Adding Custom IP to an Embedded System Using AXI

Introduction

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

Objectives

After completing this lab, you will be able to:

- Create a custom peripheral
- Add the custom peripheral to your design
- Add pin location constraints
- Generate the hardware bitstream

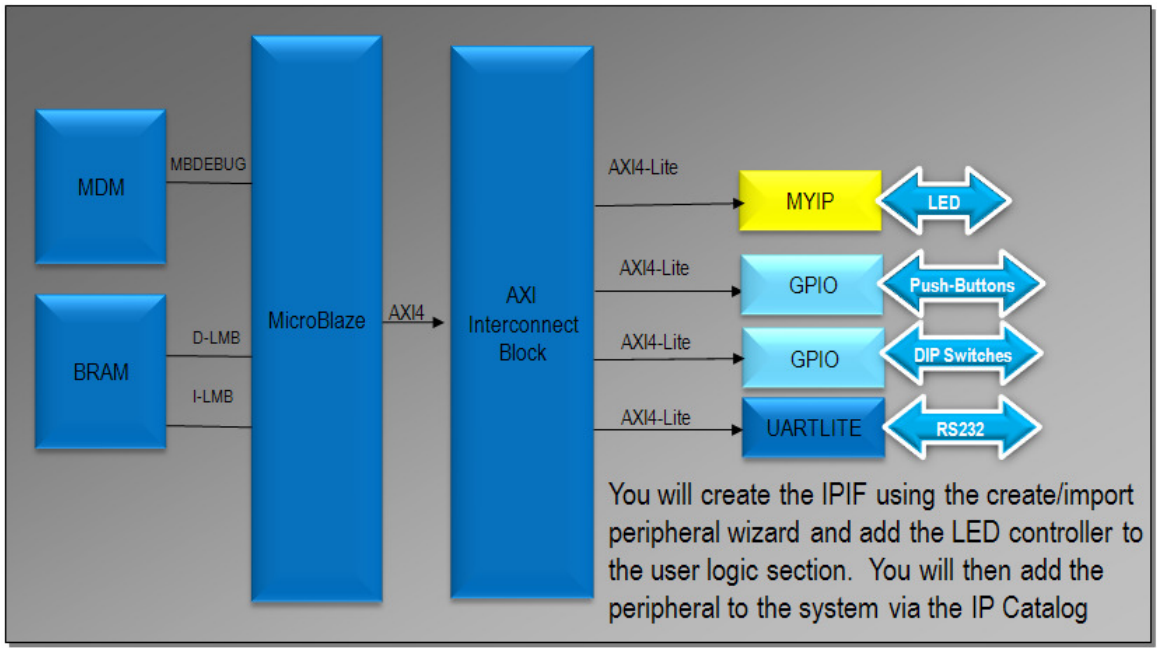
Procedure

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 4 primary steps: You will open the project, generate a peripheral template, create a peripheral, and, finally, add and connect the peripheral.

Design Description

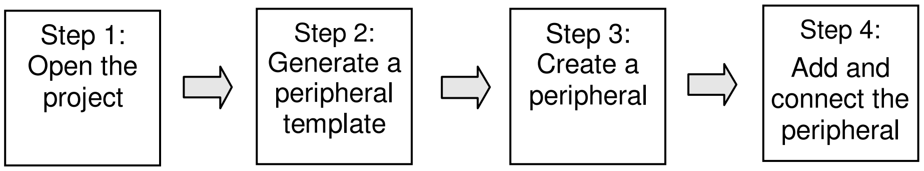
You will extend the Lab 2 hardware design by creating and adding a AXI peripheral (refer to MYIP in **Figure 1**) to the system, and connecting it to the LEDs on the Nexys3 Board. You will use the Create and Import Peripheral Wizard of Xilinx Platform Studio (XPS) to generate the peripheral templates. You will complete the peripheral by adding LEDs interface logic in the templates. Next, you will connect the peripheral to the system and add pin location constraints to connect the LEDs controller peripheral to the on-board LEDs.



You will create the IPIF using the create/import peripheral wizard and add the LED controller to the user logic section. You will then add the peripheral to the system via the IP Catalog

Figure 1. Design updated from previous lab

General Flow for this Lab


Step 1: Open the project
Step 2: Generate a peripheral template
Step 3: Create a peripheral
Step 4: Add and connect the peripheral

Opening the Project Step 1

1-1. 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, otherwise copy the *lab2* folder content from the *labsolution* folder into the *lab3* folder. Open the project in XPS.

1-1-1. If you wish to continue using the design that you created in Lab 2, create a *lab3* folder in the c*:\xup\embedded\labs* directory and copy the contents from *lab2* to *lab3*, otherwise copy the content of *lab2* folder from the *labsolution* folder.

1-1-2. Open XPS by clicking **Start** > **All Programs** > **Xilinx Design Tools** > **ISE Design Suite 14.2** > **EDK** > **Xilinx Platform Studio.**

1-1-3. Select **Open project,** and browse to *C:\xup\embedded\labs\lab3.*

1-1-4. Click **system.xmp,** and click **Open** to open the project.

Generate a Peripheral Template Step 2

2-1. You will use the Create/Import Peripheral Wizard to create a AXI4Lite interface peripheral template.

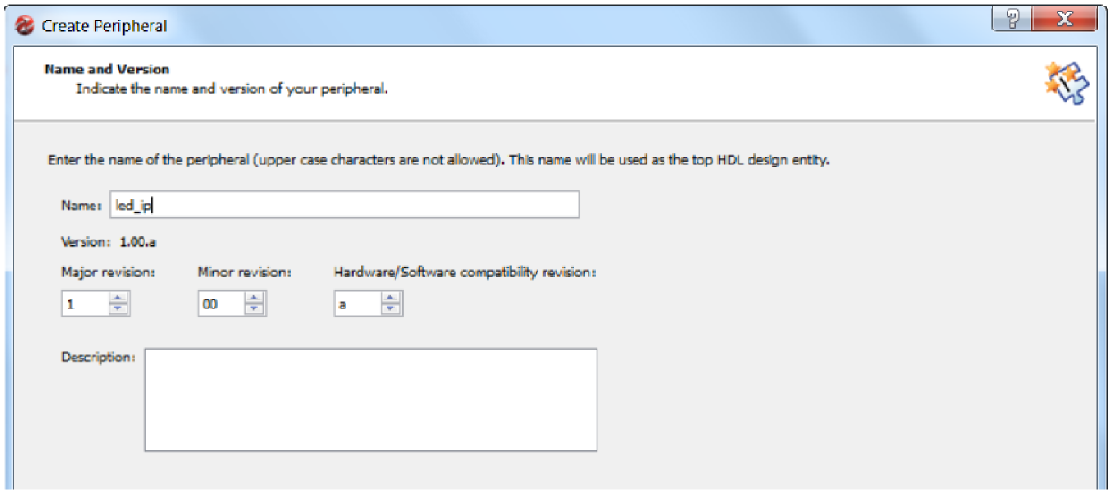
2-1-1. In XPS, select **Hardware** > **Create or Import Peripheral…** to start the wizard.

2-1-2. Click **Next** to continue to the Create and Import Peripheral Wizard flow selection.

2-1-3. In the *Select Flow* panel, select **Create templates for a new peripheral** and click **Next.**

2-1-4. Click **next** with the default option **To an XPS project** selected.

