
405            )
406            port map
407            (
408                 -- MAP USER PORTS BELOW THIS LINE ------------------
409                 --USER ports mapped here
410                 LED => LED,
411                 -- MAP USER PORTS ABOVE THIS LINE ------------------
```

Figure 3-12. Add Port Mapping Statement

❻ Open **user_logic.vhd** file from *vhdl* directory and add **LED** port definition in the USER Ports area

```
92  entity user_logic is
93        generic
94        (
95              -- ADD USER GENERICS BELOW THIS LINE ---------------
96              --USER generics added here
97              -- ADD USER GENERICS ABOVE THIS LINE ---------------
98
99              -- DO NOT EDIT BELOW THIS LINE ----------------------
100             -- Bus protocol parameters, do not add to or delete
101             C_DWIDTH   : integer := 32;
102             C_NUM_CE   : integer := 1
103             -- DO NOT EDIT ABOVE THIS LINE ----------------------
104       );
105       port
106       (
107             -- ADD USER PORTS BELOW THIS LINE ------------------
108             --USER ports added here
109             LED : out std_logic_vector (0 to 3);
110             -- ADD USER PORTS ABOVE THIS LINE ------------------
```

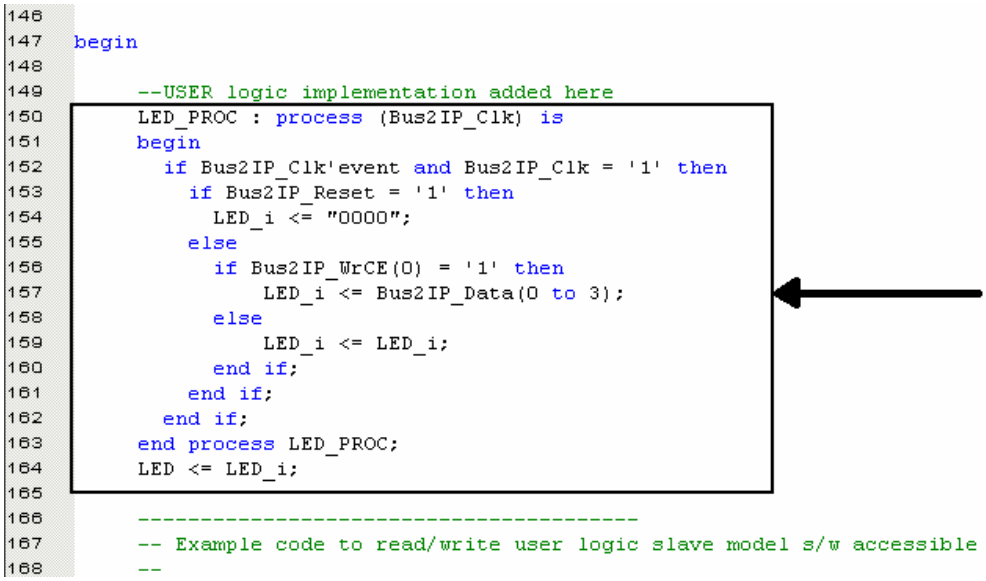
Figure 3-13. Add the LED Port Definition

❼ Search for next **--USER** and add the internal signal declaration for the user logic

```
123           IP2Bus_Error   : out     std_logic;
124           IP2Bus_ToutSup : out     std_logic
125           -- DO NOT EDIT ABOVE THIS LINE ---------------------
126     );
127 end entity user_logic;
128
129 ------------------------------------------------------------------------------
130 -- Architecture section
131 ------------------------------------------------------------------------------
132
133 architecture IMP of user_logic is
134
135     --USER signal declarations added here, as needed for user logic
136     signal LED_i : std_logic_vector (0 to 3);
137     ---------------------------------------
138     -- Signals for user logic slave model s/w accessible register example
139     ---------------------------------------
140     signal slv_reg0     : std_logic_vector(0 to C_DWIDTH-1);
141     signal slv_reg_write_select   : std_logic_vector(0 to 0);
```

Figure 3-14. Internal Signal Declaration for the User Logic

❽ Search for **–USER logic implementation** and add the following code



