LAB 3.

Adding Custom IP to an Embedded System

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 bitstream and verify operation in hardware

Design Description

You will extend the Lab 2 hardware design by creating and adding a PLB peripheral (refer to MYIP in Figure 1) to the system, and connecting it to the LCD on the Spartan-3E kit.

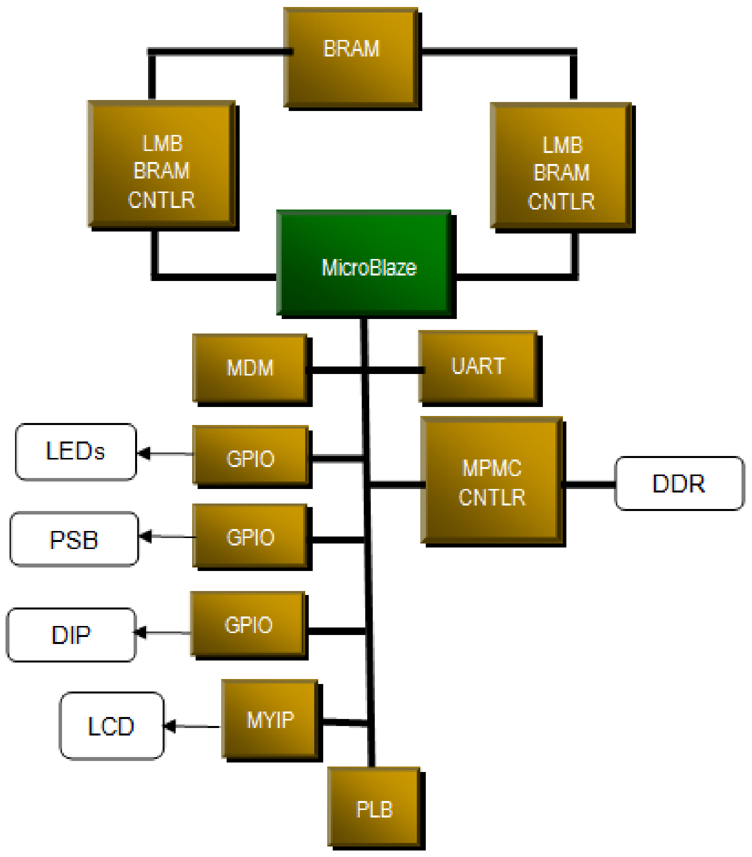


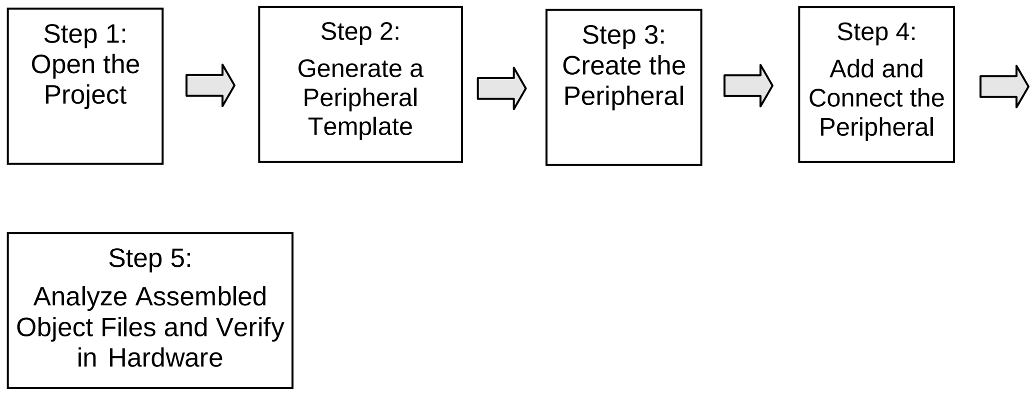
Figure 1. Design updated from previous lab

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 5 primary steps: You will open the project, generate a peripheral templates, create the peripheral, connect the peripheral 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


Step 1:
Open the Project
Step 2:
Generate a Peripheral Template
Step 3:
Create the Peripheral
Step 4:
Add and Connect the Peripheral
Step 5:
Analyze Assembled Object Files and Verify in Hardware

Open the Project Step 1

1-1. Create a *lab3* folder and copy the contents of the *lab2* 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 **D:\** directory and copy the contents from *lab2* to *lab3.*

1-1-2. Open XPS by clicking **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:\lab3***.*

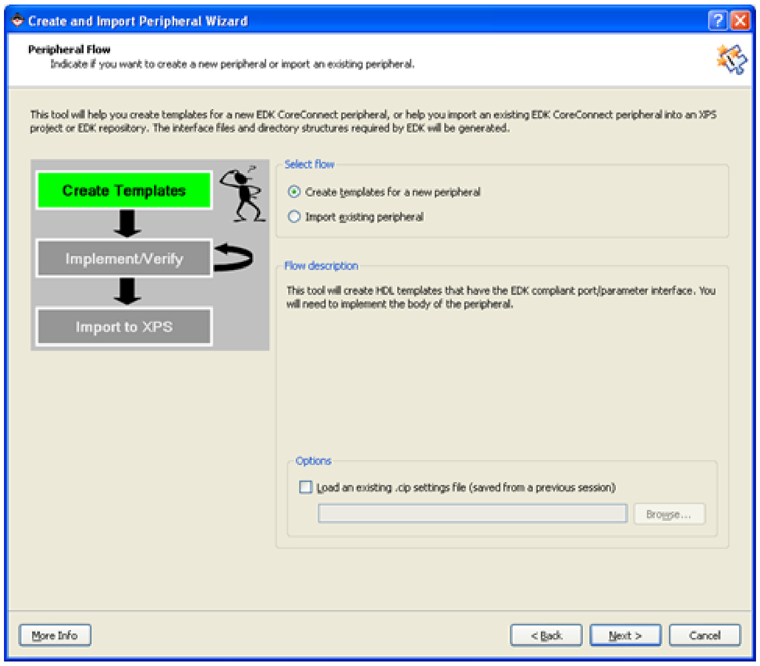
1-1-4. Click **system.xmp** to open the project.

Generate a Peripheral Template Step 2

2-1. You will use the Create/Import Peripheral Wizard to create a PLB bus 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 (**Figure 2**).



Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK CoreConnect peripheral, or help you import an existing EDK CoreConnect peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow
[✓ Create templates for a new peripheral]
Import existing peripheral

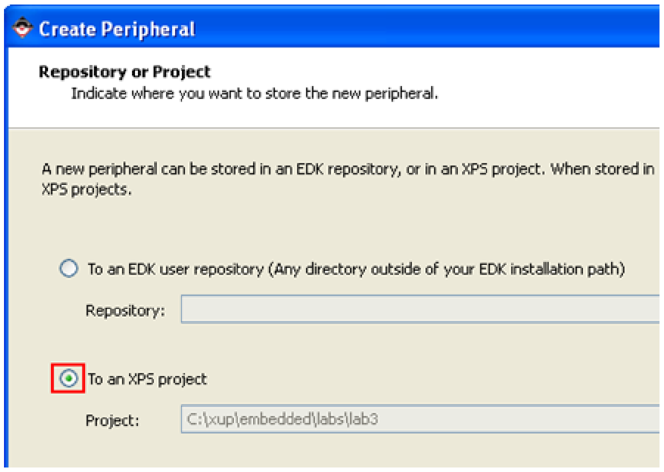
Flow description
This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

Options
☐ Load an existing .cip settings file (saved from a previous session)

Figure 2. Create and Import User Peripheral Dialog Box

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 (see **Figure 3**).


Create Peripheral

Repository or Project
Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in
XPS projects.

( ) To an EDK user repository (Any directory outside of your EDK installation path)

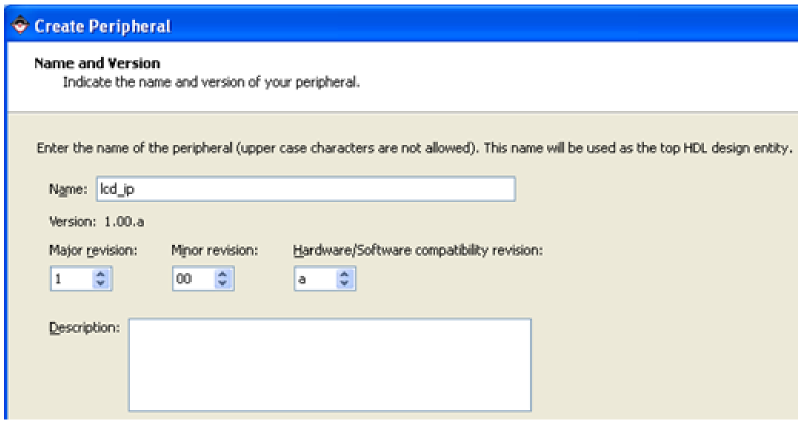
Repository:

(●) To an XPS project

Project: C:\xup\embedded\labs\lab3

Figure 3. Repository or Project Dialog Box

2-1-5. Click **Next** and enter ***lcd_ip*** in the Name field, leave the default version number of 1.00.a, click **Next** (see **Figure 4**).


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: lcd_ip

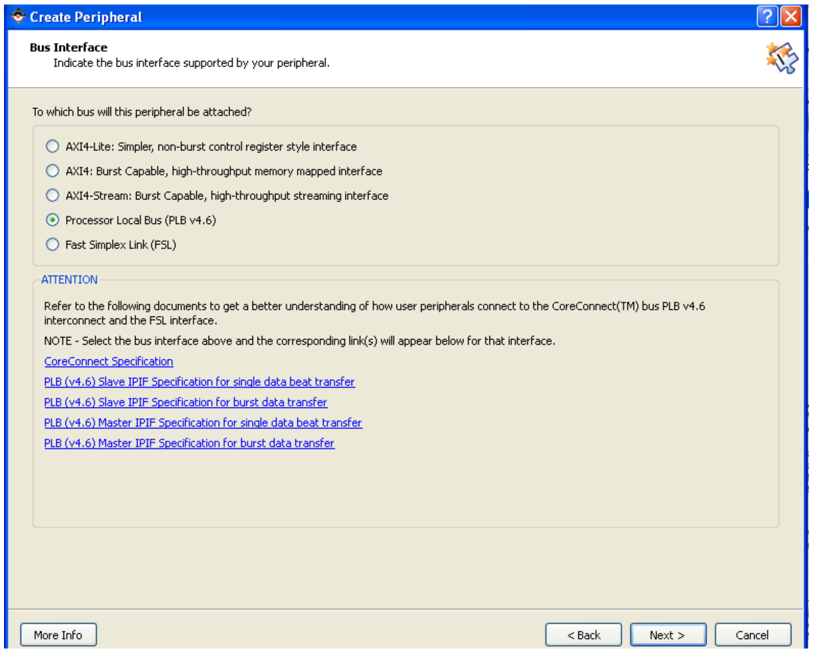
Version: 1.00.a

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

Description:

Figure 4. Provide Core Name and Version Number

2-1-6. Select **Processor Local Bus (PLB v4.6)**, 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
- ● Processor Local Bus (PLB v4.6)
- ○ Fast Simplex Link (FSL)

ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) bus PLB v4.6 interconnect and the FSL interface.

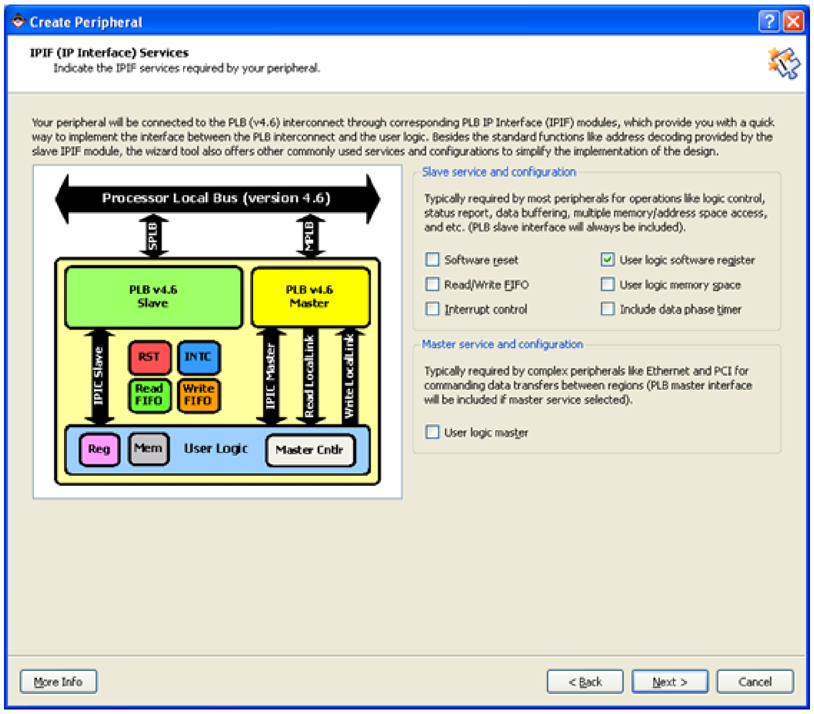
NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.