2-1-5. Click **Next** and enter ***led_ip*** in the Name field, leave the default version number of 1.00.a, and click **Next**.


Create Peripheral

Name and Version
Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name: led_ip

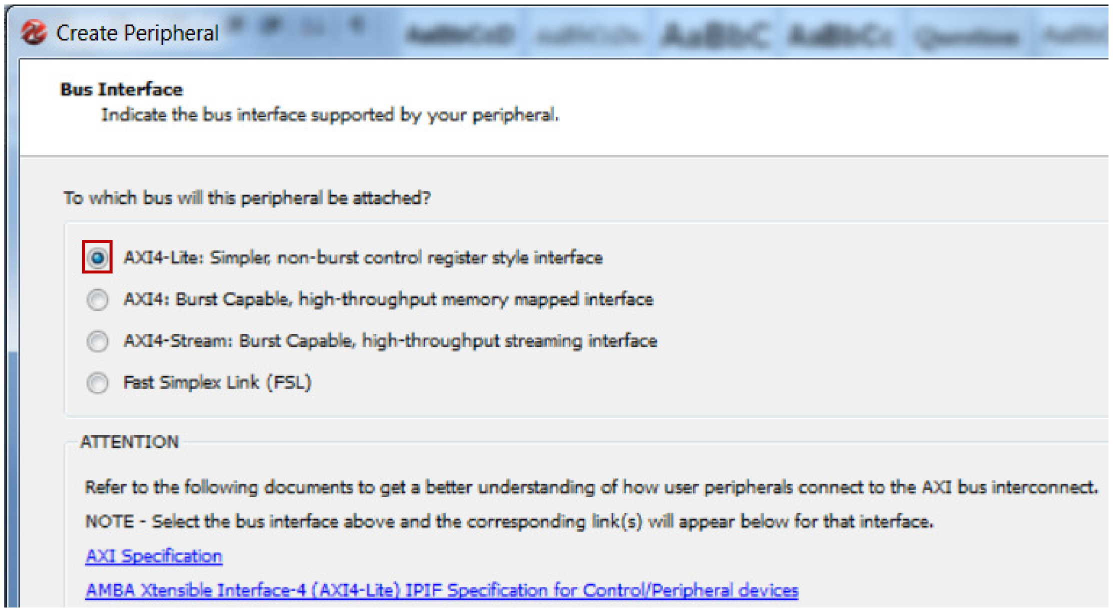
Version: 1.00.a

Major revision: 1
Minor revision: 00
Hardware/Software compatibility revision: a

Description:

Figure 2. Provide Core Name and Version Number

2-1-6. Select **AXI4-Lite** interface and click **Next.**


Create Peripheral

Bus Interface
Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

- (●) AXI4-Lite: Simpler, non-burst control register style interface
- ( ) AXI4: Burst Capable, high-throughput memory mapped interface
- ( ) AXI4-Stream: Burst Capable, high-throughput streaming interface
- ( ) Fast Simplex Link (FSL)

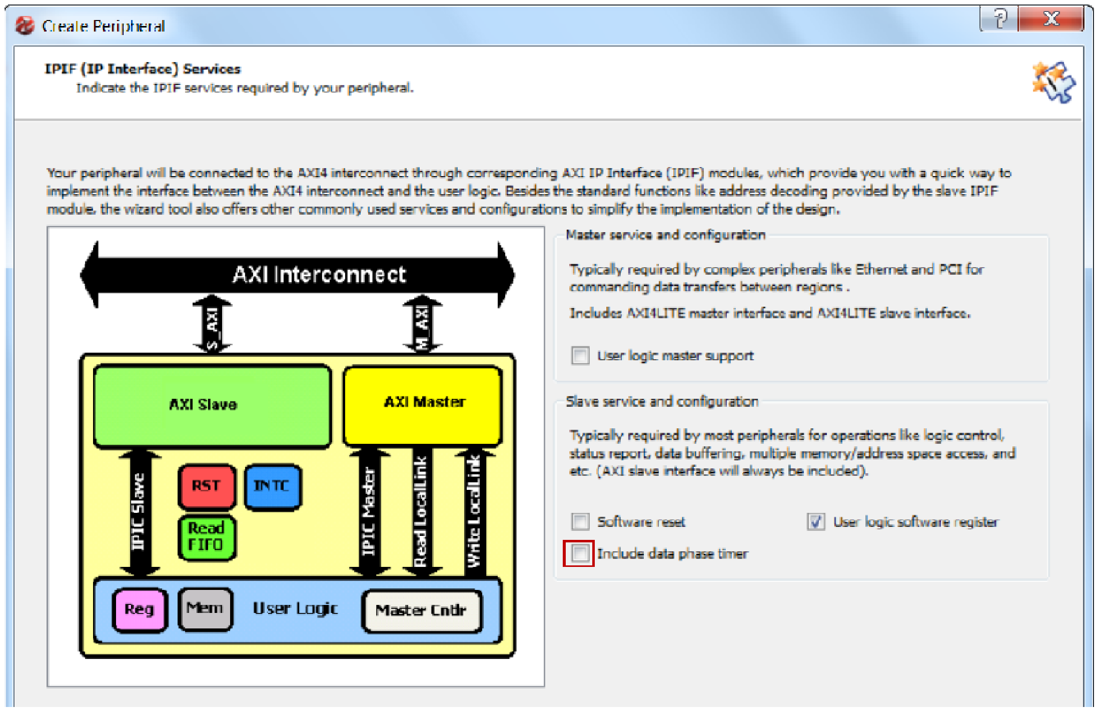
ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the AXI bus interconnect.
NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.
AXI Specification
AMBA Xtensible Interface-4 (AXI4-Lite) IPIF Specification for Control/Peripheral devices

Figure 3. Select the AXI4-Lite interface

2-2. Continuing with the wizard, select 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\labs\lab3 directory and ensure the structure.

2-2-1. In the IPIF Services panel, deselect **Include data phase timer**, leaving **User logic software register** selected, and click **Next**.


Create Peripheral

IPIF (IP Interface) Services
Indicate the IPIF services required by your peripheral.

Your peripheral will be connected to the AXI4 interconnect through corresponding AXI IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the AXI4 interconnect and the user logic. Besides the standard functions like address decoding provided by the slave IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation of the design.

Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions.
Includes AXI4LITE master interface and AXI4LITE slave interface.

[ ] User logic master support

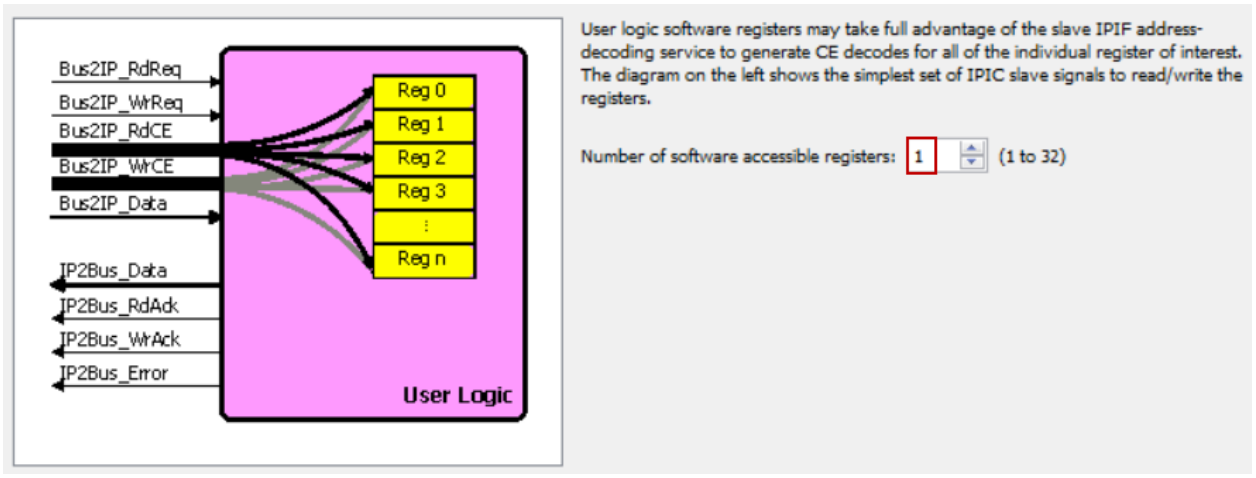
Slave service and configuration

Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (AXI slave interface will always be included).

