



Adding Custom IP to an Embedded System Lab

Objectives

This lab demonstrates how to create and add custom IP to an existing PowerPC system using the Xilinx Platform Studio (XPS) Create/Import Peripheral Wizard. The system from the previous lab will be used as the starting point. The lab will show

- How to create a custom PLB IP using the wizard
- How to customize the peripheral
- How to add the new peripheral to the embedded project

Requirements

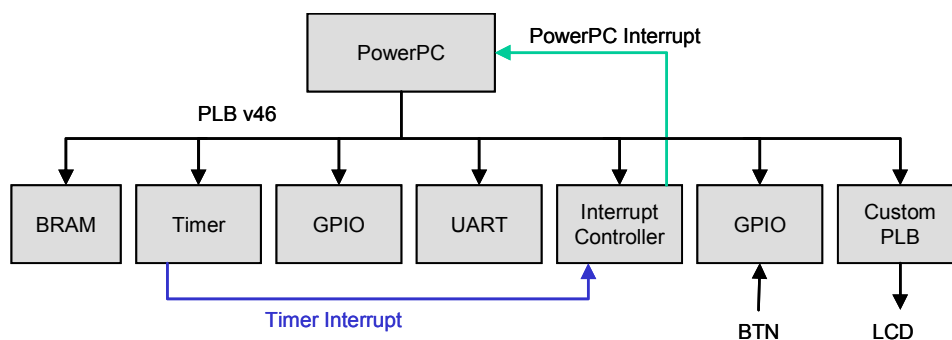
- ISE 9.2.04
- EDK 9.2.02
- ML403 Board
- Platform USB Cable
- RS232 Cable
- Completion of the Adding EDK IP to an Embedded System Lab

Reference

- Platform Studio Help
- Platform Studio SDK Help
- Embedded System Tools Reference Manual
- ML40x User Guide

Overview

The lab is divided into five main steps: creating the custom core using the custom peripheral wizard, customizing the peripheral created, adding the new IP, writing code for the IP, and testing the system on the FPGA. The test application will reside in Block memory inside the FPGA. Below is a block diagram of the hardware platform. We will create the custom IP to use the LCD display on the board.



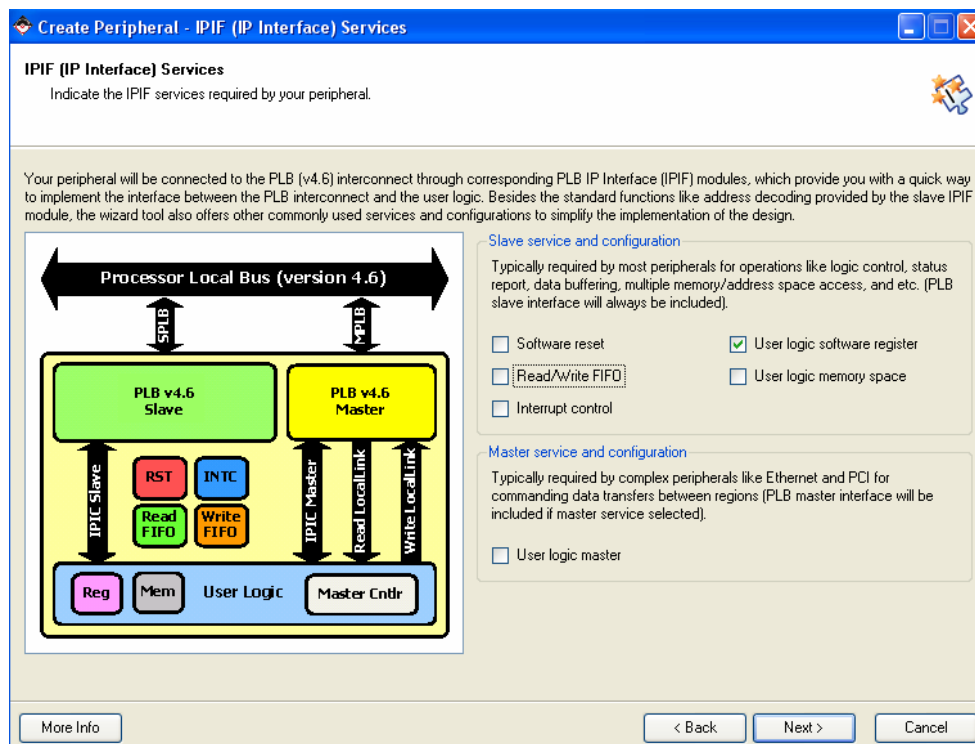
Lab Steps



I. Creating a Custom PLB IP Using the Wizard

We will use the Create/Import Peripheral wizard in XPS to create a new custom IP for the existing system.