CoreConnect Specification
PLB (v4.6) Slave IPIF Specification for single data beat transfer
PLB (v4.6) Slave IPIF Specification for burst data transfer
PLB (v4.6) Master IPIF Specification for single data beat transfer
PLB (v4.6) Master IPIF Specification for burst data transfer

Figure 5. Select the PLB bus

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 D:\lab3 directory and ensure the structure.

2-2-1. In the IPIF Services panel, deselect **Include data phase timer** and click **Next.**



Create Peripheral

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

Your peripheral will be connected to the PLB (v4.6) interconnect through corresponding PLB IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the PLB 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.

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. (PLB slave interface will always be included).

- ☐ Software reset
- ☑ User logic software register
- ☐ Read/Write FIFO
- ☐ User logic memory space
- ☐ Interrupt control
- ☐ Include data phase timer

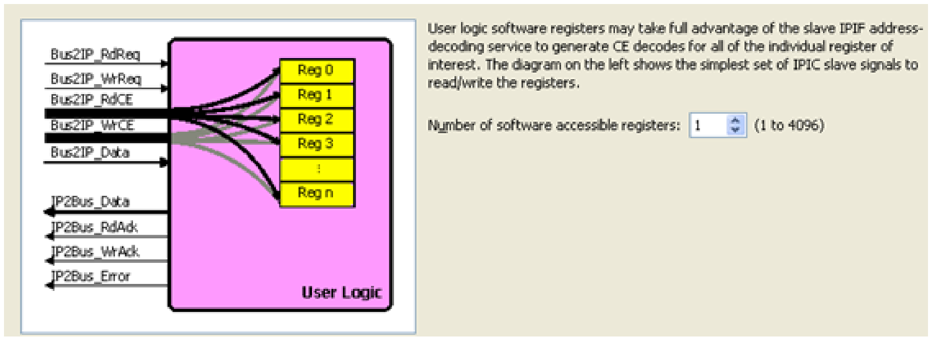
Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected).

- ☐ User logic master

Figure 6. IPIF Services Dialogue Box

2-2-2. Click **Next**, accepting the default data width, and no burst and cache line support. Click **Next** to accept default number of registers (one) (see **Figure 7**).

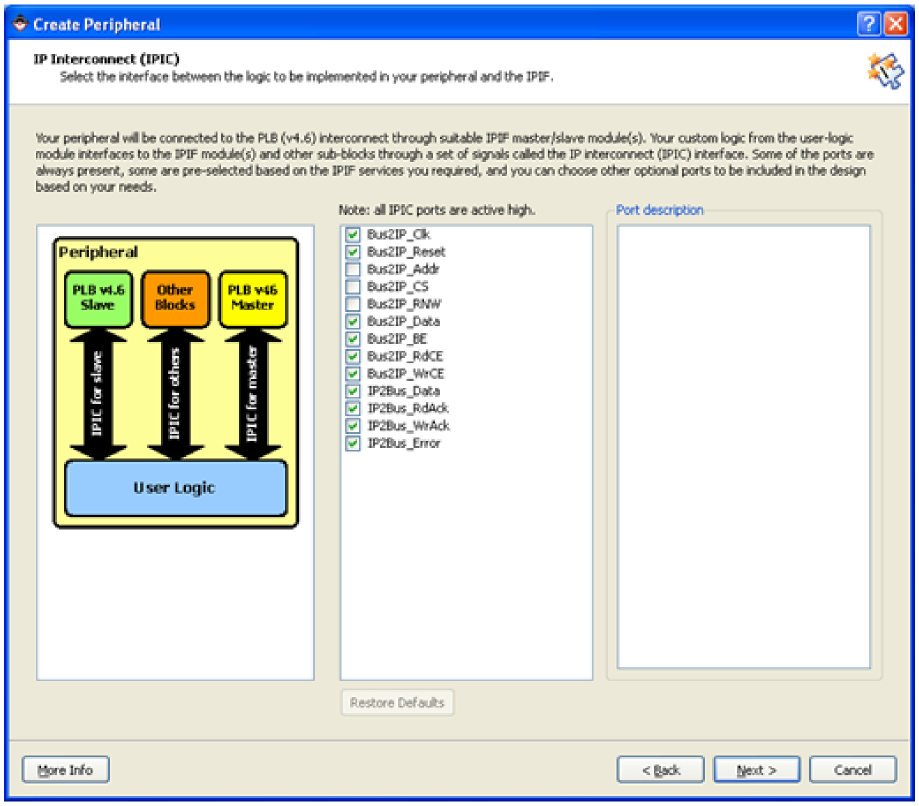


User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIC slave signals to read/write the registers.

Number of software accessible registers: 1 (1 to 4096)

Figure 7. 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 (see **Figure 8**). Click **Next.**



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

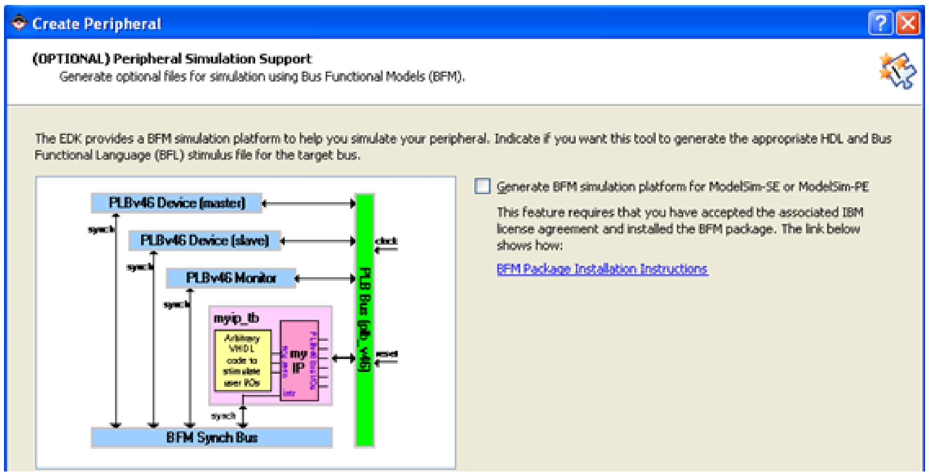
Your peripheral will be connected to the PLB (v4.6) interconnect through suitable IPIF master/slave module(s). Your custom logic from the user-logic module interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present, some are pre-selected based on the IPIF services you required, and you can choose other optional ports to be included in the design based on your needs.

Note: all IPIC ports are active high.

