



Adding Your Own Peripheral



Objectives

After completing this module, you will be able to:

- Describe the basic PLB bus transactions
- Differentiate between free- and evaluation-based IP delivered in the EDK
- Identify the requirements for integrating your Peripheral
- List the steps involved in creating and importing peripherals using Create/Import IP wizard
- Identify the limitations of creating peripherals with the wizard

Outline



- **PLB Bus and Interfacing**
- XPS Directory Structure
- XPS Peripheral Device Files
- IP Delivery in the EDK
- Create and Import Peripheral Wizard

Overview

- Peripherals are connected to the microprocessor by using the data and address buses
- Xilinx has implemented the IBM CoreConnect bus architecture
- Processor Local Bus (PLB) version 4.6 of the CoreConnect bus architecture is designed for easy connection of on-chip peripheral devices
- Any custom peripheral that connects to the PLB bus must do the following:
 - Meet the principles of the PLB protocol
 - Meet the requirements of the Platform Generator
 - This allows you to take advantage of the simple automated flow that generates system-level architecture

Features

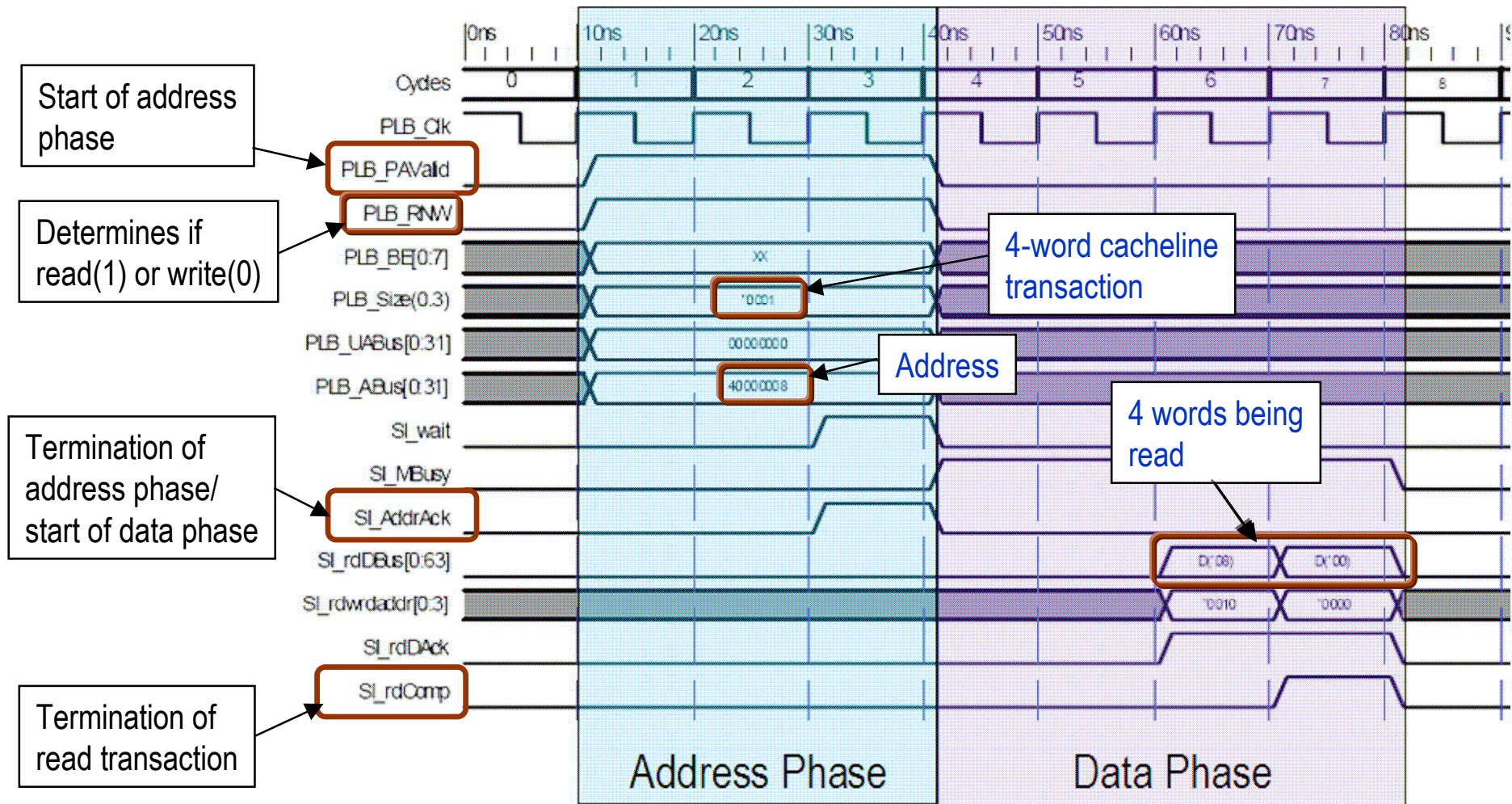
- Platform Generator supports the following features of the PLB v4.6 for PLB peripherals:
 - 32-bit address bus
 - 32, 64, or 128-bit data width
 - Selectable shared bus or point-to-point interconnect topology
 - Point-to-point optimization available for one master, 1 slave configuration
 - Point-to-point topology supports 0 cycle latency via arbitration removal
 - Selectable address pipelining support (2 level only)
 - Watchdog timer for address phase request timeout generation
 - Dynamic master request priority based arbitration
 - Vectored resets and address/qualifier registers