```
146
147  begin
148
149      --USER logic implementation added here
150      LED_PROC : process (Bus2IP_Clk) is
151      begin
152        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
153          if Bus2IP_Reset = '1' then
154            LED_i <= "0000";
155          else
156            if Bus2IP_WrCE(0) = '1' then
157                LED_i <= Bus2IP_Data(0 to 3);
158            else
159                LED_i <= LED_i;
160            end if;
161          end if;
162        end if;
163      end process LED_PROC;
164      LED <= LED_i;
165
166      ------------------------------------------
167      -- Example code to read/write user logic slave model s/w accessible
168      --
```

Figure 3-15. Add Code

❾ Save changes and close the **my_led-imp**

Importing and Adding Custom IP to the Project Step 2

Browse to **C:\xup\embedded\ppc\labs\lab3** and open **system.xmp**. Using **Tools → Create/Import Peripheral**, import the created **my_led** custom IP to the project

❶ Browse to **C:\xup\embedded\ppc\labs\lab3** and double-click on **system.xmp**. Open up the **Create and Import Peripheral Wizard** by selecting **Tools → Create/Import Peripheral…** from XPS menu and click **Next** to continue

❷ In the **Select Flow** panel, select the **Import existing peripheral** mode and click **Next**

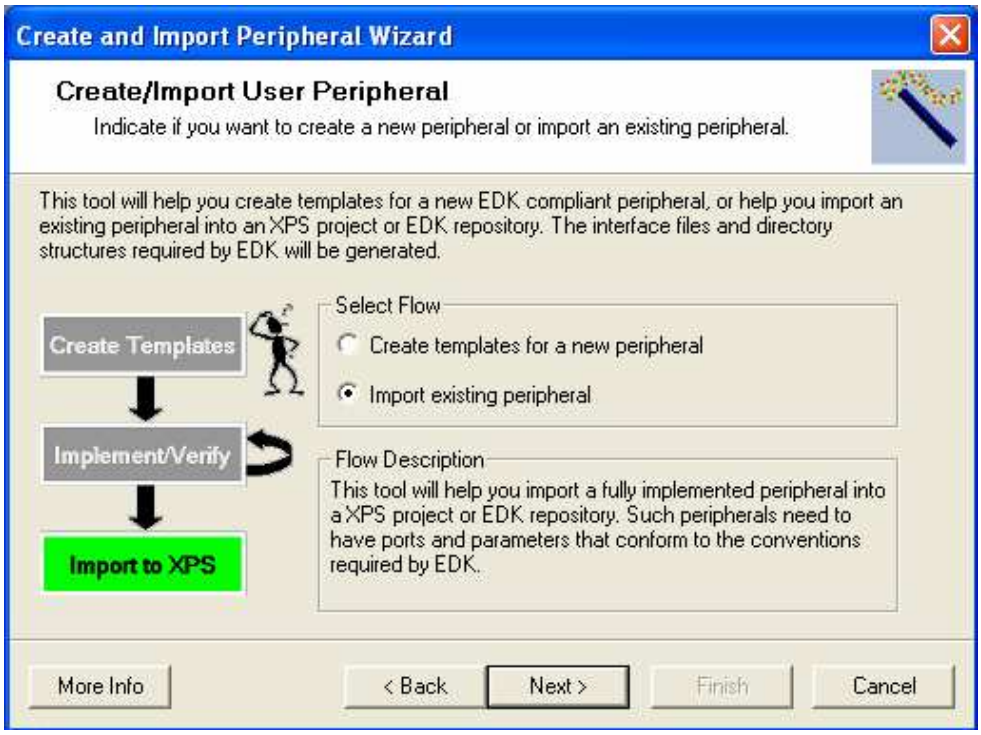


Figure 3-16. Create/Import User Peripheral Dialog Box

❸ In the **Repository or Project** panel, choose to import To an existing XPS project and select your current project from the drop down list **c:\xup\embedded\ppc\labs\lab3\system.xmp** and click **Next**

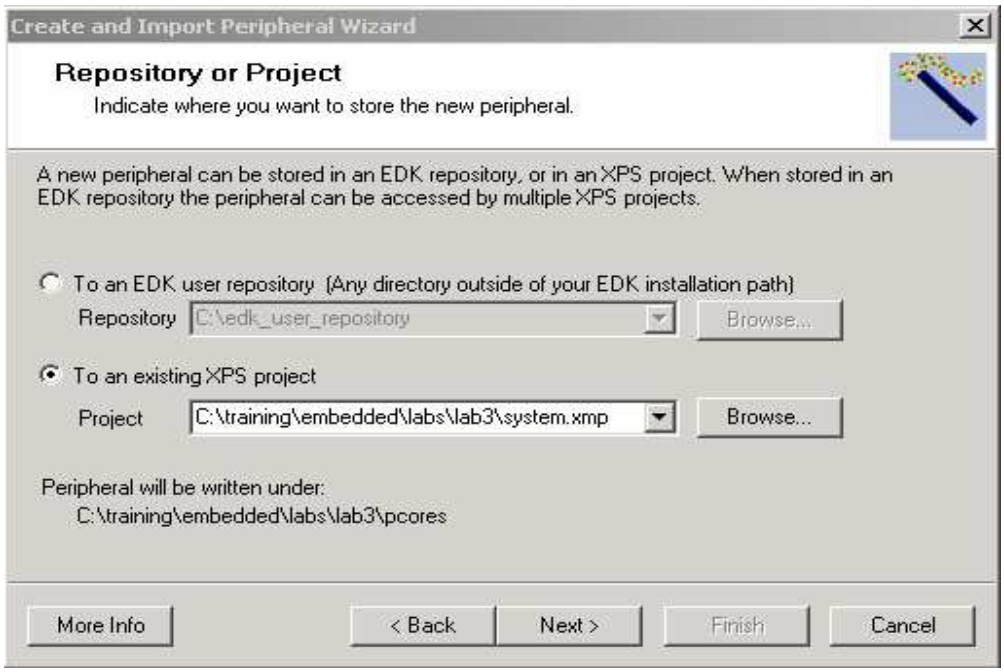


Figure 3-17. Repository or Project Dialog Box

❹ In the **Core Name and Version** panel, select **my_led** from the drop down list box, check **Use Version** and accept 1.00.a, and click **Next**

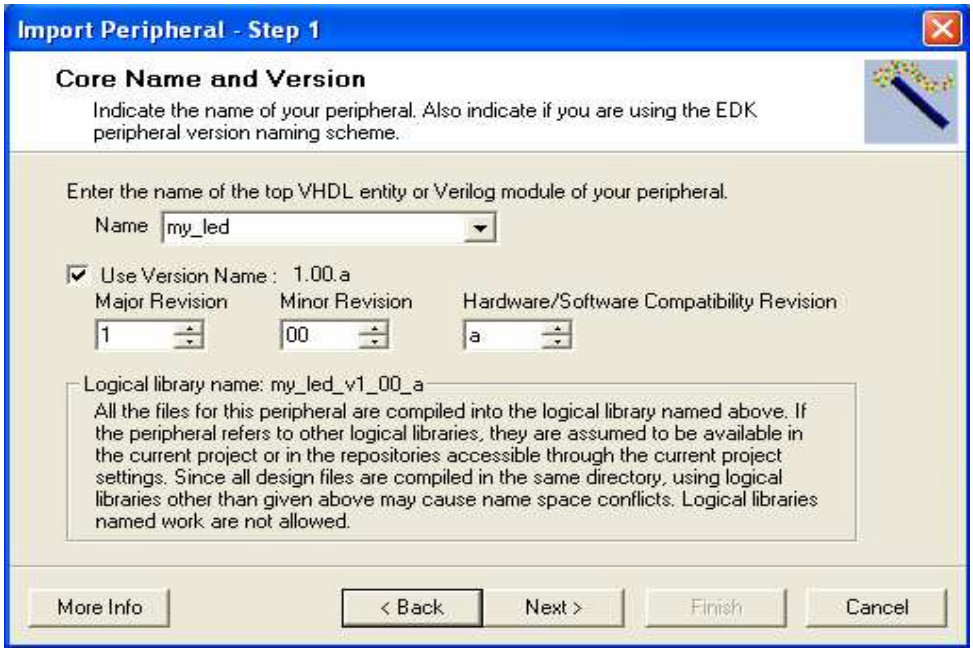


Figure 3-18. Core Name and Version Dialog Box

❺ In the **Source File Types** panel, check **HDL Source Files(*.vhd, *.v)**, and click **Next**

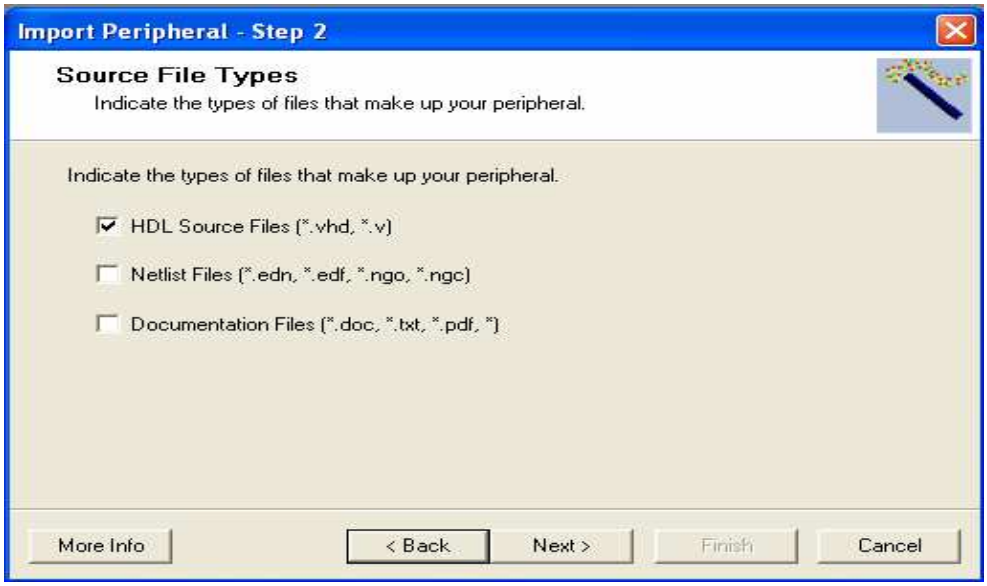


Figure 3-19. Source File Types Dialog Box

❻ In the **HDL Source Files** panel, select **Use existing Peripheral Analysis Order file (*.pao)** as the way to locate the HDL source files, specify the PAO file generated in the create mode by using the **Browse** button and browsing to **c:\xup\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\data\my_led_v2_1_0.pao**, and click **Next**