- ☑ Bus2IP_Clk
- ☑ Bus2IP_Reset
- ☐ Bus2IP_Addr
- ☐ Bus2IP_CS
- ☐ Bus2IP_RNW
- ☑ Bus2IP_Data
- ☑ Bus2IP_BE
- ☑ Bus2IP_RdCE
- ☑ Bus2IP_WrCE
- ☑ IP2Bus_Data
- ☑ IP2Bus_RdAck
- ☑ IP2Bus_WrAck
- ☑ IP2Bus_Error

Figure 8. IP Interconnect (IPIC) Dialog Box

2-2-4. In the **(OPTIONAL) Peripheral Simulation Support** panel, leave **Generate BFM simulation platform** unchecked (see Figure 9), and click **Next**.


Create Peripheral

(OPTIONAL) Peripheral Simulation Support

Generate optional files for simulation using Bus Functional Models (BFM).

The EDK provides a BFM simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate the appropriate HDL and Bus Functional Language (BFL) stimulus file for the target bus.

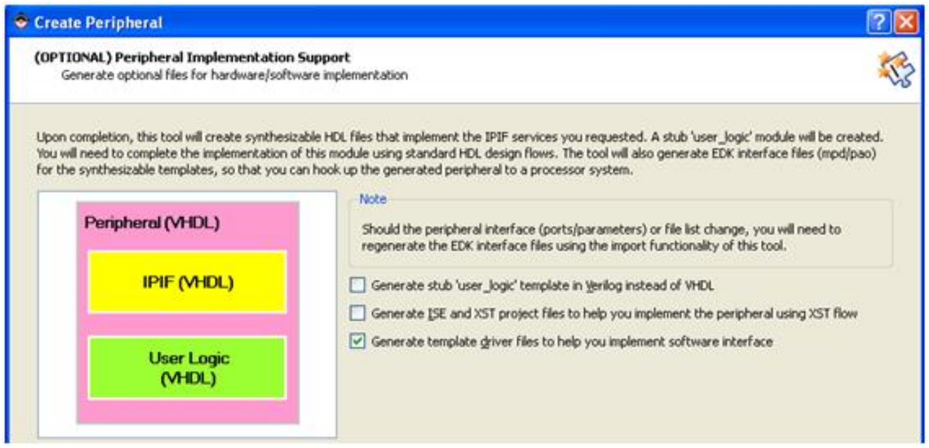
☐ Generate BFM simulation platform for ModelSim-SE or ModelSim-PE

This feature requires that you have accepted the associated IBM license agreement and installed the BFM package. The link below shows how:

BFM Package Installation Instructions

Figure 9. Peripheral Simulation Support Dialog Box

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 (see Figure 10).


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.

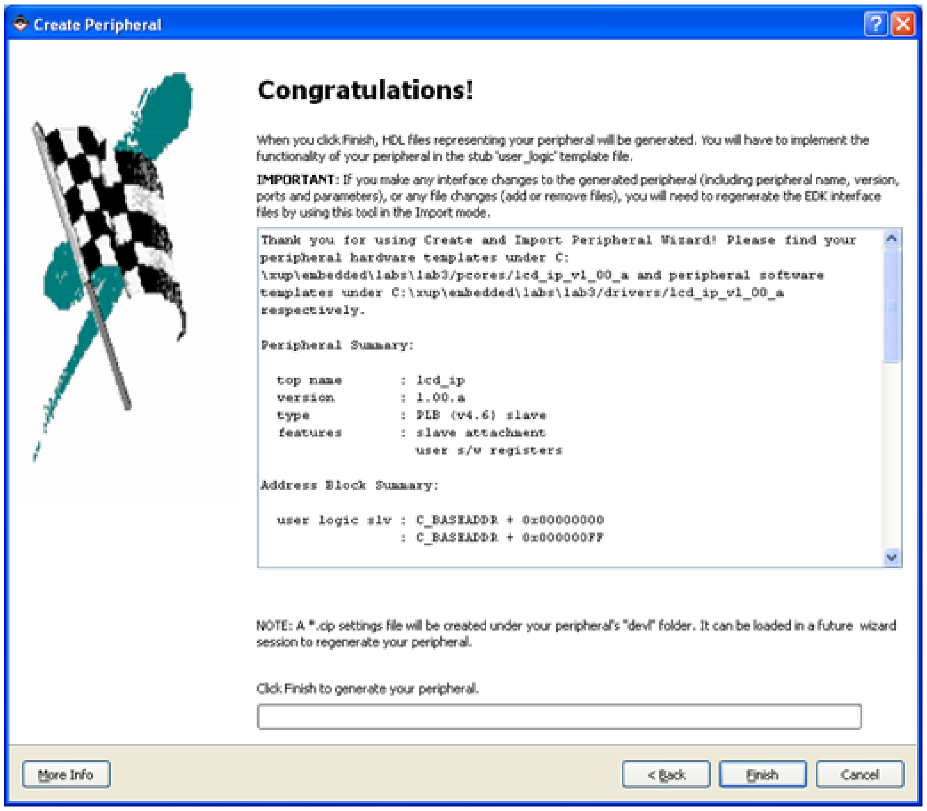
☐ Generate stub 'user_logic' template in Verilog instead of VHDL

☐ Generate ISE and XST project files to help you implement the peripheral using XST flow

☑ Generate template driver files to help you implement software interface

Figure 10. Peripheral Implementation Options Dialog Box

2-2-6. Click **Next**, and you will see the summary information panel (**Figure 11**).



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/lcd_ip_v1_00_a and peripheral software
templates under C:\xup\embedded\labs\lab3/drivers/lcd_ip_v1_00_a
respectively.

Peripheral Summary:

  top name       : lcd_ip
  version        : 1.00.a
  type           : PLB (v4.6) slave
  features       : slave attachment
                   user s/w registers

Address Block Summary:

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

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 11. 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 **lcd_ip** is added to the **Project Local pcores** repository (**Figure 12**).

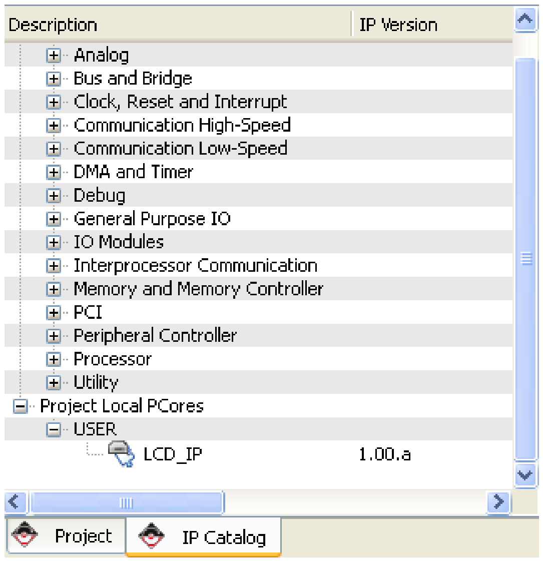


Figure 12. IP Catalog Updated Entry

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 13**).