Transactions

- PLB_Size communicates information about transaction data width, type, and length, as shown below
 - “0000”: single data beat, PLB_BE indicates number of bytes
 - “0001”: 4-word cache line
 - “0010”: 8-word cache line
 - “0011”: 16-word cache line
 - “0100”-“0111” and “1110”-“1111”: Reserved
 - “1010”: Word burst transfer
 - “1011”: Double Word burst transfer
 - “1100”: Quad Word burst transfer
 - “1101”: Octal Word burst transfer
- Length of burst specified by PLB_BE

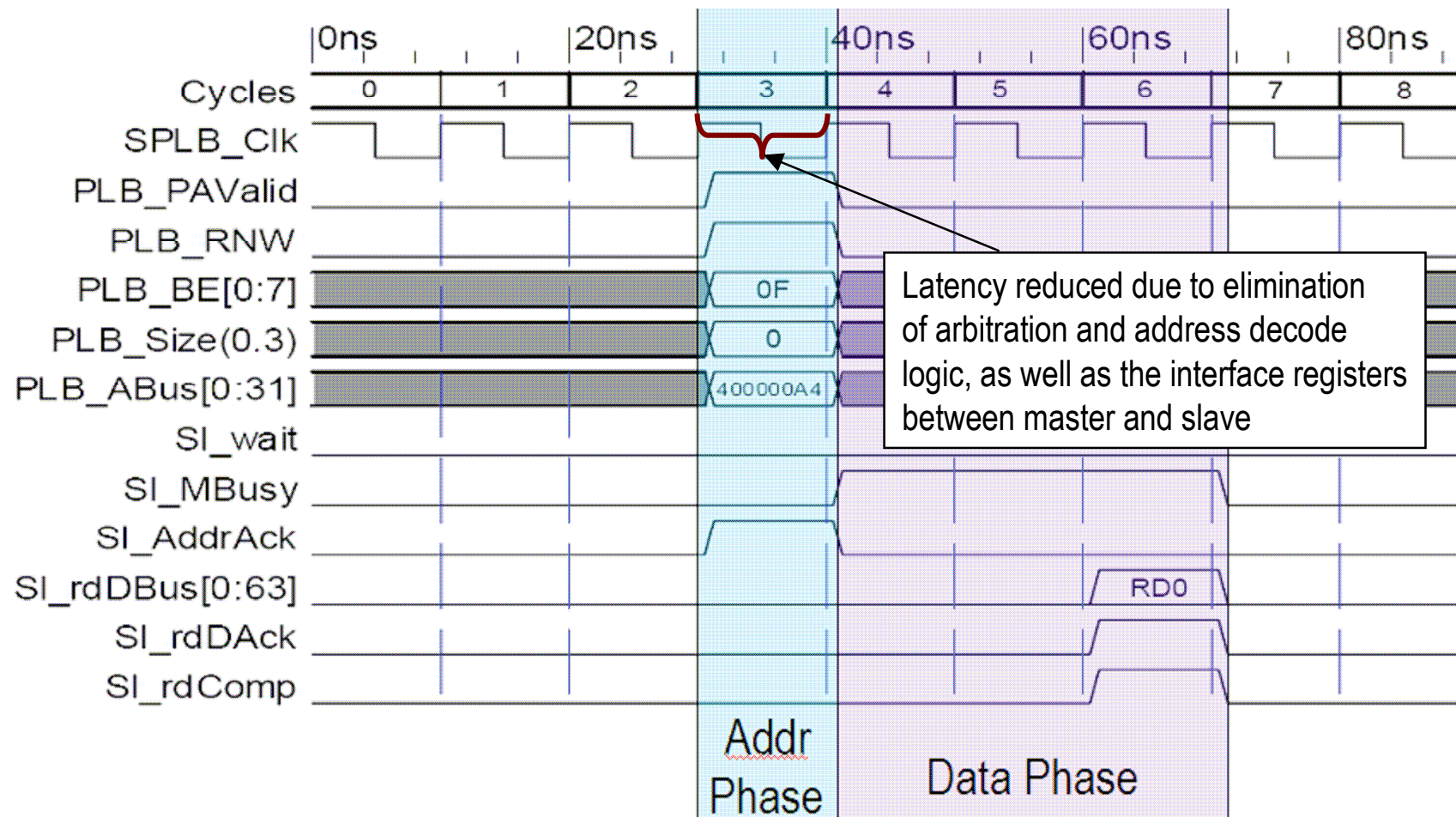
Address and Data Phases

Example of a Read Transaction (refer to EDK docs for detailed signal descriptions)