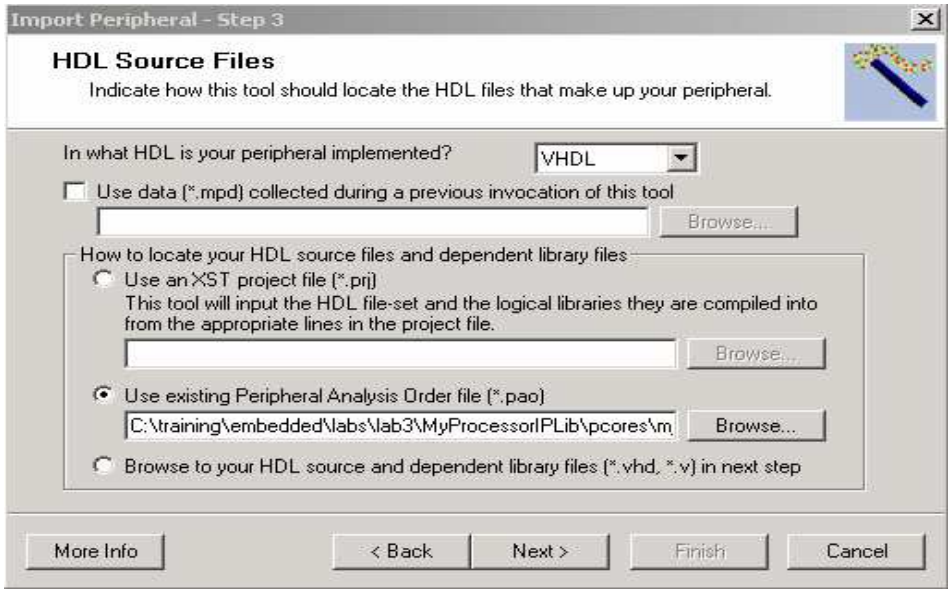


Figure 3-20. HDL Source Files Dialog Box

❼ The **HDL Analysis Information** panel shows you all the dependent library files and HDL source files to compile your peripheral, as well as corresponding logical libraries those files will be compiled into. For this custom peripheral, wizard based on the PAO file automatically intuits all files needed. Click **Next** to continue

❽ In the **Bus Interfaces** panel, check the **OPB Slave (SOPB)** bus interface

❾ Click **Next** until the **Identify Interrupt Signals** panel is reached, select No Interrupt, and click **Next** to continue

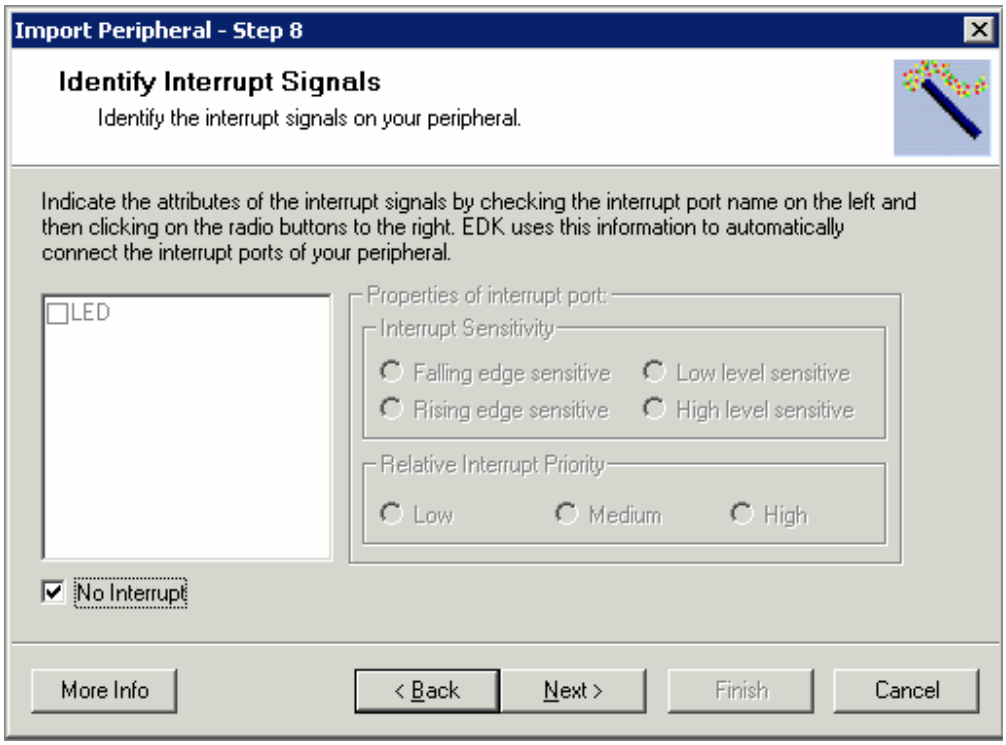


Figure 3-20. Identify Interrupt Signals Dialog Box

❿ Click **Next** until you reach the last page and click **Finish** to close wizard

You will see following message box. Click **Yes**

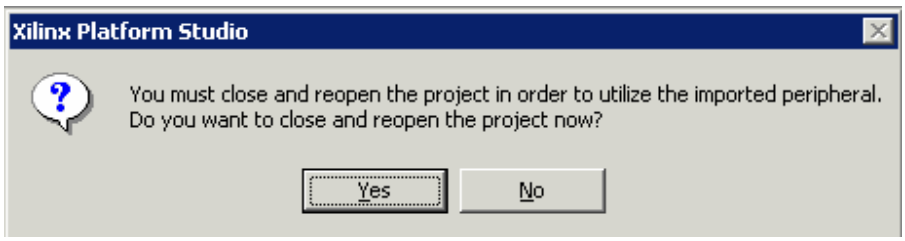


Figure 3-21. Confirmation Dialog Box

Using **Project → Add/Edit Core… (dialog)** from **XPS**, add **my_led** to the project, make bus connections, generate address for the **my_led** instance, add necessary ports to the instance, name them appropriately and bring out the data port. Add following to the UCF file:

```
Net fpga_0_LEDs_4Bit_GPIO_d_out<0> LOC=AC4;
Net fpga_0_LEDs_4Bit_GPIO_d_out<1> LOC=AC3;
Net fpga_0_LEDs_4Bit_GPIO_d_out<2> LOC=AA6;
Net fpga_0_LEDs_4Bit_GPIO_d_out<3> LOC=AA5;
```

1. Click **Project → Add/Edit Core… (dialog)** from XPS.

2. In the **Peripherals** tab, click the **OPB** radio button from the **Bus** group, click the **Custom IP** radio button from the **Component Filter** group, then highlight the custom peripheral **my_led** from the list box, and click **<< Add** button to add it to the hardware system

3. Switch to the **Bus Connections** tab, attach **my_led_0** to OPB bus as a salve device by clicking on the cell that corresponds to **my_led_0 sopb** row and **opb** column

4. Switch to the **Addresses** tab, select size as **512** from drop down box for **my_led_0** and click the **Generate Addresses** button to let XPS automatically assign addresses for all the peripherals for you, including the **my_led** custom peripheral

5. Switch to the **Ports** tab and perform the following

   - Select **my_led_0** from the Ports Filter combo box
   - Select the **LED** and **OPB_Clk** ports displayed under **my_led_0** from the right list box
   - Click **<< Add** to add them to the Internal Ports Connections table
   - Locate **my_led_0**'s **OPB_Clk** port in the table, modify its Net Name to **sys_clk_s** by selecting from its drop down list box
   - Locate **my_led_0**'s **LED** port in the table, modify its **Net Name** to **fpga_0_LEDs_4Bit_GPIO_d_out** by typing in the Net Name field
   - Select **my_led_0**'s **LED** port click on **Make External**
   - In the **External Ports Connections** table, select **my_led_0_LED** and type **[0:3]** in the corresponding **Range** field

6. Click **OK** to close the **Add/Edit Hardware Platform Specifications** dialog, open up the **system.mhs** file and verify following snippets

   ```
   PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = IN

   PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = OUT

   PORT sys_clk_pin = dcm_clk_s, DIR = IN
   ```

PORT sys_rst_pin = sys_rst_s, DIR = IN

PORT PUSH = PUSH, VEC = [0:4], DIR = I

PORT DIP = DIP, VEC = [0:3], DIR = I

PORT fpga_0_LEDs_4Bit_GPIO_d_out = fpga_0_LEDs_4Bit_GPIO_d_out, VEC = [0:3], DIR = O

❼ Add following pin assignments in the **UCF** file

```
Net fpga_0_LEDs_4Bit_GPIO_d_out<0> LOC=AC4;
Net fpga_0_LEDs_4Bit_GPIO_d_out<1> LOC=AC3;
Net fpga_0_LEDs_4Bit_GPIO_d_out<2> LOC=AA6;
Net fpga_0_LEDs_4Bit_GPIO_d_out<3> LOC=AA5;
```

❽ Save and close the UCF file

Develop Application and Verify the Design in Hardware Step 5

Run **LibGen**. Edit **TestApp.c** source file in the **TestApp** software project to display its settings on the LEDs. Compile the program successfully.

❶ In XPS, select **Options → Project Options** and then select the **Hierarchy and Flow** tab

❷ Under **Implementation Tool Flow**, select the **XPS (Xflow)** option and click **OK** to accept the settings

❸ Click **Tools → Generate Libraries and BSPs** to run the library generator

❹ In the **Application** tab, double-click on **TestApp_Memory.c** under the **Sources** of **TestApp_Memory** software project

❺ Edit the **TestApp_Memory.c** file to match the following

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

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

XGpio dip_push;
int i,psb_check, dip_check;

   print("-- Start of the program --\r\n");
   ....
   XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);
   XGpio_SetDataDirection(&dip_push,1,0xffffffff);

   XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);
   XGpio_SetDataDirection(&dip_push,2,0xffffffff);

»    while(1)
»    {
»    »    psb_check = XGpio_DiscreteRead(&dip_push,1);
»    »    xil_printf("Push Buttons Status : %x\n\r",psb_check);
»    »    dip_check = XGpio_DiscreteRead(&dip_push,2);
»    »    xil_printf("Dip Switch Status : %x\n\r",dip_check);
»    »    MY_LED_mWriteReg(XPAR_MY_LED_0_BASEADDR,0,dip_check);
»    »    sleep(1);
»    }

   print("-- End of the program --\r\n");
   return 0;
}
```

❹ Compile the program successfully

❺ Click **Tools → Update Bitstream** to generate the bit file

This may take over 10 minutes.