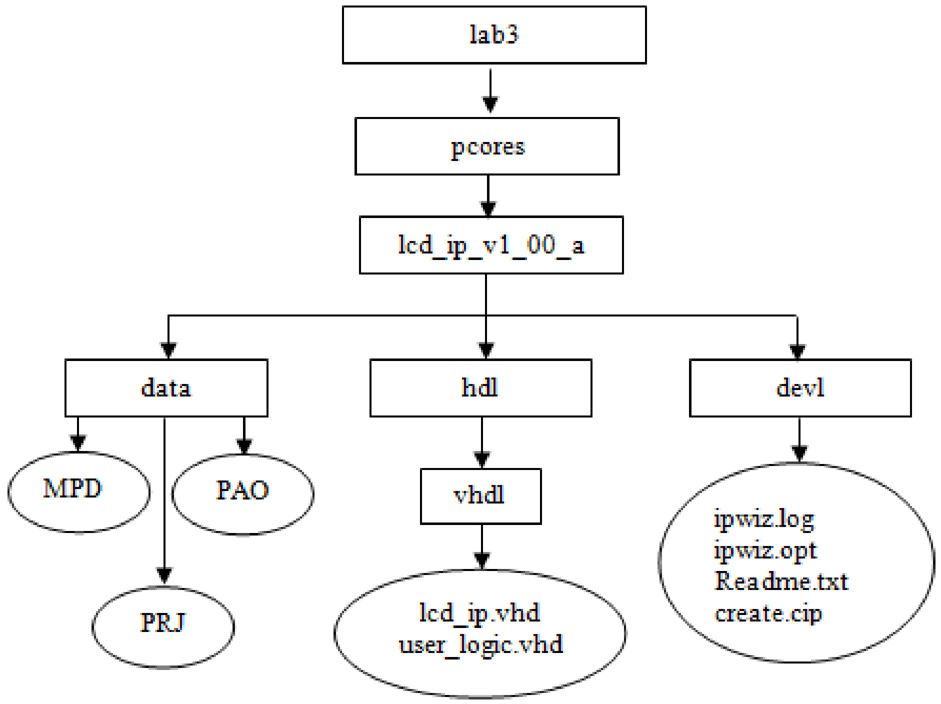


Figure 13. Structure Created by the Create and Import Peripheral Wizard

Create the Peripheral Step 3

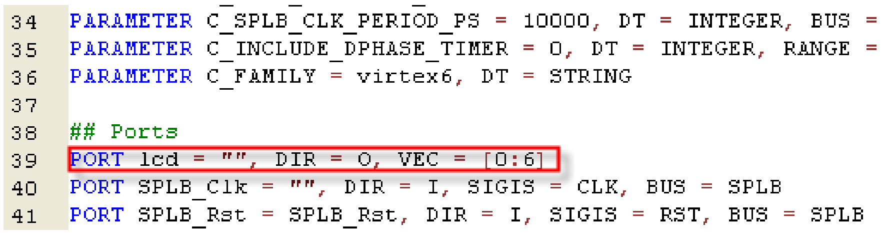
3-1. Update the MPD file to include the lcd data output of the LCD controller peripheral so the port can be connected in XPS.

Add a port called “lcd” to the MPD file.

3-1-1. Open **lcd_ip_v2_1_0.mpd** in the **pcores\ lcd_ip_v1_00_a** data under **lab3** directory.

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

PORT lcd=””,dir=O,VEC=[0:6]



```
PARAMETER C_SPLB_CLK_PERIOD_PS = 10000, DT = INTEGER, BUS =
PARAMETER C_INCLUDE_DPHASE_TIMER = 0, DT = INTEGER, RANGE =
PARAMETER C_FAMILY = virtex6, DT = STRING

## Ports
PORT lcd = "", DIR = O, VEC = [0:6]
PORT SPLB_Clk = "", DIR = I, SIGIS = CLK, BUS = SPLB
PORT SPLB_Rst = SPLB_Rst, DIR = I, SIGIS = RST, BUS = SPLB
```

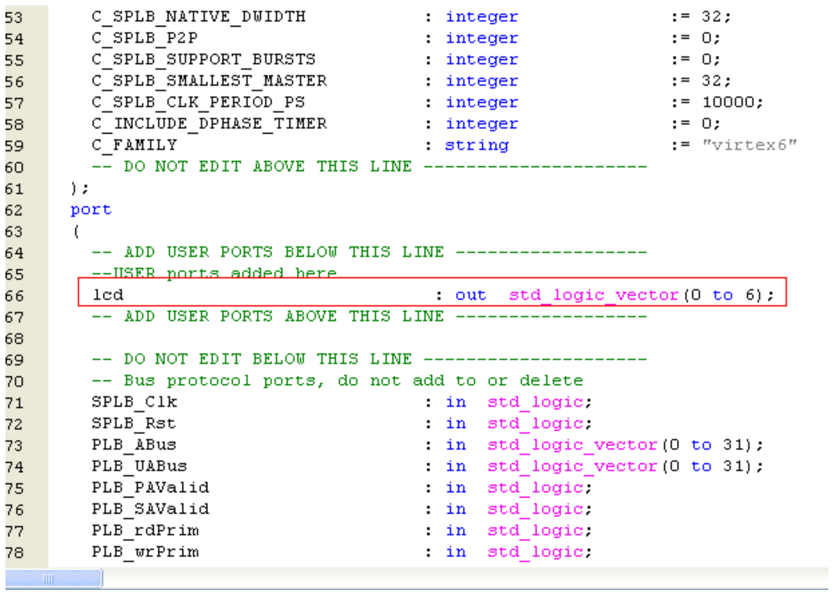
Figure 14. Update the MPD file for the LCD Controller Peripheral

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

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

3-2-1. Open **lcd_ip.vhd** in the **pcores\ lcd_ip_v1_00_a\ hdl\ vhdl** directory.