Point-to-Point Bus Topology

For configurations with single master and single slave (results in reduced address phase)



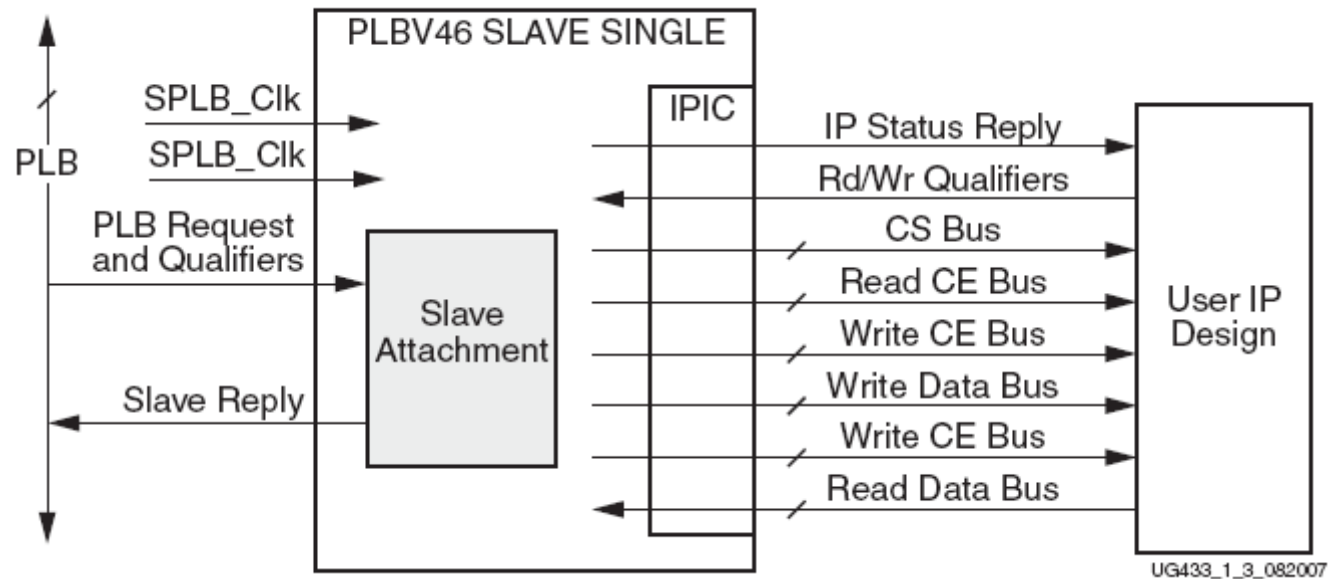
Connecting Devices to the PLB

PLBv46 IPIF templates provided for connecting custom peripherals

- There are four PLBv46 IPIF modules available
 - Slave Single
 - Slave Burst
 - Master Single
 - Master Burst
- Automatically generated by create/import peripheral wizard (will cover this later)

Slave Single

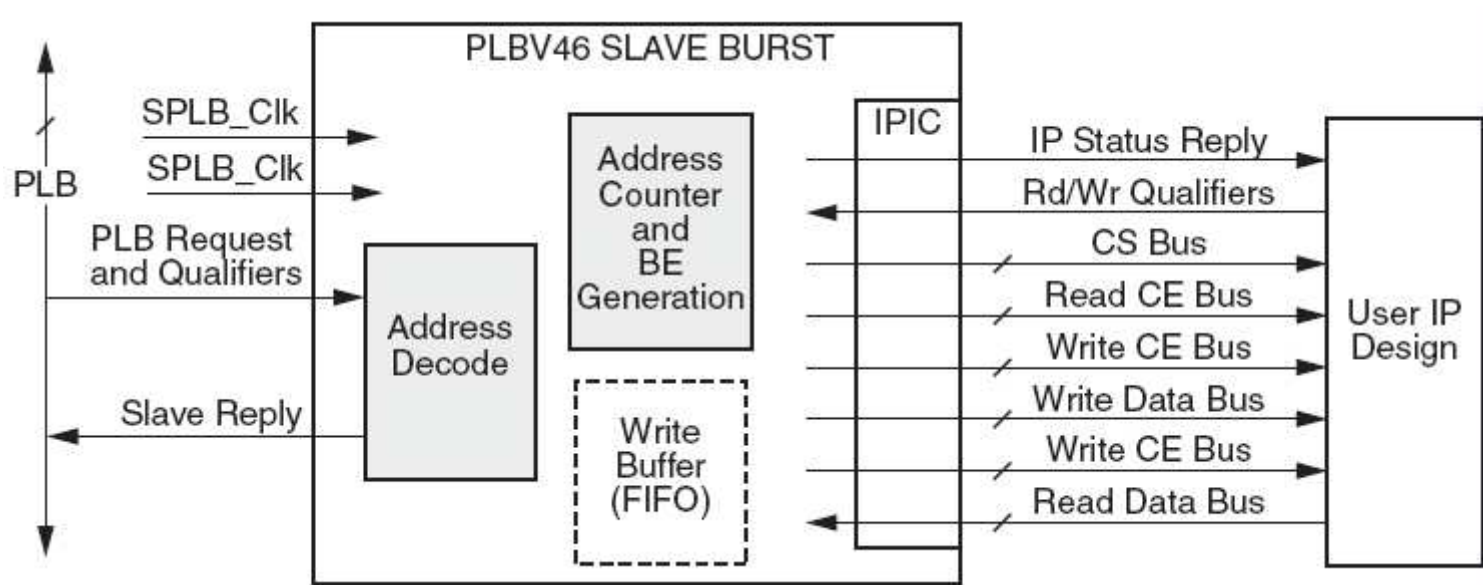
Main target use is for register access in the user IP design