[ ] Software reset
[✓] User logic software register
[ ] Include data phase timer

Figure 4. IPIF Services Dialogue Box

2-2-2. Click **Next,** accepting the default number of registers to be **1**.

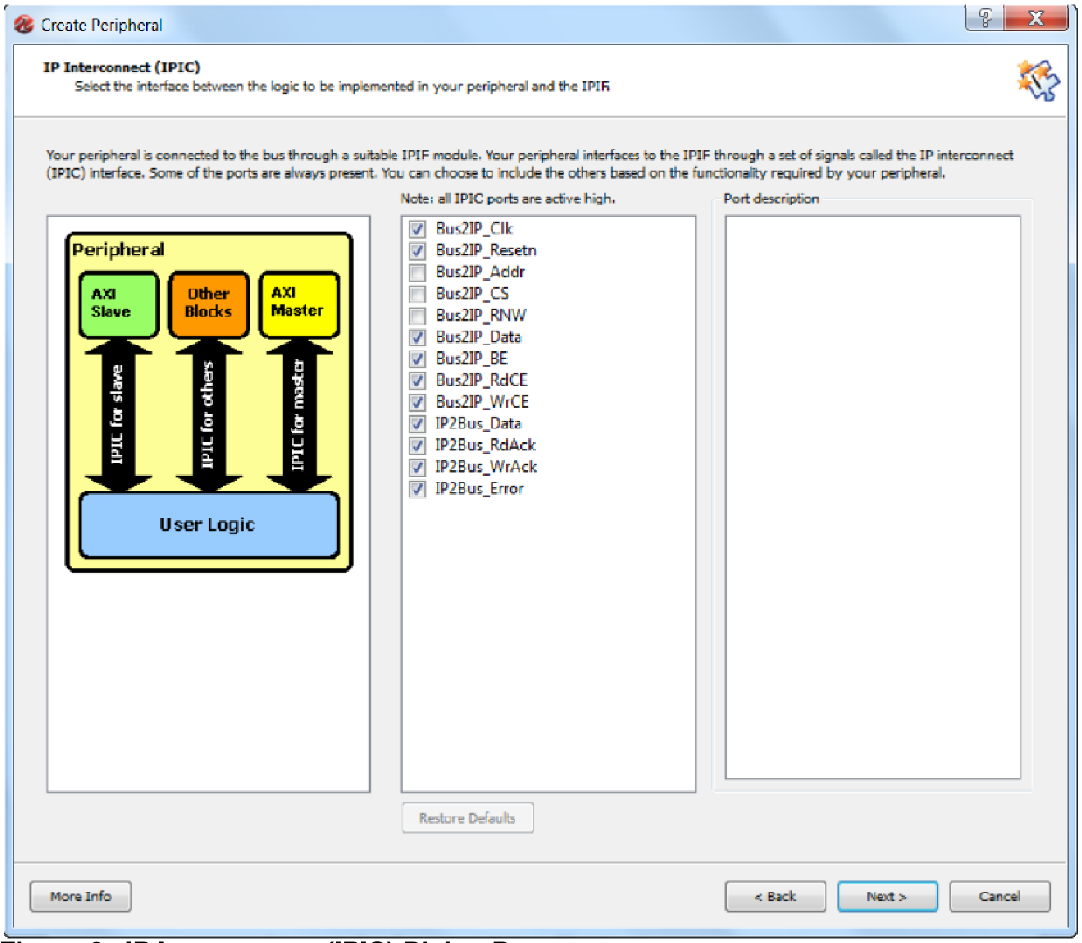


User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIC slave signals to read/write the registers.

Number of software accessible registers: 1 (1 to 32)

Figure 5. User SW Registers

2-2-3. Scroll through the **IP Interconnect (IPIC)** panel, which displays the default IPIC signals that are available for the user logic based on the previous selection. Notice that Bus2IP_Addr, Bus2IP_CS, Bus2IP_RNW are not checked since the custom peripheral does not have any memory support. Click **Next.**


Create Peripheral

IP Interconnect (IPIC)
Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral is connected to the bus through a suitable IPIF module. Your peripheral interfaces to the IPIF through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present. You can choose to include the others based on the functionality required by your peripheral.

Note: all IPIC ports are active high.

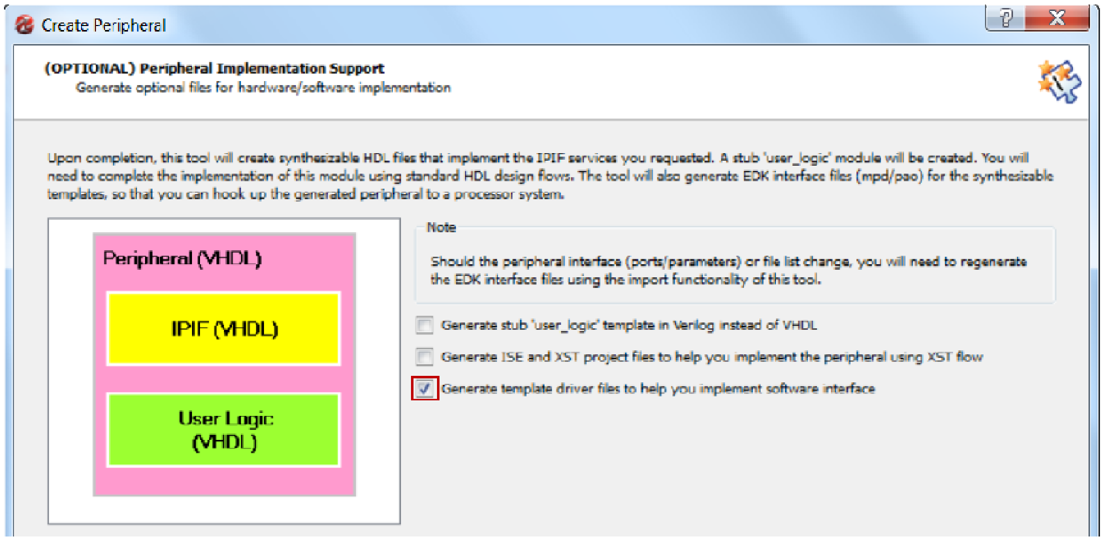
- [x] Bus2IP_Clk
- [x] Bus2IP_Resetn
- [ ] Bus2IP_Addr
- [ ] Bus2IP_CS
- [ ] Bus2IP_RNW
- [x] Bus2IP_Data
- [x] Bus2IP_BE
- [x] Bus2IP_RdCE
- [x] Bus2IP_WrCE
- [x] IP2Bus_Data
- [x] IP2Bus_RdAck
- [x] IP2Bus_WrAck
- [x] IP2Bus_Error

Port description

Figure 6. IP Interconnect (IPIC) Dialog Box

2-2-4. In the **(OPTIONAL) Peripheral Simulation Support** panel, leave **Generate BFM simulation platform** unchecked, and click **Next.**

2-2-5. In the **(OPTIONAL) Peripheral Implementation Options** panel, click **Generate template driver files to help you to implement software interface,** leaving others unchecked.