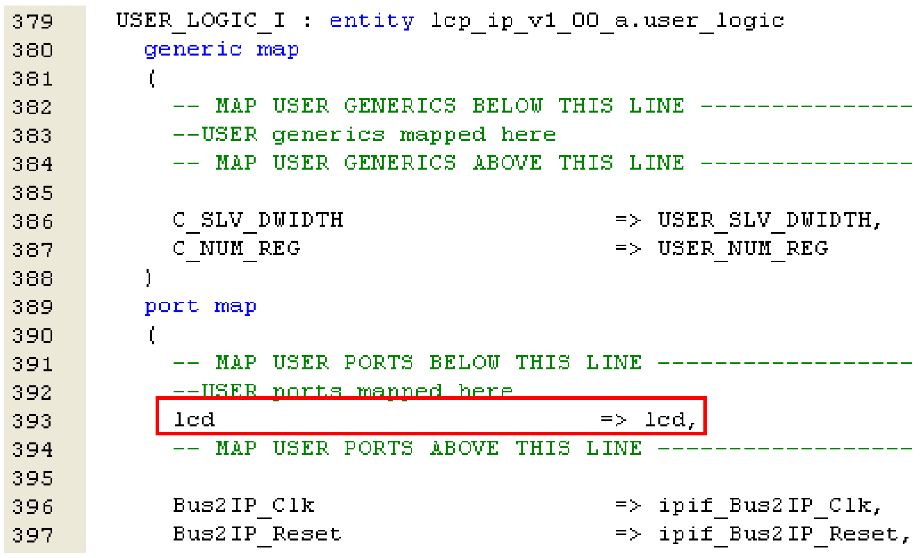
3-2-2. Add user port **lcd** of width 7 under **USER ports added here** token (see **Figure 15**).



```
    C_SPLB_NATIVE_DWIDTH          : integer              := 32;
    C_SPLB_P2P                    : integer              := 0;
    C_SPLB_SUPPORT_BURSTS         : integer              := 0;
    C_SPLB_SMALLEST_MASTER        : integer              := 32;
    C_SPLB_CLK_PERIOD_PS          : integer              := 10000;
    C_INCLUDE_DPHASE_TIMER        : integer              := 0;
    C_FAMILY                      : string               := "virtex6"
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    lcd                             : out  std_logic_vector(0 to 6);
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    SPLB_Clk                      : in  std_logic;
    SPLB_Rst                      : in  std_logic;
    PLB_ABus                      : in  std_logic_vector(0 to 31);
    PLB_UABus                     : in  std_logic_vector(0 to 31);
    PLB_PAValid                   : in  std_logic;
    PLB_SAValid                   : in  std_logic;
    PLB_rdPrim                    : in  std_logic;
    PLB_wrPrim                    : in  std_logic;
```

Figure 15. Add the User Port LCD

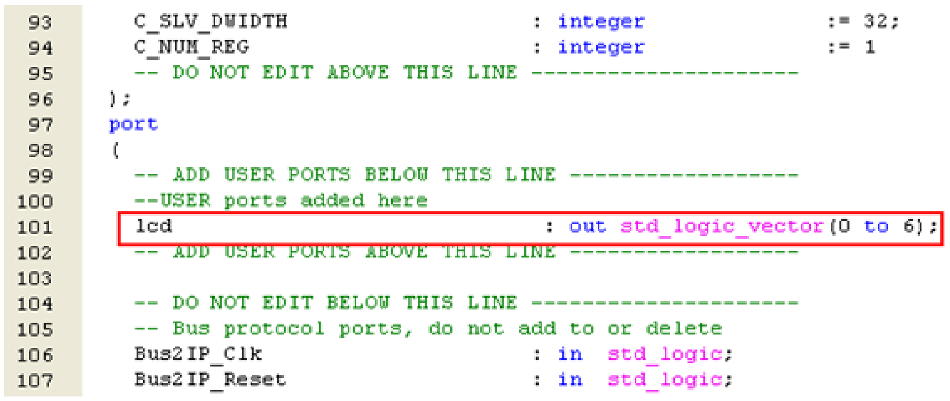
3-2-3. Search for next **--USER** and add port mapping statement, save the file and then close it.



```
USER_LOGIC_I : entity lcp_ip_v1_00_a.user_logic
  generic map
  (
    -- MAP USER GENERICS BELOW THIS LINE ---------------
    --USER generics mapped here
    -- MAP USER GENERICS ABOVE THIS LINE ---------------

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

    Bus2IP_Clk                     => ipif_Bus2IP_Clk,
    Bus2IP_Reset                   => ipif_Bus2IP_Reset,
```

Figure 16. Add Port Mapping Statement

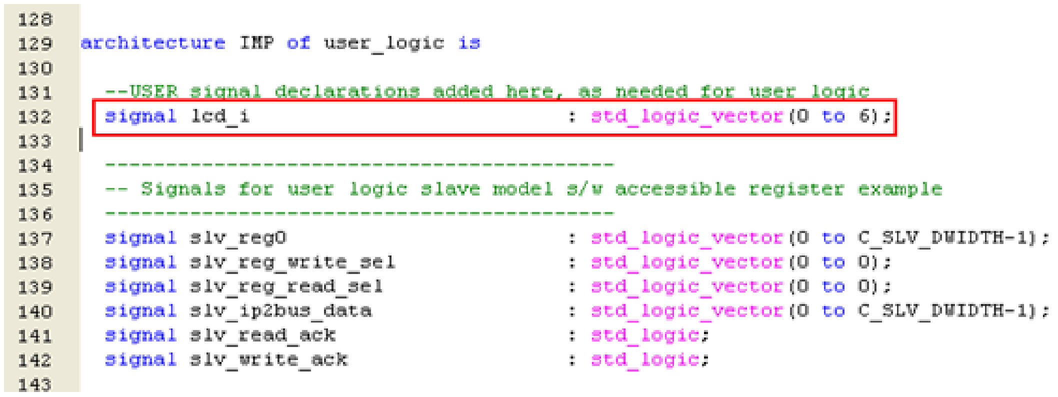
3-2-4. Open **user_logic.vhd** file from the *vhdl* directory and add **lcd** port definition in the USER Ports area.



```
    C_SLV_DWIDTH                   : integer              := 32;
    C_NUM_REG                      : integer              := 1
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    lcd                              : out std_logic_vector(0 to 6);
    -- 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_Reset                   : in  std_logic;
```

Figure 17. Add the lcd Port Definition

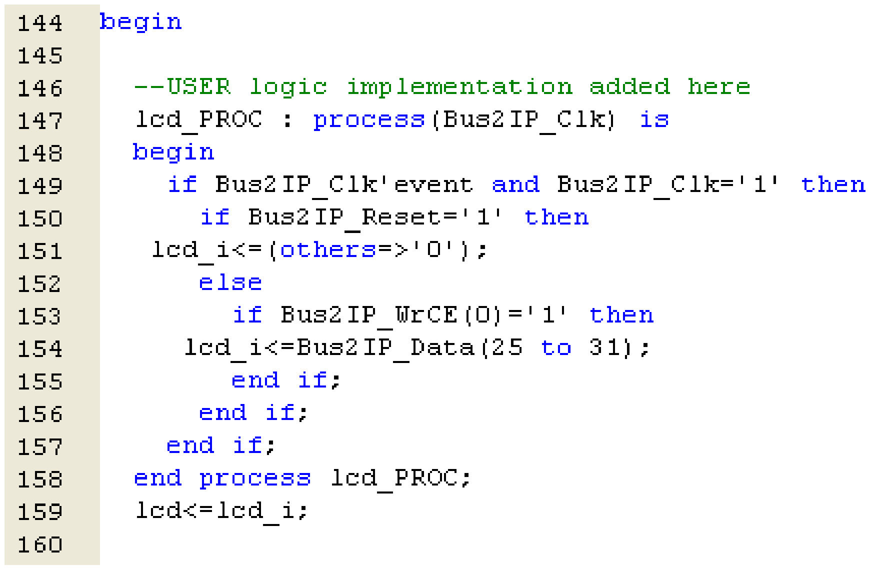
3-2-5. Search for next **--USER** and the enter the internal signal declaration according to the figure below



```

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
  signal lcd_i                          : std_logic_vector(0 to 6);

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