- Supports 32-bit address and 32-bit data bus
- Only single Read and Write data transfers are supported
 - No burst transfers

Slave Burst

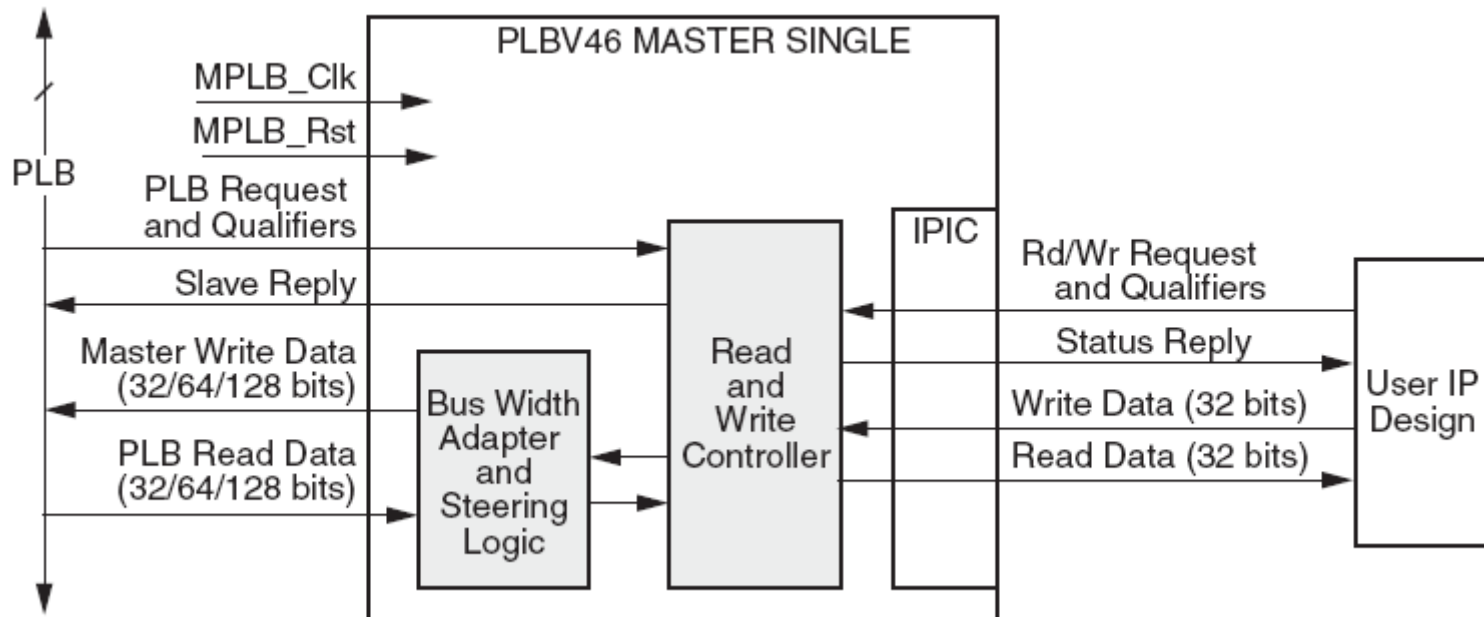
Main target use is for higher throughput slaves such as memory controllers and bridges



- Supports 32-bit address bus and 32-bit, 64-bit, or 128-bit data bus
- Single Read and Write data transfers, fixed length burst transfers (up to 16 data beats), and cacheline transfers are supported