❻ Download the bit file on the board

❼ Change dip switch settings and see the corresponding LED turning ON and OFF, verifying the functionality

Conclusion

This lab led you through the creating a custom IP. Once created, it guided you to import and add the custom IP into a system. Further, you developed an application using the custom IP driver and verified its functionality in hardware. Thus you were able to see the ease of creating/importing a custom IP.

Completed MHS File

```
# ##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 7.1.1 Build EDK_H.11.3
# Wed Jun 08 23:27:28 2005
# Target Board:  Xilinx XUP Virtex-II Pro Development System Rev A
# Family:        virtex2p
# Device:        xc2vp30
# Package:       ff896
# Speed Grade:   -6
# Processor: PPC 405
# Processor clock frequency: 300.000000 MHz
# Bus clock frequency: 100.000000 MHz
# Debug interface: FPGA JTAG
# On Chip Memory :  64 KB
# ##############################################################################


 PARAMETER VERSION = 2.1.0


 PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = INPUT
 PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = OUTPUT
 PORT sys_clk_pin = dcm_clk_s, DIR = INPUT
 PORT sys_rst_pin = sys_rst_s, DIR = INPUT
 PORT PUSH = PUSH, VEC = [0:4], DIR = I
 PORT DIP = DIP, VEC = [0:3], DIR = I
 PORT fpga_0_LEDs_4Bit_GPIO_d_out = fpga_0_LEDs_4Bit_GPIO_d_out, VEC = [0:3], DIR = O


BEGIN ppc405
 PARAMETER INSTANCE = ppc405_0
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_0
 BUS_INTERFACE IPLB = plb
 BUS_INTERFACE DPLB = plb
 PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ
 PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ
 PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ
 PORT RSTC405RESETCHIP = RSTC405RESETCHIP
 PORT RSTC405RESETCORE = RSTC405RESETCORE
 PORT RSTC405RESETSYS = RSTC405RESETSYS
 PORT CPMC405CLOCK = proc_clk_s
 PORT PLBCLK = sys_clk_s
END

BEGIN ppc405
 PARAMETER INSTANCE = ppc405_1
 PARAMETER HW_VER = 2.00.c
 BUS_INTERFACE JTAGPPC = jtagppc_0_1
END
```

```
BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_0
 PARAMETER HW_VER = 2.00.a
 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = reset_block
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
 PORT Core_Reset_Req = C405RSTCORERESETREQ
 PORT System_Reset_Req = C405RSTSYSRESETREQ
 PORT Rstc405resetchip = RSTC405RESETCHIP
 PORT Rstc405resetcore = RSTC405RESETCORE
 PORT Rstc405resetsys = RSTC405RESETSYS
 PORT Dcm_locked = dcm_0_lock
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Slowest_sync_clk = sys_clk_s
 PORT Ext_Reset_In = sys_rst_s
END

BEGIN plb_v34
 PARAMETER INSTANCE = plb
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT PLB_Clk = sys_clk_s
END

BEGIN opb_v20
 PARAMETER INSTANCE = opb
 PARAMETER HW_VER = 1.10.c
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT SYS_Rst = sys_bus_reset
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb2opb_bridge
 PARAMETER INSTANCE = plb2opb
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_RNG0_BASEADDR = 0x40000000
 PARAMETER C_RNG0_HIGHADDR = 0x7fffffff
 PARAMETER C_NUM_ADDR_RNG = 1
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE MOPB = opb
 PORT OPB_Clk = sys_clk_s
 PORT PLB_Clk = sys_clk_s
END
```

```
BEGIN opb_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BAUDRATE = 115200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_CLK_FREQ = 100000000
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE SOPB = opb
 PORT RX = fpga_0_RS232_Uart_1_RX
 PORT TX = fpga_0_RS232_Uart_1_TX
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = plb_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0xffff0000
 PARAMETER c_highaddr = 0xffffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = plb_bram1
 PORT PLB_Clk = sys_clk_s
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = plb_bram1
END

BEGIN dcm_module
 PARAMETER INSTANCE = dcm_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_CLK0_BUF = TRUE
 PARAMETER C_CLKFX_BUF = TRUE
 PARAMETER C_CLKFX_DIVIDE = 1
 PARAMETER C_CLKFX_MULTIPLY = 3
 PARAMETER C_CLKIN_PERIOD = 10.000000
 PARAMETER C_CLK_FEEDBACK = 1X
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT LOCKED = dcm_0_lock
 PORT CLKIN = dcm_clk_s
 PORT RST = net_gnd
 PORT CLKFX = proc_clk_s
 PORT CLK0 = sys_clk_s
 PORT CLKFB = sys_clk_s
END
```

```
BEGIN opb_gpio
 PARAMETER INSTANCE = dip_push
 PARAMETER HW_VER = 3.01.b
 PARAMETER C_GPIO_WIDTH = 32
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_IS_BIDIR = 0
 PARAMETER C_IS_DUAL = 1
 PARAMETER C_ALL_INPUTS_2 = 1
 PARAMETER C_IS_BIDIR_2 = 0
 PARAMETER C_BASEADDR = 0x40000000
 PARAMETER C_HIGHADDR = 0x4000ffff
 BUS_INTERFACE SOPB = opb
 PORT GPIO2_in = DIP
 PORT GPIO_in = PUSH
 PORT OPB_Clk = sys_clk_s
END

BEGIN plb_bram_if_cntlr
 PARAMETER INSTANCE = plb_bram_if_cntlr_2
 PARAMETER HW_VER = 1.00.b
 PARAMETER c_plb_clk_period_ps = 10000
 PARAMETER c_baseaddr = 0x00000000
 PARAMETER c_highaddr = 0x00003fff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = plb_bram2
 PORT plb_clk = sys_clk_s
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntrl_2_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = plb_bram2
END

BEGIN my_led
 PARAMETER INSTANCE = my_led_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x7d800000
 PARAMETER C_HIGHADDR = 0x7d80ffff
 BUS_INTERFACE SOPB = opb
 PORT LED = fpga_0_LEDs_4Bit_GPIO_d_out
 PORT OPB_Clk = sys_clk_s
END
```