```

Figure 18. 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 to **W:\A.Bukowiec\zajecia\SW\source** directory.



```vhdl
begin

  --USER logic implementation added here
  lcd_PROC : process(Bus2IP_Clk) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk='1' then
      if Bus2IP_Reset='1' then
   lcd_i<=(others=>'0');
      else
        if Bus2IP_WrCE(0)='1' then
     lcd_i<=Bus2IP_Data(25 to 31);
        end if;
      end if;
    end if;
  end process lcd_PROC;
  lcd<=lcd_i;

```

Figure 19. Add Code

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

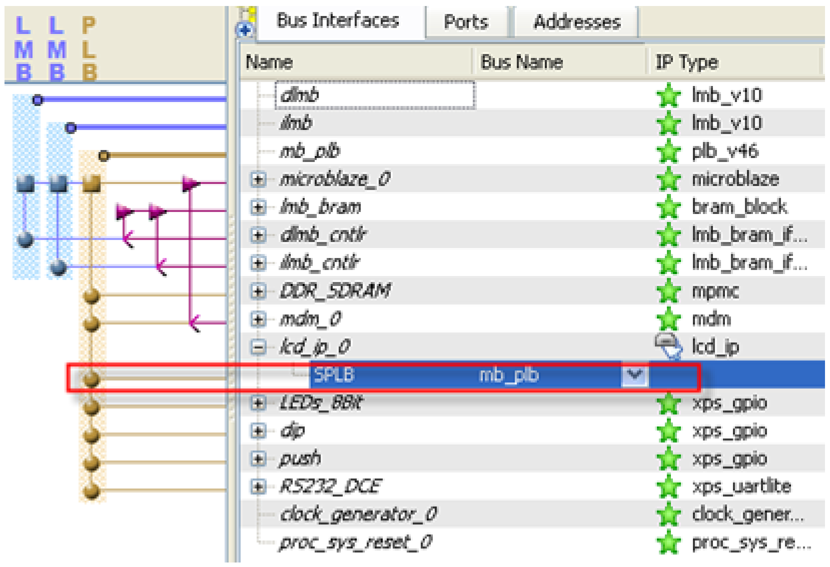
3-2-8. Select **Project → Rescan User Repositories** to have the changes in effect.

Add and Connect the Peripheral Step 4

4-1. Add and connect the LCD peripheral to the PLB bus in the System Assembly View. Make internal and external port connections. Assign an address range to it. Establish the LCD data port as external FPGA pins and assign the pin location constraints so the peripheral interfaces to the LCD display on the Spartan-3E Starter Kit.

4-1-1. In **IP Catalog**, select **lcd_ip** core, drag and drop it in the **System Assembly View** panel. Click OK to accept the default settings.

4-1-2. Make sure that the **Bus Interfaces** filter is selected in the System Assembly View and click on the circle in the bus connection diagram to make bus connection (**Figure 20**).



Name	Bus Name	IP Type
dlmb		lmb_v10
ilmb		lmb_v10
mb_plb		plb_v46
microblaze_0		microblaze
lmb_bram		bram_block
dlmb_cntlr		lmb_bram_if...
ilmb_cntlr		lmb_bram_if...
DDR_SDRAM		mpmc
mdm_0		mdm
lcd_ip_0		lcd_ip
SPLB	mb_plb	
LEDs_8Bit		xps_gpio
dip		xps_gpio
push		xps_gpio
RS232_DCE		xps_uartlite
clock_generator_0		clock_gener...
proc_sys_reset_0		proc_sys_re...

Figure 20. Making Bus Connection

4-1-3. Select the **Ports** filter, and connect the **lcd** port of the **lcd_ip_0** instance as an external pin by selecting **Make External (Figure 21).**

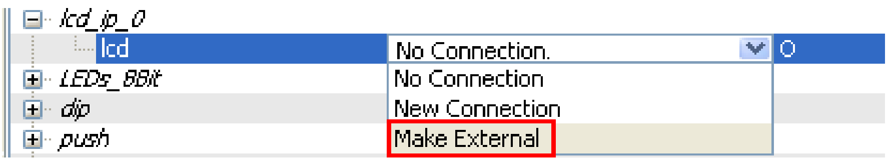
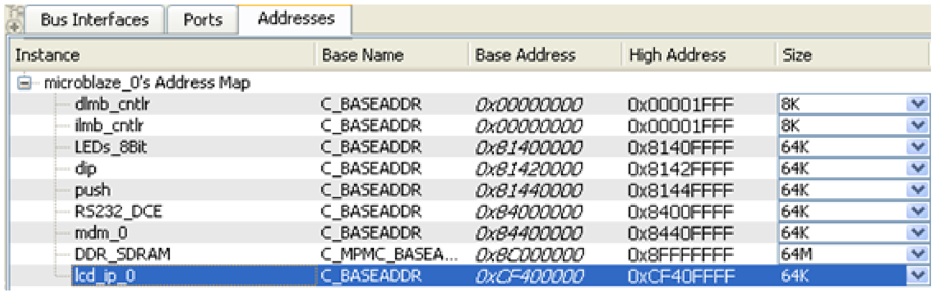


Figure 21. Assign the lcd_0 Instance

4-1-4. Select **Addresses** filter and click the **Generate Addresses** button.

Your results should look similar to that below (as shown in **Figure 22**).