Master Single

Main target use is for register access in the user IP design

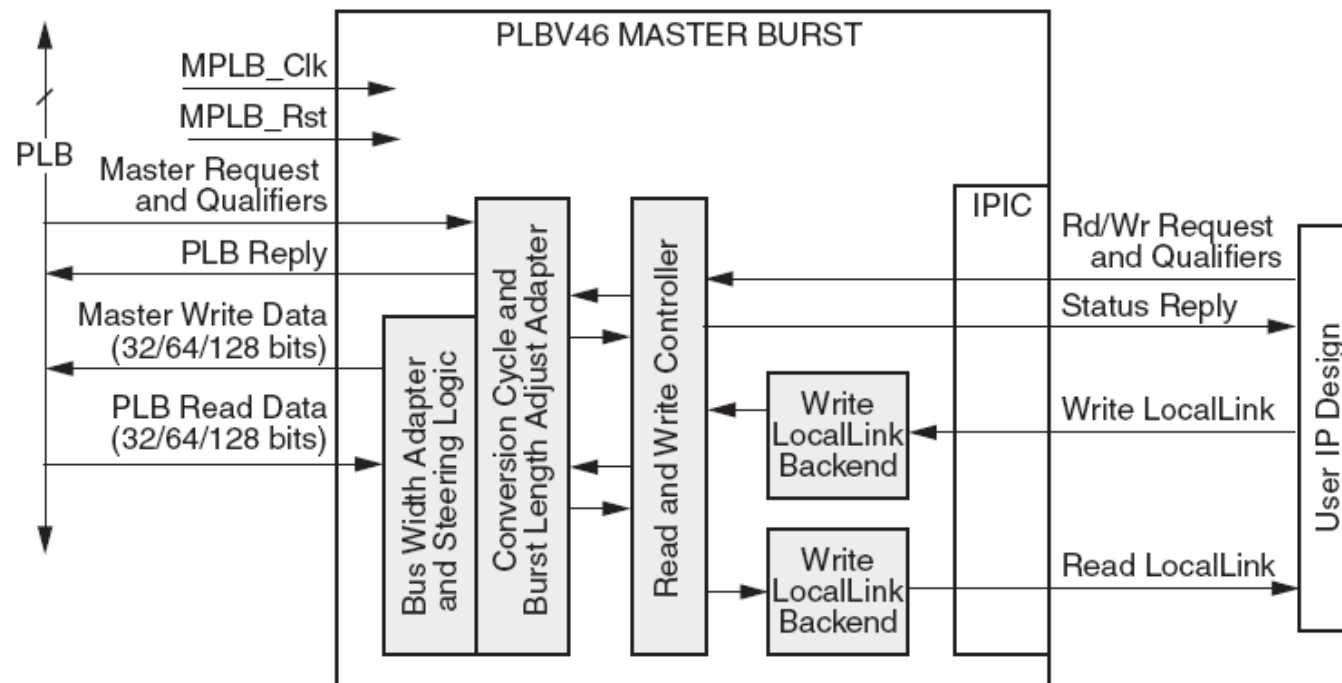


UG493_1_7_082007

- Supports 32-bit address and 32-bit data bus
- Only single Read and Write data transfers are supported
 - No burst transfers
- Uses reduced LocalLink interface to connect to user IP design

Master Burst

Main target use is for high performance, high data throughput master devices



UG433_1_8_052407

- Supports 32-bit address bus and 32-bit, 64-bit, or 128-bit data bus
- Single Read and Write data transfers and fixed length burst transfers (up to 16 data beats) can be initiated to slaves of the same or different sizes
- With Master Burst, the User IP reads and writes from the PLB Master via the Xilinx LocalLink Interface Protocol

Outline



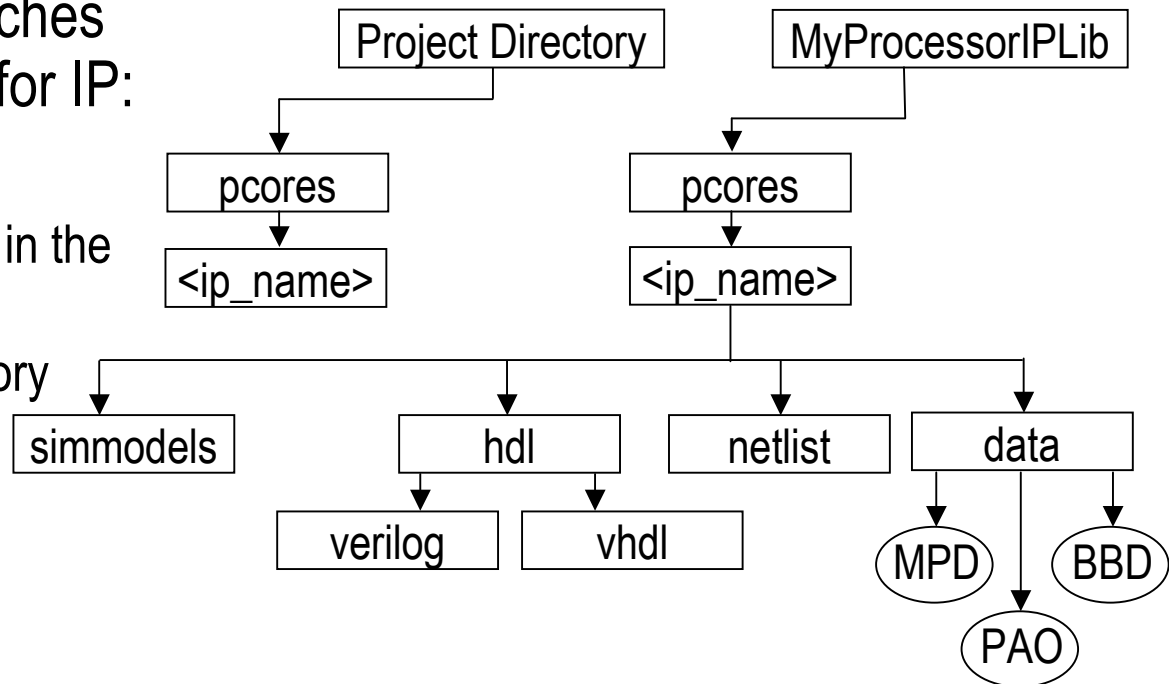
- PLB Bus and Interfacing
- **XPS Directory Structure**
- XPS Peripheral Device Files
- IP Delivery in the EDK
- Create and Import Peripheral Wizard