Create Peripheral

(OPTIONAL) Peripheral Implementation Support
Generate optional files for hardware/software implementation

Upon completion, this tool will create synthesizable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the synthesizable templates, so that you can hook up the generated peripheral to a processor system.

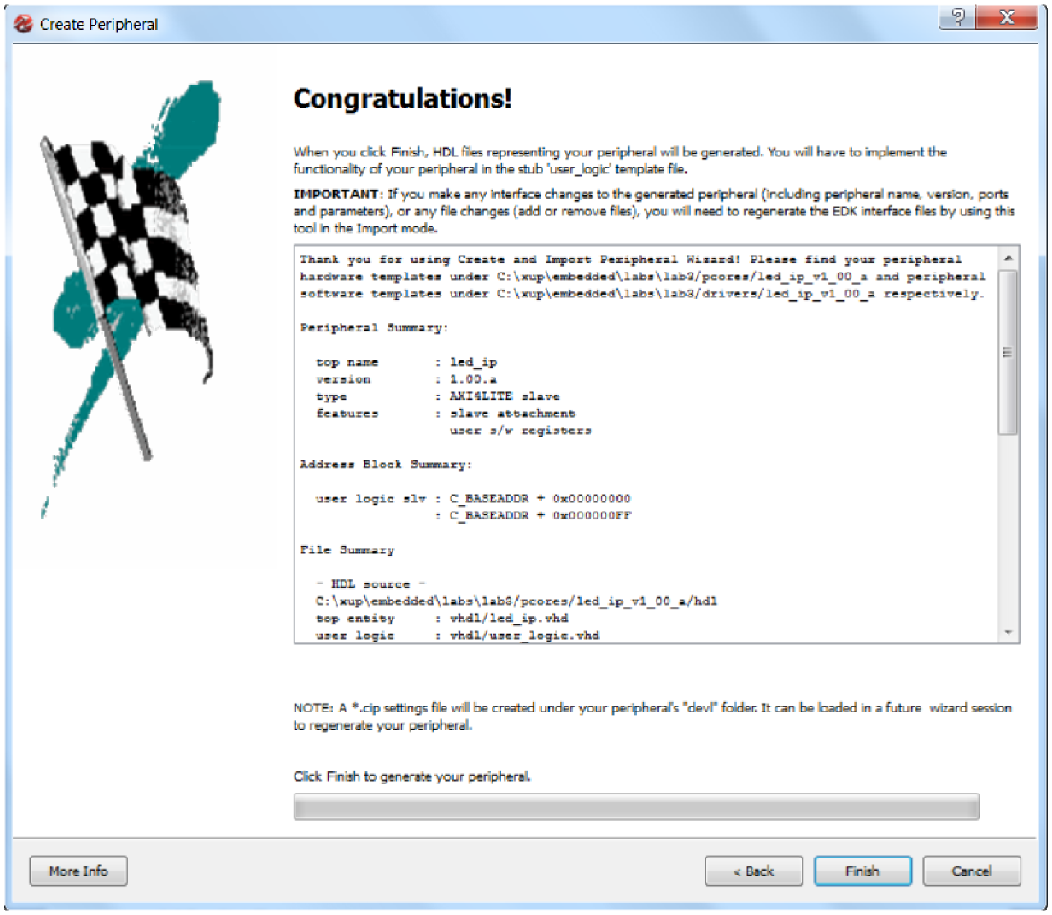
Note

Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

- [ ] Generate stub 'user_logic' template in Verilog instead of VHDL
- [ ] Generate ISE and XST project files to help you implement the peripheral using XST flow
- [x] Generate template driver files to help you implement software interface

Figure 7. Peripheral Implementation Options dialog box

2-2-6. Click **Next**, and you will see the summary information panel.



Congratulations!

When you click Finish, HDL files representing your peripheral will be generated. You will have to implement the functionality of your peripheral in the stub 'user_logic' template file.

IMPORTANT: If you make any interface changes to the generated peripheral (including peripheral name, version, ports and parameters), or any file changes (add or remove files), you will need to regenerate the EDK interface files by using this tool in the Import mode.

```
Thank you for using Create and Import Peripheral Wizard! Please find your peripheral
hardware templates under C:\xup\embedded\labs\lab3/pcores/led_ip_v1_00_a and peripheral
software templates under C:\xup\embedded\labs\lab3/drivers/led_ip_v1_00_a respectively.

Peripheral Summary:

  top name       : led_ip
  version        : 1.00.a
  type           : AXI4LITE slave
  features       : slave attachment
                   user s/w registers

Address Block Summary:

  user logic slv : C_BASEADDR + 0x00000000
                 : C_BASEADDR + 0x000000FF

File Summary

  - HDL source -
  C:\xup\embedded\labs\lab3/pcores/led_ip_v1_00_a/hdl
  top entity     : vhdl/led_ip.vhd
  user logic     : vhdl/user_logic.vhd
```

NOTE: A *.cip settings file will be created under your peripheral's "devl" folder. It can be loaded in a future wizard session to regenerate your peripheral.

Click Finish to generate your peripheral.

Figure 8. Congratulations dialog box

2-2-7. Click **Finish** to close the wizard.

2-2-8. Click on **IP Catalog** tab in XPS and observe that **LED_IP** is added under USER sub-folder under the **Project Local PCores** repository.

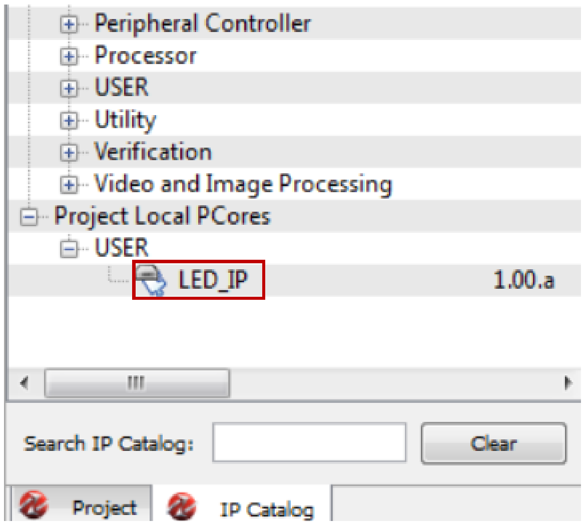


Figure 9. IP Catalog updated entry

The peripheral which you just added becomes part of the available cores list. Use Windows Explorer to browse to your project directory and ensure that the following structure has been created by the Create and Import Peripheral Wizard.