1. The ML403 board has a 16-character x 2-line LCD (Lumex LCM-S01602DTR/M) on the board to display text information. The data interface to the LCD is connected to the FPGA to support 4-bit mode only. We will create the new peripheral to handle the hardware interface to the LCD. To simplify the code, only writing to the LCD will be supported.
2. **Open** the ISE project completed in the *Adding EDK IP to an Embedded System Lab*.
3. **In ISE double-click** on *Manage Processor Design* to start XPS.
4. **In XPS**, go to **Hardware > Create or Import Peripheral...** Click on **Next**.
5. Make sure that **Create templates for a new peripheral** is selected then click **Next**.
6. **Select To an XPS project**. Click on **Next**.
7. Enter the name for the new peripheral, **plb_lcd**, and then click **Next**.
8. **Select Processor Local Bus (PLB)**. Click **Next**.
9. The IPIF is a module isolating the user interface to the bus. In addition to facilitating bus attachment, the IPIF provides additional optional services. The services include software registers, user address range, FIFOs, software reset, interrupt support and bus-master access.
10. We will only be using software registers to control the peripheral. **Select User logic software register**. Click **Next**.



11. For non-burst slaves the data width is always 32-bit to save resources. Click **Next**.
12. We will use one 32-bit wide register to communicate with the LCD hardware. Four of the bits will be used for the data and one bit will be used to select instruction versus data information. **Select 1** for the number of registers. Click on **Next**.

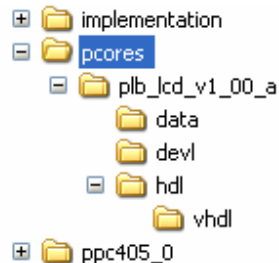


13. The IP Interconnect (IPIC) uses a predefined set of signals between the user logic and the PLB bus. We will use the default signals already selected. Click on **Next**.
14. Bus Functional Models can be generated to accelerate the IP verification. This lab does not cover the BFM simulation of the peripheral at this time. Click on **Next**.
15. The wizard can also generate custom drivers for the peripheral and an ISE project. We will be using XPS and writing our own code to test the peripheral. **Uncheck** both options. Click on **Next**.
16. Click on **Finish** to create the peripheral.

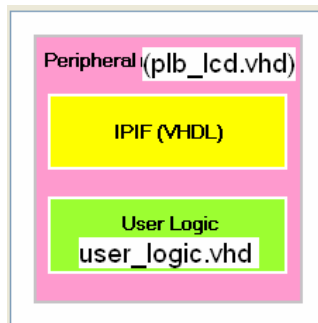
II. Customizing the New IP

The new peripheral was created inside the XPS project directory, in the *pcores* directory. We will first add VHDL code for the LCD functionality then modify the files needed to integrate the peripheral within XPS.

1. **Open** the *pcores* directory. The HDL code resides inside the *hdl > vhdl* directory. The files needed to integrate the IP to XPS are located in the *data* directory.



2. **Open** the VHDL folder. The HDL for *plb_lcd* consists of two files. The *plb_lcd.vhd* file is the top level VHDL which defines the IPIF interface and instantiates the *user_logic* module. The *user_logic.vhd* file contains the user logic.



3. **Open** the *plb_lcd.vhd* file. We will add the external ports for the LCD and map them to the *user_logic* module.
 - **Add** the external ports for the LCD interface to the ports declaration:

```
port
(
  -- ADD USER PORTS BELOW THIS LINE -----
  LCD_E      : out std_logic;
  LCD_RW     : out std_logic;
  LCD_RS     : out std_logic;
  LCD_DB     : out std_logic_vector(3 downto 0);
  -- ADD USER PORTS ABOVE THIS LINE -----
```



- **Add** the ports to the USER_LOGIC_I component instantiation:

```
-- MAP USER PORTS BELOW THIS LINE -----  
LCD_E      => LCD_E,  
LCD_RW     => LCD_RW,  
LCD_RS     => LCD_RS,  
LCD_DB     => LCD_DB,  
-- MAP USER PORTS ABOVE THIS LINE -----
```

- **Save** and **close** the file.

4. The user_logic.vhd file was created for the user to place his custom code. **Open** the user_logic.vhd file. We will add the VHDL to use the LCD. You can cut and paste the code into your own file. Bits 28 through 31 of the software controlled register will be used for the LCD data. Bit 0 will be used to select control versus data information. The HDL code will detect when the register is being written and handle the timing to access the LCD controller. The user_logic file already contained the HDL section to read and write to/from the software controlled register.