Peripheral Storage

User peripherals can be located in the project directory or a peripheral repository

- Platform Generator searches the following directories for IP:

- pcores directory (located in the project directory)
- MyProcessorIPLib directory (user defined)



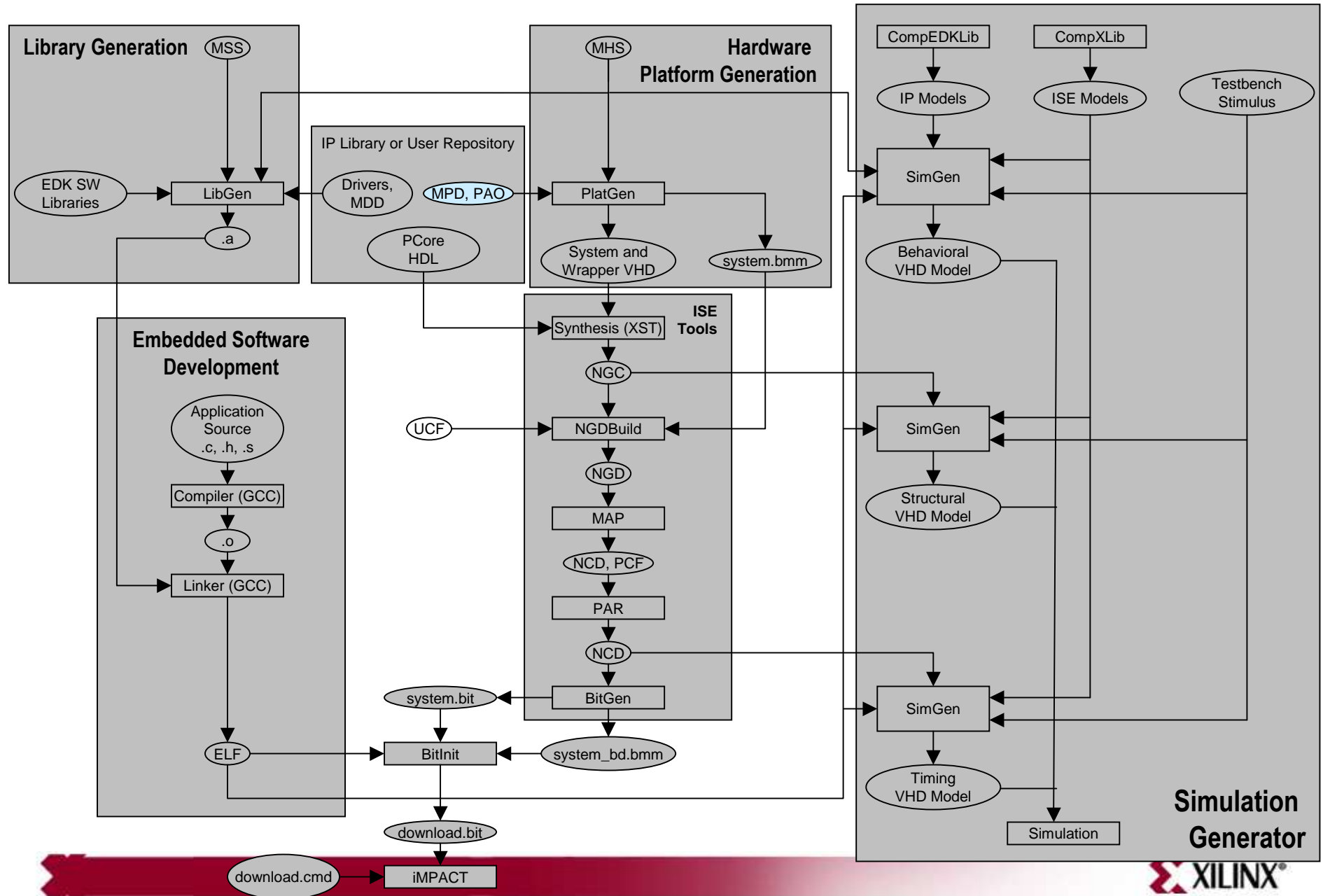
- Repository Directory listed using **Project** → **Project Options** → **Device and Repository** tab
- $\$XILINX_EDK/EDK/hw/XilinxProcessorIPLib/pcores$ (UNIX)
- $\%XILINX_EDK\%EDK\hw\XilinxProcessorIPLib\pcores$ (PC)