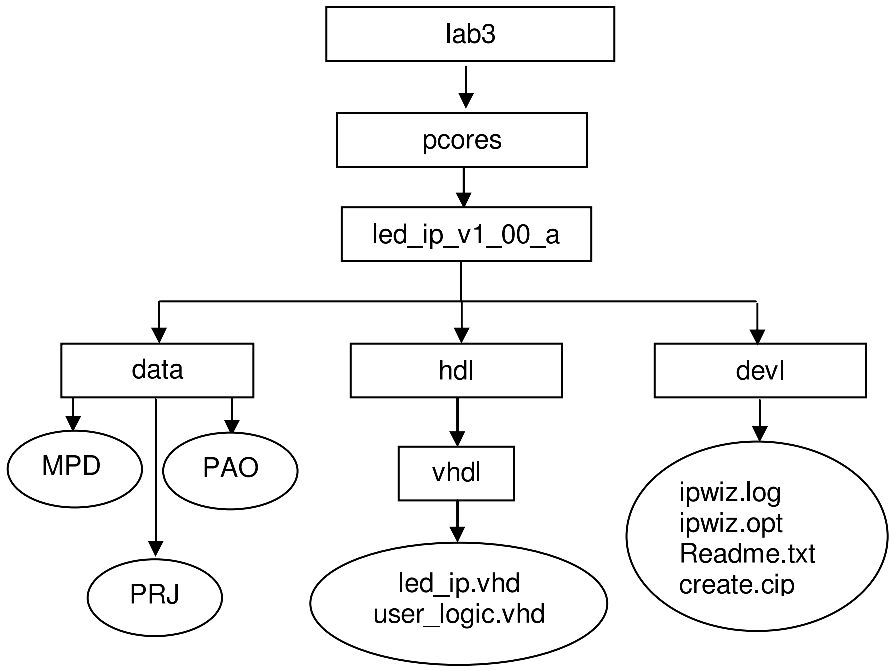


Figure 10. Structure created by the Create and Import Peripheral Wizard

Create the Peripheral — Step 3

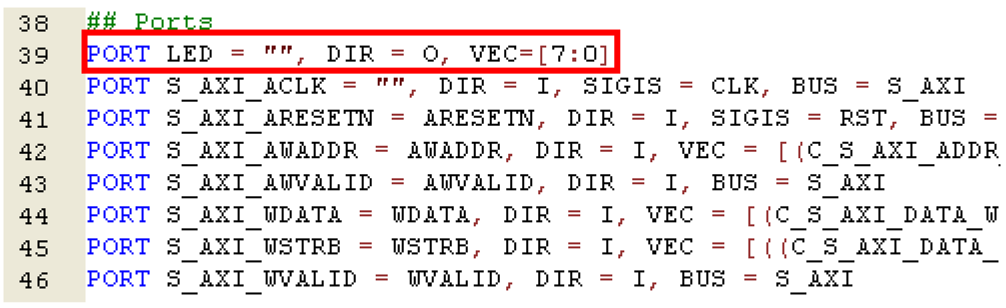
3-1. Update the MPD file to include the led data output of the LED controller peripheral so the port can be connected in XPS.

Add a port called "led" to the MPD file.

3-1-1. Open **led_ip_v2_1_0.mpd** in the **pcores\led_ip_v1_00_a\data** under the **lab3** directory.

3-1-2. Add following line before the **S_AXI_ACLK** port under the **Ports** section.

PORT LED="", DIR=O, VEC=[7:0]



```
## Ports
PORT LED = "", DIR = O, VEC=[7:0]
PORT S_AXI_ACLK = "", DIR = I, SIGIS = CLK, BUS = S_AXI
PORT S_AXI_ARESETN = ARESETN, DIR = I, SIGIS = RST, BUS =
PORT S_AXI_AWADDR = AWADDR, DIR = I, VEC = [(C_S_AXI_ADDR
PORT S_AXI_AWVALID = AWVALID, DIR = I, BUS = S_AXI
PORT S_AXI_WDATA = WDATA, DIR = I, VEC = [(C_S_AXI_DATA_W
PORT S_AXI_WSTRB = WSTRB, DIR = I, VEC = [((C_S_AXI_DATA_
PORT S_AXI_WVALID = WVALID, DIR = I, BUS = S_AXI
```

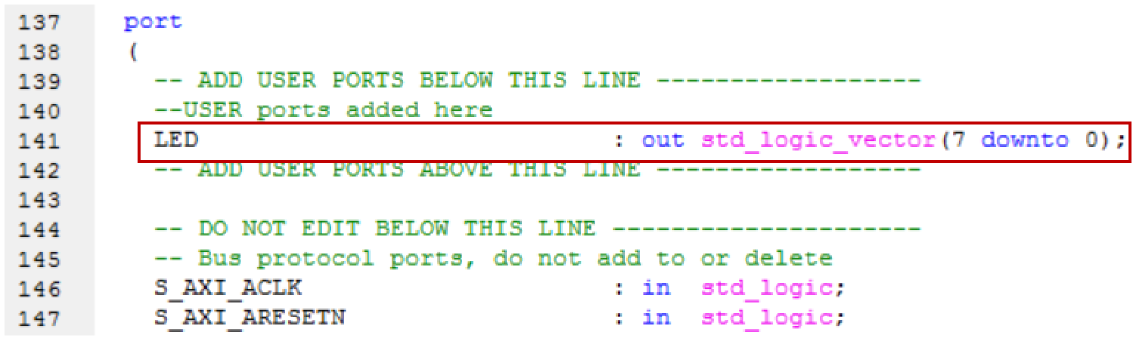
Figure 11. Update the MPD file for the LED Controller Peripheral

3-1-3. Save the file and close it.

3-2. Create the LED controller using the appropriate HDL template files generated from the Create/Import peripheral wizard: led_ip.vhd and user_logic.vhd. You can edit these files using a standard text editor.

3-2-1. Open **led_ip.vhd** in the **pcores\ led_ip_v1_00_a\hdl\vhdl** directory.

3-2-2. Add user port **LED** of width 8 under **USER ports added here** token by search for **--USER ports added here.**

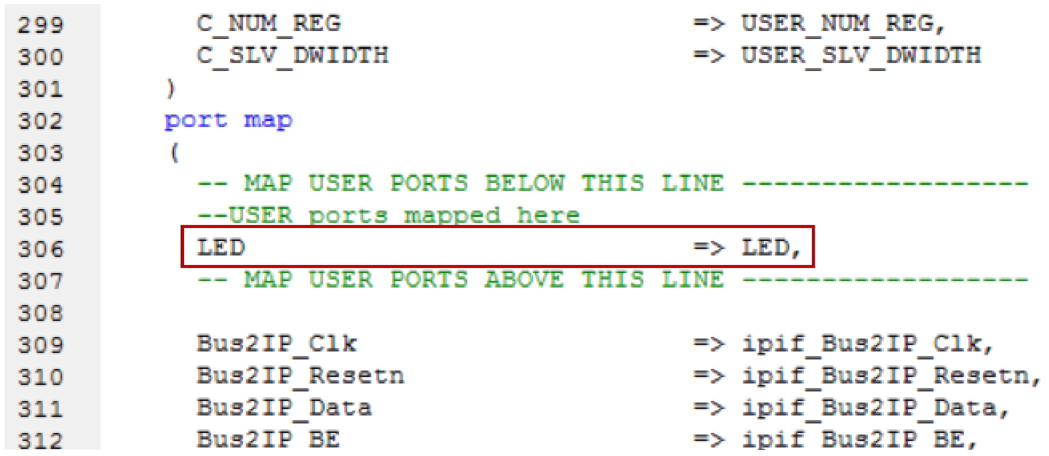


```
port
(
  -- ADD USER PORTS BELOW THIS LINE ------------------
  --USER ports added here
  LED                                : out std_logic_vector(7 downto 0);
  -- ADD USER PORTS ABOVE THIS LINE ------------------

  -- DO NOT EDIT BELOW THIS LINE ---------------------
  -- Bus protocol ports, do not add to or delete
  S_AXI_ACLK                         : in  std_logic;
  S_AXI_ARESETN                      : in  std_logic;
```

Figure 12. Add the user port LED

3-2-3. Search for next **--USER** and add port mapping statement, save the file and then close it.