- **Add** the ports declaration

```
-- ADD USER PORTS BELOW THIS LINE -----  
LCD_E      : out std_logic;  
LCD_RW     : out std_logic;  
LCD_RS     : out std_logic;  
LCD_DB     : out std_logic_vector(3 downto 0);  
-- ADD USER PORTS ABOVE THIS LINE -----
```

- **Add** user signals declaration

```
--USER signal declarations added here, as needed for user logic  
signal enable_e      : std_logic;  
signal timing_count  : std_logic_vector (4 downto 0);  
signal LCD_E_i       : std_logic;
```

- **Add** the user HDL code before the *end IMP;* line at the end of the file. The display will be written with the contents of the S/W register.

```
-----  
--- LCD Controller  
-----  
  
-- The LCD has a slow interface and a full cycle takes  
-- a minimum of 540ns. A counter is used to control the  
-- LCD hardware timing. The timing starts when a new value  
-- is written to the S/W register  
LCD_TIMING : process (Bus2IP_Clk) is  
begin  
  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then  
    if Bus2IP_Reset = '1' or slv_reg_write_sel(0) = '1' then  
      timing_count <= (others => '0');  
    else  
      timing_count <= timing_count + 1;  
    end if;  
  end if;  
end process LCD_TIMING;  
  
-- Provide an enable for the LCD_E timing  
E_Enable : process (Bus2IP_Clk) is  
begin  
  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then  
    if Bus2IP_Reset = '1' or timing_count = "11010" then  
      enable_e <= '0';  
    elsif slv_reg_write_sel(0) = '1' then  
      enable_e <= '1';  
    end if;  
  end if;  
end process E_Enable;
```



```
        else
            enable_e <= enable_e;
        end if;
    end if;
end process E_Enable;

-- Generate the LCD Enable signal
E_TIMING : process (Bus2IP_Clk) is
begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
        if Bus2IP_Reset = '1' then
            LCD_E_i <= '0';
        elsif timing_count = "00110" and enable_e = '1' then
            LCD_E_i <= '1';
        elsif timing_count = "11001" then
            LCD_E_i <= '0';
        else
            LCD_E_i <= LCD_E_i;
        end if;
    end if;
end process E_TIMING;

LCD_E <= LCD_E_i;

-- Always Write
LCD_RW <= '0';

-- Connect bits 28 to 31 of slv_reg0
LCD_DB <= slv_reg0 (28 to 31);

-- Connect bit 0 of slv_reg0
LCD_RS <= slv_reg0 (0);
```

- **Save** and **Close** the file.

5. The new external ports need to be added to the definition file for the peripheral in order to be used in XPS. **Open** the data directory and open the file *plb_lcd_v2_1_0.mpd*.

- **Add** ports

```
## Ports
PORT LCD_E = "", DIR = O
PORT LCD_RW = "", DIR = O
PORT LCD_RS = "", DIR = O
PORT LCD_DB = "", DIR = O, VEC = [3:0], ENDIAN = LITTLE
```

- **Save** and **close** the file.

6. In XPS, rescan the user IP directories. **Project > Rescan User Repositories**.

7. The new core will now be available.

III. Adding the Custom Peripheral to the System

We will add and connect the new custom IP to the existing system following the same instructions as in the previous lab.

1. In XPS, **click** on the *IP Catalog* tab in the Project Information Area.
2. **Expand** the *Project Local pcores* list to view the custom IP.
3. **Select** *PLB_LCD* then **drag and drop** it to the *System Assembly* window.



4. **Click** on the circle to the left of `plb_lcd_0` to connect it to the PLB bus.
5. **Click** on the *Addresses* tab. **Click** on the *Generate Addresses* button to generate the address range for the new IP.
6. **Click** on the *Ports* tab. **Expand** `plb_lcd_0` from the list. It will show the connections available for the peripheral.
7. In the ports list, for `LCD_E`, `LCD_RW`, `LCD_RS` and `LCD_DB` **click** on the port and select **Make External**.
8. We need to update the constraint file to add the pinout information for the LCD. In **ISE**, select the `ppc_system.ucf` file. **Expand** the User Constraints and **double-click** on *Edit Constraints (Text)*.
9. After Synthesis is complete, **add** the lines:

```
NET plb_lcd_0_LCD_E_pin LOC = AE13 | IOSTANDARD = LVCMOS33;
NET plb_lcd_0_LCD_RS_pin LOC = AC17 | IOSTANDARD = LVCMOS33;
NET plb_lcd_0_LCD_RW_pin LOC = AB17 | IOSTANDARD = LVCMOS33;
NET plb_lcd_0_LCD_DB_pin<3> LOC = AF12 | IOSTANDARD = LVCMOS33;
NET plb_lcd_0_LCD_DB_pin<2> LOC = AE12 | IOSTANDARD = LVCMOS33;
NET plb_lcd_0_LCD_DB_pin<1> LOC = AC10 | IOSTANDARD = LVCMOS33;
NET plb_lcd_0_LCD_DB_pin<0> LOC = AB10 | IOSTANDARD = LVCMOS33;
```

10. **Save** and **Close** the file.
11. In **ISE**, select the `ppc_system` source
12. **Double-click** on *Update Bitstream with Processor Data* to update the bit file with the new peripheral.