Outline

- PLB Bus and Interfacing
- XPS Directory Structure
- **XPS Peripheral Device Files**
- IP Delivery in the EDK
- Create and Import Peripheral Wizard



XPS Peripheral Device Files



XPS Peripheral Device Files

- Microprocessor Peripheral Definition (MPD)
 - Provides default parameters and options for peripheral device in XPS
- Peripheral Analysis Order (PAO)
 - Contains the list of HDL files that are needed for synthesis and defines the analyze order for compilation
- Black-Box Definition (BBD)
 - Manages the file locations of optimized hardware netlists for the black-box sections of your peripheral design

MPD File

Peripheral Options

```
16 BEGIN xps_gpio
17
18 ## Peripheral Options
19 OPTION IPTYPE = PERIPHERAL
20 OPTION IMP_NETLIST = TRUE
21 OPTION HDL = VHDL
22 OPTION LAST_UPDATED = 9.2
23 OPTION USAGE_LEVEL = BASE_USER
24 OPTION DESC = XPS General Purpose IO
25 OPTION LONG_DESC = General Purpose Input/Output (GPIO) core for the PLBV46 bus.
26 OPTION IP_GROUP = General Purpose IO:MICROBLAZE:PPC
27 OPTION ARCH_SUPPORT_MAP = (spartan3=PREFERRED, virtex4lx=PREFERRED, virtex4sx=PREFERRED, v
28
```

- Select various options
 - HDL Language
 - Supported device architectures
 - Supported processors
 - Provide description

MPD File

Bus Interfaces and Parameters

Specify possible bus interfaces

```
31 ## Bus Interfaces
32 BUS_INTERFACE BUS = SPLB, BUS_STD = PLBV46, BUS_TYPE
33
34 ## Generics for VHDL or Parameters for Verilog
35 PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_ve
36 PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_ve
37 PARAMETER C_SPLB_AWIDTH = 32, DT = INTEGER, BUS = SPL
38 PARAMETER C_SPLB_DWIDTH = 32, DT = INTEGER, BUS = SPL
39 PARAMETER C_SPLB_P2P = 0, DT = INTEGER, BUS = SPLB
40 PARAMETER C_SPLB_MID_WIDTH = 1, DT = INTEGER, BUS =
41 PARAMETER C_SPLB_NUM_MASTERS = 1, DT = INTEGER, BUS
42 PARAMETER C_SPLB_NATIVE_DWIDTH = 32, DT = INTEGER, B
43 PARAMETER C_SPLB_SUPPORT_BURSTS = 0, DT = INTEGER, B
44 PARAMETER C_FAMILY = virtex5, DT = STRING
45 PARAMETER C_GPIO_WIDTH = 32, DT = INTEGER, RANGE = (
46 PARAMETER C_ALL_INPUTS = 0, DT = INTEGER, RANGE = (0
47 PARAMETER C_INTERRUPT_PRESENT = 0, DT = INTEGER, RAN
48 PARAMETER C_IS_BIDIR = 1, DT = INTEGER, RANGE = (0,1
49 PARAMETER C_DOUT_DEFAULT = 0x00000000, DT = std_logi
50 PARAMETER C_TRI_DEFAULT = 0xffffffff, DT = std_logic
51 PARAMETER C_IS_DUAL = 0, DT = INTEGER, RANGE = (0,1)
52 PARAMETER C_ALL_INPUTS_2 = 0, DT = INTEGER, RANGE =
53 PARAMETER C_IS_BIDIR_2 = 1, DT = INTEGER, RANGE = (0
54 PARAMETER C_DOUT_DEFAULT_2 = 0x00000000, DT = std_lo
55 PARAMETER C_TRI_DEFAULT_2 = 0xffffffff, DT = std_log
```