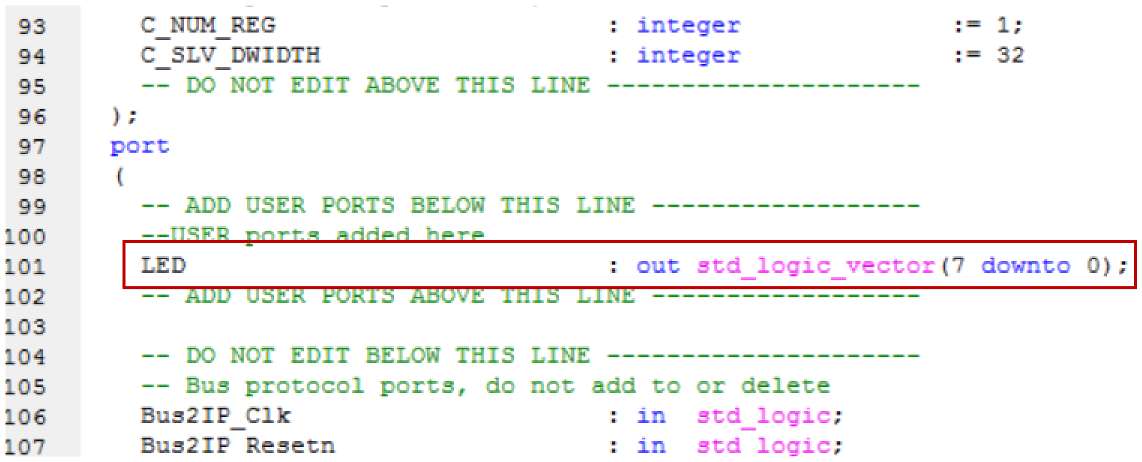


```
      C_NUM_REG                          => USER_NUM_REG,
      C_SLV_DWIDTH                       => USER_SLV_DWIDTH
    )
    port map
    (
      -- MAP USER PORTS BELOW THIS LINE ------------------
      --USER ports mapped here
      LED                                => LED,
      -- MAP USER PORTS ABOVE THIS LINE ------------------

      Bus2IP_Clk                         => ipif_Bus2IP_Clk,
      Bus2IP_Resetn                      => ipif_Bus2IP_Resetn,
      Bus2IP_Data                        => ipif_Bus2IP_Data,
      Bus2IP_BE                          => ipif_Bus2IP_BE,
```

Figure 13. Add port mapping statement

3-2-4. Open **user_logic.vhd** file from the *vhdl* directory and add **LED** port definition in the USER Ports area.

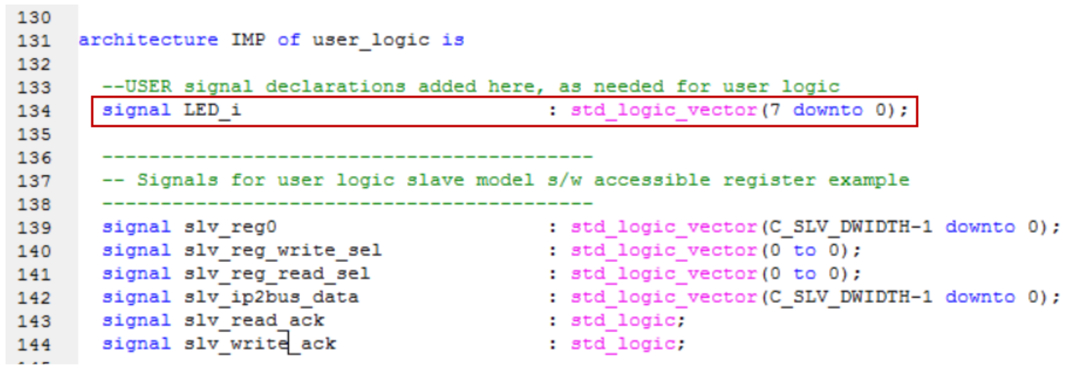


```
    C_NUM_REG                      : integer              := 1;
    C_SLV_DWIDTH                   : integer              := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    LED                            : out std_logic_vector(7 downto 0);
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Resetn                  : in  std_logic;
```

Figure 14. Add the LED port definition

3-2-5. Search for next **--USER** and the enter the internal signal declaration according to the figure below.



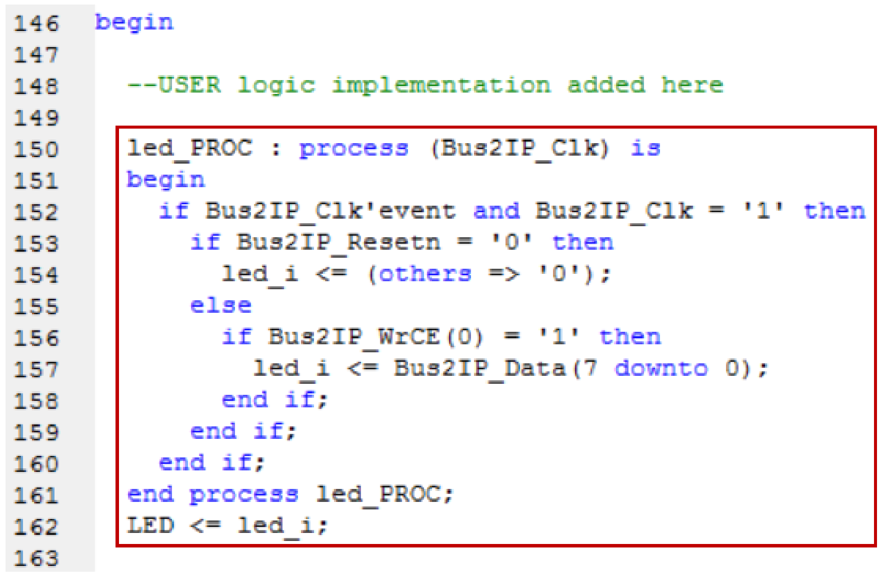
```
architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
  signal LED_i                          : std_logic_vector(7 downto 0);

  ------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ------------------------------------------
  signal slv_reg0                       : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg_write_sel              : std_logic_vector(0 to 0);
  signal slv_reg_read_sel               : std_logic_vector(0 to 0);
  signal slv_ip2bus_data                : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;
```

Figure 15. Internal signal declaration for the User Logic

3-2-6. Search for **--USER logic implementation** and add the following code or copy it from lab3_user_logic.vhd file located at **c:\xup\embedded\sources** directory.



```
begin

  --USER logic implementation added here

  led_PROC : process (Bus2IP_Clk) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Resetn = '0' then
        led_i <= (others => '0');
      else
        if Bus2IP_WrCE(0) = '1' then
          led_i <= Bus2IP_Data(7 downto 0);
        end if;
      end if;
    end if;
  end process led_PROC;
  LED <= led_i;
```

Figure 16. Add code

3-2-7. Save changes and close the **user_logic.vhd**

3-2-8. Select **Project** > **Rescan User Repositories** to have the changes in effect.