Completed MSS File

```
 PARAMETER VERSION = 2.2.0


BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_0
 PARAMETER STDIN = RS232_Uart_1
 PARAMETER STDOUT = RS232_Uart_1
END

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER PROC_INSTANCE = ppc405_1
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_0
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
 PARAMETER CORE_CLOCK_FREQ_HZ = 300000000
END

BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu_ppc405
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ppc405_1
 PARAMETER COMPILER = powerpc-eabi-gcc
 PARAMETER ARCHIVER = powerpc-eabi-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = plbarb
 PARAMETER DRIVER_VER = 1.01.a
 PARAMETER HW_INSTANCE = plb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
 PARAMETER HW_INSTANCE = opb
END

BEGIN DRIVER
```

```
 PARAMETER DRIVER_NAME = plb2opb
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb2opb
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER HW_INSTANCE = RS232_Uart_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb_bram_if_cntlr_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = dcm_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = dip_push
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plb_bram_if_cntlr_2
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = my_led
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = my_led_0
END
```

Completed C File

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

//====================================================
int main (void)
{

XGpio dip_push;
int i,psb_check, dip_check;

  print("-- Start of the program --\r\n");

   XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);
   XGpio_SetDataDirection(&dip_push,1,0xffffffff);

   XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);
   XGpio_SetDataDirection(&dip_push,2,0xffffffff);

        while(1)
        {
                psb_check = XGpio_DiscreteRead(&dip_push,1);
                xil_printf("Push Buttons Status : %x\n\r",psb_check);
                dip_check = XGpio_DiscreteRead(&dip_push,2);
                xil_printf("Dip Switch Status : %x\n\r",dip_check);
                MY_LED_mWriteReg(XPAR_MY_LED_0_BASEADDR,0,dip_check);
                sleep(1);
        }

  print("-- End of the program --\r\n");
  return 0;
}
```