IV. Writing Code for the Custom IP

To test the new peripheral we will add code to the `Lab_Test` project created with Platform Studio SDK.

1. The LCD is controlled with a software register. From the address map defined in the `user_logic` code, the register is located at `BaseAddress + 0x0`.
2. In **XPS**, **click** on the **Launch Platform Studio SDK** button on the toolbar.
3. Click on **Cancel** in the *Application Wizard* window.
4. **Expand** the processor instance `ppc405_0_sw_platform/ppc405_0` in the **Navigator** window. **Expand** the *include* directory. **Double-click** on the `xparameters.h` file to view the driver parameters for the custom peripheral:

```
/* Definitions for peripheral PLB_LCD_0 */
#define XPAR_PLB_LCD_0_BASEADDR 0xC9C00000
#define XPAR_PLB_LCD_0_HIGHADDR 0xC9C0FFFF
```

5. **Double-click** on the `main.c` file in the `Lab_Test` project. We will add code before the while statement used for the push button. The LCD datasheet provides the information to initialize and write characters to the LCD (www.lumex.com). Bit 0 of the register will control the `LCD_RS` signal.
6. The hardware only provides the physical interface to the LCD. The LCD controller requires 40µs to 1.64ms delays between commands and 1µs delays between data cycles. The hardware could be modified to handle the delays and check the LCD controller status.
7. After initialization, we will write "Custom IP" on the LCD display.
8. **Add** the sleep functions header file after the last `#include`

```
#include "sleep.h"
```



9. Add a new global pointer after *Xuint32 Push_Read*;

```
Xuint32      *LCD_Data;
```

10. Define the base address for the LCD found in the *xparameters.h* file

- Add the following code before `while(1) {`

```
LCD_Data = (Xuint32 *)XPAR_PLB_LCD_0_BASEADDR;
```

11. Initialize the LCD. Refer to the ML40x User Guide for information on initializing the LCD.

- Add the following code before `while(1){` to initialize the screen

```
// wait
sleep(1);

// Initialize the LCD, RS is set to 0 for instructions
*(LCD_Data) = 0x00000002;      //Function Set
usleep(50);
*(LCD_Data) = 0x00000002;
usleep(1);
*(LCD_Data) = 0x0000000C;
usleep(50);

*(LCD_Data) = 0x00000000;      //Display On
usleep(1);
*(LCD_Data) = 0x0000000F;
usleep(50);

*(LCD_Data) = 0x00000000;      //Clear Display
usleep(1);
*(LCD_Data) = 0x00000001;
sleep(2);

*(LCD_Data) = 0x00000000;      //Entry Mode
usleep(1);
*(LCD_Data) = 0x00000006;
usleep(100);
```

12. Write out the characters. Bit 0 asserted indicates data information.

- Add the following code before `while(1) {`

```
//Write "Custom IP" on the display
*(LCD_Data) = 0x80000004; usleep(1); *(LCD_Data) = 0x80000003;
usleep(60);
*(LCD_Data) = 0x80000007; usleep(1); *(LCD_Data) = 0x80000005;
usleep(60);
*(LCD_Data) = 0x80000007; usleep(1); *(LCD_Data) = 0x80000003;
usleep(60);
*(LCD_Data) = 0x80000007; usleep(1); *(LCD_Data) = 0x80000004;
usleep(60);
*(LCD_Data) = 0x80000006; usleep(1); *(LCD_Data) = 0x8000000F;
usleep(60);
*(LCD_Data) = 0x80000006; usleep(1); *(LCD_Data) = 0x8000000D;
usleep(60);
*(LCD_Data) = 0x8000000A; usleep(1); *(LCD_Data) = 0x80000000;
usleep(60);
*(LCD_Data) = 0x80000004; usleep(1); *(LCD_Data) = 0x80000009;
usleep(60);
*(LCD_Data) = 0x80000005; usleep(1); *(LCD_Data) = 0x80000000;
usleep(1);
```

13. **Save** and **close** the file. Verify that the code compiled without errors.

14. The system is ready to be downloaded to the board.



V. Test the Generated System with the Sample Application

1. Connect a serial cable to the RS232 port and connect the USB platform cable.
2. Power-up the board.
3. Open Hyperterminal and connect at 9600, no parity.
4. Download to the FPGA from SDK: **Device Configuration > Program Hardware.**
5. Verify that the print statement appears on the Hyperterminal window.
6. The LCD should show "Custom IP".
7. Using the Push Button (GPIO_SW_S) should turn on the LEDs while pressed.