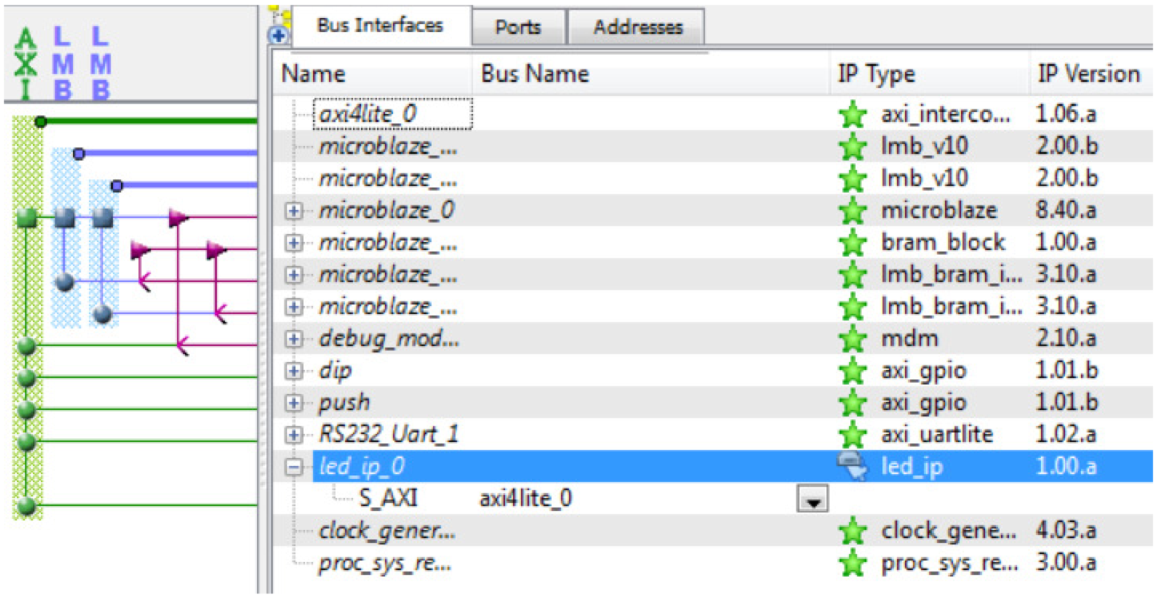
Add and Connect the Peripheral Step 4

4-1. Add the LED to the design and connect to the AXI4Lite bus in the System Assembly View. Make internal and external port connections. Assign an address range to it. Establish the LED data port as external FPGA pins.

4-1-1. Add the **led_ip** core to the system by double-clicking its entry in the IP Catalog.

4-1-2. Click **Yes** to add the IP instance, and **OK** twice to connect to the processor system using the default settings

4-1-3. Select the **Bus Interfaces** tab in the System Assembly View and verify that the peripheral is connected to axi4lite_0.



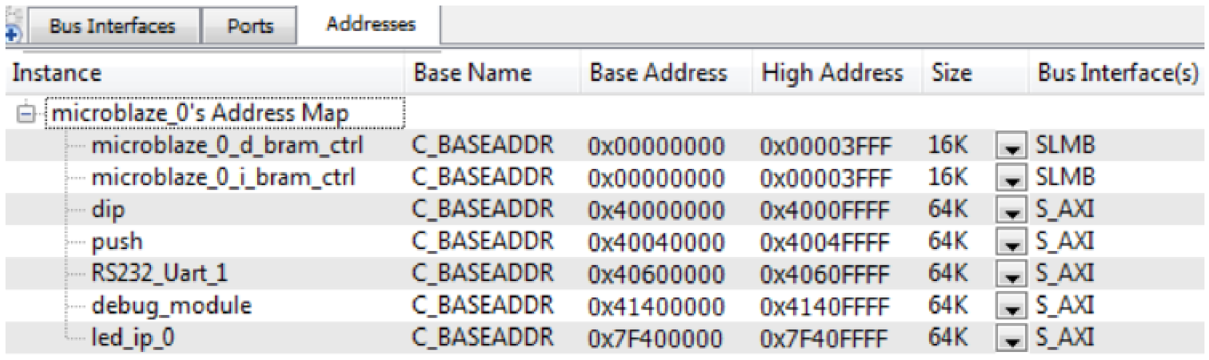
Name	Bus Name	IP Type	IP Version
axi4lite_0		axi_interco...	1.06.a
microblaze_...		lmb_v10	2.00.b
microblaze_...		lmb_v10	2.00.b
microblaze_0		microblaze	8.40.a
microblaze_...		bram_block	1.00.a
microblaze_...		lmb_bram_i...	3.10.a
microblaze_...		lmb_bram_i...	3.10.a
debug_mod...		mdm	2.10.a
dip		axi_gpio	1.01.b
push		axi_gpio	1.01.b
RS232_Uart_1		axi_uartlite	1.02.a
led_ip_0		led_ip	1.00.a
S_AXI	axi4lite_0		
clock_gener...		clock_gene...	4.03.a
proc_sys_re...		proc_sys_re...	3.00.a

Figure 17. Making bus connection

4-1-4. Select the **Ports** tab, right-click the **led** port of the **led_ip_0** instance and select **Make External**.

4-1-5. Select **Addresses** filter and verify that the address is assigned to the led_ip instance.

The Addresses tab should look similar to as shown.



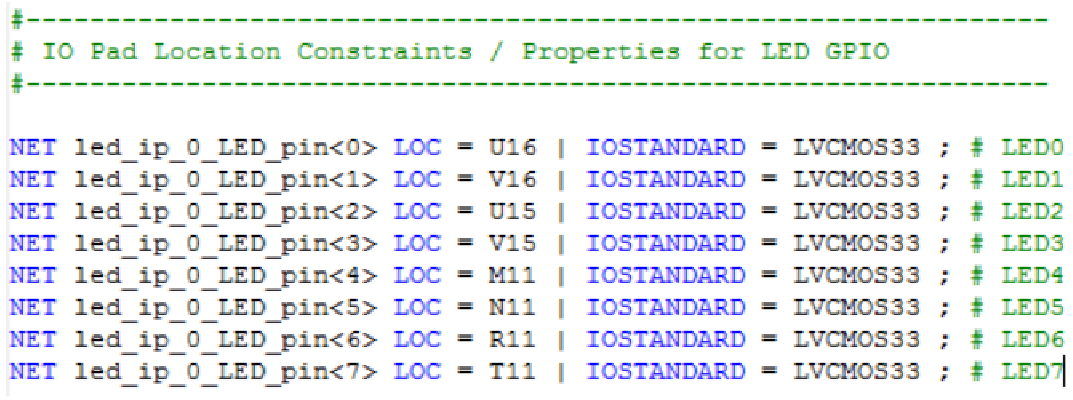
Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)
microblaze_0's Address Map					
microblaze_0_d_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB
microblaze_0_i_bram_ctrl	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB
dip	C_BASEADDR	0x40000000	0x4000FFFF	64K	S_AXI
push	C_BASEADDR	0x40040000	0x4004FFFF	64K	S_AXI
RS232_Uart_1	C_BASEADDR	0x40600000	0x4060FFFF	64K	S_AXI
debug_module	C_BASEADDR	0x41400000	0x4140FFFF	64K	S_AXI
led_ip_0	C_BASEADDR	0x7F400000	0x7F40FFFF	64K	S_AXI

Figure 18. Generate Addresses

4-2. Modify the system.ucf file to assign external LED controller connections to the proper FPGA pin locations.