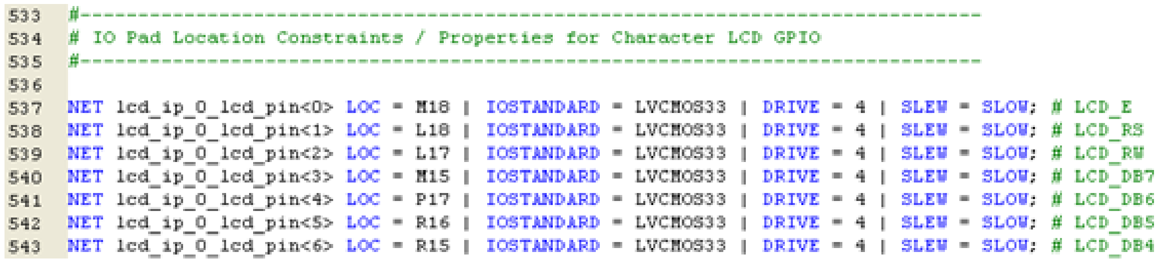
Instance	Base Name	Base Address	High Address	Size
microblaze_0's Address Map				
dlmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001FFF	8K
LEDs_8Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K
dip	C_BASEADDR	0x81420000	0x8142FFFF	64K
push	C_BASEADDR	0x81440000	0x8144FFFF	64K
RS232_DCE	C_BASEADDR	0x84000000	0x8400FFFF	64K
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K
DDR_SDRAM	C_MPMC_BASEA...	0x8C000000	0x8FFFFFFF	64M
lcd_ip_0	C_BASEADDR	0xCF400000	0xCF40FFFF	64K

Figure 22. Generate Addresses

4-2. Modify the system.ucf file to assign external LCD 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 System tab.

4-2-2. Open the *to **W:\A.Bukowiec\zajecia\SW\source\lab3.ucf*** file and copy the pin assignments into the ucf.



```
#----------------------------------------------------------------------------
# IO Pad Location Constraints / Properties for Character LCD GPIO
#----------------------------------------------------------------------------

NET lcd_ip_0_lcd_pin<0> LOC = M18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_E
NET lcd_ip_0_lcd_pin<1> LOC = L18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_RS
NET lcd_ip_0_lcd_pin<2> LOC = L17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_RW
NET lcd_ip_0_lcd_pin<3> LOC = M15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_DB7
NET lcd_ip_0_lcd_pin<4> LOC = P17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_DB6
NET lcd_ip_0_lcd_pin<5> LOC = R16 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_DB5
NET lcd_ip_0_lcd_pin<6> LOC = R15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW; # LCD_DB4
```

Figure 23. Adding UCF Constraints

4-2-3. Save and close the file.

Analyze Assembled ObjectFiles and Verify in Hardware Step 5

5-1. Add a software new software program. Use SDK to generate the configuration file and program the Spartan-3E Starter kit.

5-1-1. Start SDK by clicking **Project → Export Hardware Design to SDK**.

5-1-2. Click on **Export & Launch SDK** button with default settings and browse to **D:\lab3\SDK\SDK_Export**.

5-1-3. I n SDK, on **Project Explorer** tab, right-click on **lab2.c** file **in Testapp → src** and select **delete.**

5-1-4. Right click on **src** select **import**, then chose **General →File System** and add **lab3.c** file from the **source** folder.

5-1-5. Select **TestApp**, right-click and select **Generate Linker Script.**

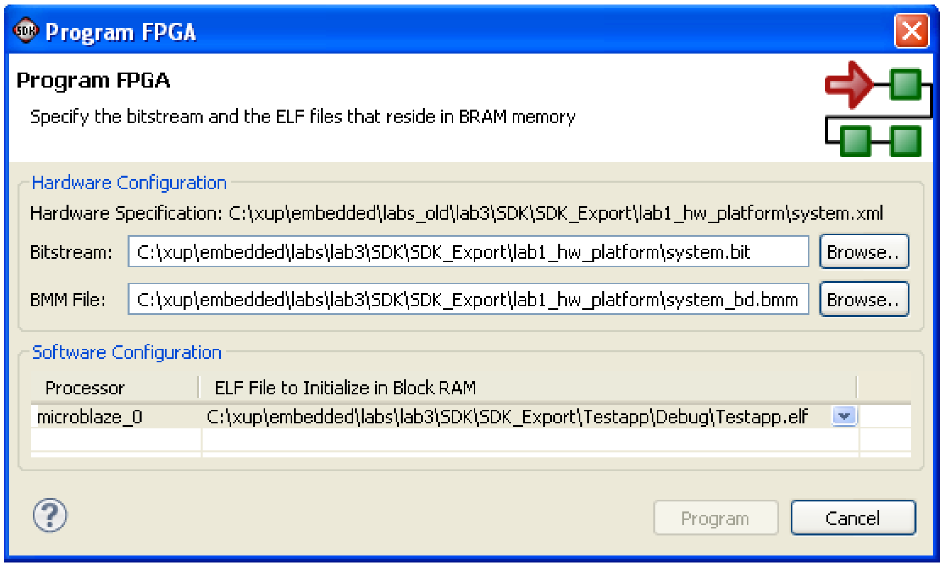
5-1-6. Target everything to **ilmb** and **dlmb** memories, set heap and stack to 400 bytes each, click **Generate** and click **Yes.**

5-1-7. Connect the USB and RS-232 cables to the Spartan-3E Starter kit and power it up.

5-1-8. Start a PuTTY with the following settings:

- Baud rate: 115200
- Data bits: 8
- Parity: none
- Stop bits: 1
- Flow control: none

5-1-9. Select **Xilinx Tools → Program FPGA** in SDK (**Figure 24**).



Hardware Specification: C:\xup\embedded\labs_old\lab3\SDK\SDK_Export\lab1_hw_platform\system.xml

Bitstream: C:\xup\embedded\labs\lab3\SDK\SDK_Export\lab1_hw_platform\system.bit

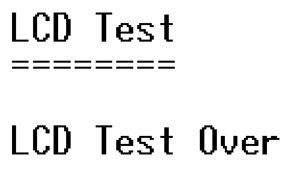
BMM File: C:\xup\embedded\labs\lab3\SDK\SDK_Export\lab1_hw_platform\system_bd.bmm

Processor	ELF File to Initialize in Block RAM
microblaze_0	C:\xup\embedded\labs\lab3\SDK\SDK_Export\Testapp\Debug\Testapp.elf

Figure 24. Program FPGA dialog box

5-1-10. Click on **drop-down** button and select **TestApp.elf** file and click **Program.**

5-1-11. Note: A message displayed PuTTY as shown in **Figure 25.**



```
LCD Test
========

LCD Test Over
```

Figure 25. Screen Shot after the BitStream Downloading

You should see LCD Test Over in the PuTTY window and "Welcome to the #1 Prof Workshop" on the LCD panel on the Spartan-3E Starter kit.

Task 1

Modify the **main** function of the **lab3.c** source file to display DIP Switch a Push Buttons Statuses in decimal format on the LCD panel. The whole message should looks:

```
DIP Sw Status: #XX
Push Btn Status: #YY
```

Conclusion

The Create and Import Peripheral Wizard was used to create peripheral templates for the PLB bus. Logic was added to the templates to create an LCD interface peripheral. The peripheral was then integrated into an existing processor system and tested in hardware using a provided software application to display a message on the on-board LCD.