4-2-1. Open the **system.ucf** file by double-clicking the **UCF File: data\system.ucf** entry under Project Files in the Project tab.

4-2-2. Open the ***C:\xup\embedded\sources\lab3.ucf*** file and copy the pin assignments under the original ucf file.



```
#-----------------------------------------------------------------
# IO Pad Location Constraints / Properties for LED GPIO
#-----------------------------------------------------------------

NET led_ip_0_LED_pin<0> LOC = U16 | IOSTANDARD = LVCMOS33 ; # LED0
NET led_ip_0_LED_pin<1> LOC = V16 | IOSTANDARD = LVCMOS33 ; # LED1
NET led_ip_0_LED_pin<2> LOC = U15 | IOSTANDARD = LVCMOS33 ; # LED2
NET led_ip_0_LED_pin<3> LOC = V15 | IOSTANDARD = LVCMOS33 ; # LED3
NET led_ip_0_LED_pin<4> LOC = M11 | IOSTANDARD = LVCMOS33 ; # LED4
NET led_ip_0_LED_pin<5> LOC = N11 | IOSTANDARD = LVCMOS33 ; # LED5
NET led_ip_0_LED_pin<6> LOC = R11 | IOSTANDARD = LVCMOS33 ; # LED6
NET led_ip_0_LED_pin<7> LOC = T11 | IOSTANDARD = LVCMOS33 ; # LED7
```

Figure 19. Adding UCF constraints

4-2-3. Save and close the file

4-2-4. Click on **Hardware > Generate Bitstream.**

4-2-5. Close the project after Bitstream generation is done.

We will do the software development in Lab 4 and verify the functionality in hardware.

Conclusion

The Create and Import Peripheral Wizard was used to create peripheral templates for the AXI4Lite interface. Logic was added to the templates to create an LED interface peripheral. The peripheral was then integrated into an existing processor system.

Completed MHS File

```
# ##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 14.2 Build EDK_P.28xd
# Wed Sep 19 09:57:18 2012
# Target Board:  digilent nexys3 Rev B
# Family:    spartan6
# Device:    xc6slx16
# Package:   csg324
# Speed Grade:  -3
# ##############################################################################
 PARAMETER VERSION = 2.1.0


 PORT RS232_Uart_1_sout = RS232_Uart_1_sout, DIR = O
 PORT RS232_Uart_1_sin = RS232_Uart_1_sin, DIR = I
 PORT RESET = RESET, DIR = I, SIGIS = RST, RST_POLARITY = 1
 PORT GCLK = GCLK, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
 PORT dip_GPIO_IO_I_pin = dip_GPIO_IO_I, DIR = I, VEC = [7:0]
 PORT push_GPIO_IO_I_pin = push_GPIO_IO_I, DIR = I, VEC = [3:0]
 PORT led_ip_0_LED_pin = led_ip_0_LED, DIR = O, VEC = [7:0]


BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT MB_Debug_Sys_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
 PORT Dcm_locked = proc_sys_reset_0_Dcm_locked
 PORT MB_Reset = proc_sys_reset_0_MB_Reset
 PORT Slowest_sync_clk = clk_100_0000MHz
 PORT Interconnect_aresetn = proc_sys_reset_0_Interconnect_aresetn
 PORT Ext_Reset_In = RESET
 PORT BUS_STRUCT_RESET = proc_sys_reset_0_BUS_STRUCT_RESET
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_ilmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHz
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_i_bram_ctrl
 PARAMETER HW_VER = 3.10.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00003fff
 BUS_INTERFACE SLMB = microblaze_0_ilmb
 BUS_INTERFACE BRAM_PORT = microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_dlmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHz
END
```

```
BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_d_bram_ctrl
 PARAMETER HW_VER = 3.10.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00003fff
 BUS_INTERFACE SLMB = microblaze_0_dlmb
 BUS_INTERFACE BRAM_PORT = microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN bram_block
 PARAMETER INSTANCE = microblaze_0_bram_block
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
 BUS_INTERFACE PORTB = microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 8.40.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0X00000000
 PARAMETER C_ICACHE_HIGHADDR = 0X3FFFFFFF
 PARAMETER C_USE_ICACHE = 0
 PARAMETER C_ICACHE_ALWAYS_USED = 0
 PARAMETER C_DCACHE_BASEADDR = 0X00000000
 PARAMETER C_DCACHE_HIGHADDR = 0X3FFFFFFF
 PARAMETER C_USE_DCACHE = 0
 PARAMETER C_DCACHE_ALWAYS_USED = 0
 BUS_INTERFACE ILMB = microblaze_0_ilmb
 BUS_INTERFACE DLMB = microblaze_0_dlmb
 BUS_INTERFACE M_AXI_DP = axi4lite_0
 BUS_INTERFACE DEBUG = microblaze_0_debug
 PORT MB_RESET = proc_sys_reset_0_MB_Reset
 PORT CLK = clk_100_0000MHz
END

BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 2.10.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_UART = 1
 PARAMETER C_BASEADDR = 0x41400000
 PARAMETER C_HIGHADDR = 0x4140ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_debug
 PORT Debug_SYS_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
 PORT S_AXI_ACLK = clk_100_0000MHz
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER HW_VER = 4.03.a
 PARAMETER C_CLKIN_FREQ = 100000000
 PARAMETER C_CLKOUT0_FREQ = 100000000
 PARAMETER C_CLKOUT0_GROUP = NONE
 PORT LOCKED = proc_sys_reset_0_Dcm_locked
 PORT CLKOUT0 = clk_100_0000MHz
 PORT RST = RESET
 PORT CLKIN = GCLK
END

BEGIN axi_interconnect
 PARAMETER INSTANCE = axi4lite_0
 PARAMETER HW_VER = 1.06.a
 PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
 PORT interconnect_aclk = clk_100_0000MHz
 PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
END

BEGIN axi_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 115200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 1
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHz
 PORT TX = RS232_Uart_1_sout
 PORT RX = RS232_Uart_1_sin
END

BEGIN axi_gpio
 PARAMETER INSTANCE = dip
 PARAMETER HW_VER = 1.01.b
 PARAMETER C_GPIO_WIDTH = 8
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_BASEADDR = 0x40000000
 PARAMETER C_HIGHADDR = 0x4000ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHz
 PORT GPIO_IO_I = dip_GPIO_IO_I
END

BEGIN axi_gpio
 PARAMETER INSTANCE = push
 PARAMETER HW_VER = 1.01.b
 PARAMETER C_GPIO_WIDTH = 4
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_BASEADDR = 0x40040000
 PARAMETER C_HIGHADDR = 0x4004ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHz
 PORT GPIO_IO_I = push_GPIO_IO_I
END

BEGIN led_ip
 PARAMETER INSTANCE = led_ip_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x7f400000
 PARAMETER C_HIGHADDR = 0x7f40ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHz
 PORT LED = led_ip_0_LED
END
```