Over-rides HDL generics

```
entity xps_gpio is
generic
(
C_BASEADDR      : std_logic_vector(0 to 31) := X"FFFFFFFF";
C_HIGHADDR      : std_logic_vector(0 to 31) := X"00000000";
C_SPLB_AWIDTH   : integer range 32 to 36   := 32;
C_SPLB_DWIDTH   : integer range 32 to 128  := 32;
C_SPLB_P2P      : integer range 0 to 1     := 0;
C_SPLB_MID_WIDTH : integer range 1 to 4     := 1;
C_SPLB_NUM_MASTERS : integer range 1 to 16  := 1;
C_SPLB_NATIVE_DWIDTH : integer range 32 to 128 := 32;
C_SPLB_SUPPORT_BURSTS: integer range 0 to 1  := 0;
C_FAMILY        : string                   := "virtex5";
C_GPIO_WIDTH    : integer                  := 32;
C_ALL_INPUTS    : integer range 0 to 1     := 0;
C_INTERRUPT_PRESENT : integer range 0 to 1  := 0;
C_IS_BIDIR      : integer range 0 to 1     := 1;
C_DOUT_DEFAULT  : std_logic_vector         := X"0000_0000";
C_TRI_DEFAULT   : std_logic_vector         := X"FFFF_FFFF";
C_IS_DUAL       : integer range 0 to 1     := 0;
C_ALL_INPUTS_2  : integer range 0 to 1     := 0;
C_IS_BIDIR_2    : integer range 0 to 1     := 1;
C_DOUT_DEFAULT_2 : std_logic_vector        := X"0000_0000";
C_TRI_DEFAULT_2 : std_logic_vector        := X"FFFF_FFFF"
);
```



MPD File

Lists peripheral signal ports that are accessible in XPS

```
90 PORT Sl_wrBTerm = Sl_wrBTerm, DIR = O, BUS = SPLB
91 PORT Sl_rdDBus = Sl_rdDBus, DIR = O, VEC = [0:(C_SPLB_DWIDTH-1)], BUS = SPLB
92 PORT Sl_rdWdAddr = Sl_rdWdAddr, DIR = O, VEC = [0:3], BUS = SPLB
93 PORT Sl_rdDack = Sl_rdDack, DIR = O, BUS = SPLB
94 PORT Sl_rdComp = Sl_rdComp, DIR = O, BUS = SPLB
95 PORT Sl_rdBTerm = Sl_rdBTerm, DIR = O, BUS = SPLB
96 PORT Sl_MBusy = Sl_MBusy, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
97 PORT Sl_MWrErr = Sl_MWrErr, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
98 PORT Sl_MRdErr = Sl_MRdErr, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
99 PORT Sl_MIRQ = Sl_MIRQ, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
100 PORT IP2INTC Irpt = "", DIR = O, SIGIS = INTERRUPT, SENSITIVITY = LEVEL_HIGH, INTERRUPT_PR
101 PORT GPIO_IO = "", DIR = IO, VEC = [0:(C_GPIO_WIDTH-1)], THREE_STATE = TRUE, TRI_I = GPIO
102 PORT GPIO_IO_I = "", DIR = I, VEC = [0:(C_GPIO_WIDTH-1)]
103 PORT GPIO_IO_O = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)]
104 PORT GPIO_IO_T = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)]
105 PORT GPIO_in = "", DIR = I, VEC = [0:(C_GPIO_WIDTH-1)], PERMIT = BASE_USER, DESC = 'GPIO1
106 PORT GPIO_d_out = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)], PERMIT = BASE_USER, DESC = 'GPI
107 PORT GPIO_t_out = "", DIR = O, VEC = [0:(C_GPIO_WIDTH-1)], PERMIT = BASE_USER, DESC = 'GPI
108 PORT GPIO2_IO = "", DIR = IO, VEC = [0:(C_GPIO_WIDTH-1)], THREE_STATE = TRUE, TRI_I = GPIO
```

Bus Signals

User data and control signals



PAO File

Contains a list of HDL files required for synthesis, and defines the analyze order for compilation

```
#####  
## Filename:      C:/xup/embedded/labs/lab3/pcores/lcd_ip_v1_00_a/data/lcd_ip_v2_1_0.pao  
## Description:   Peripheral Analysis Order  
## Date:         Mon Dec 10 16:46:46 2007 (by Create and Import Peripheral Wizard)  
#####  
  
lib proc_common_v2_00_a proc_common_pkg vhd1  
lib proc_common_v2_00_a ipif_pkg vhd1  
lib proc_common_v2_00_a or_muxcy vhd1  
lib proc_common_v2_00_a or_gate128 vhd1  
lib proc_common_v2_00_a family_support vhd1  
lib proc_common_v2_00_a pselect_f vhd1  
lib proc_common_v2_00_a counter_f vhd1  
lib plbv46_slave_single_v1_00_a plb_address_decoder vhd1  
lib plbv46_slave_single_v1_00_a plb_slave_attachment vhd1  
lib plbv46_slave_single_v1_00_a plbv46_slave_single vhd1  
# add user source files in the order of dependency below  
  
lib lcd_ip_v1_00_a user_logic vhd1  
lib lcd_ip_v1_00_a lcd_ip vhd1
```

Order of dependency



Update this section



BBD File

Manages file locations of optimized hardware netlists for black-box sections of the peripheral design

- The NGC netlists are copied into the project/implementation directory
 - The MPD file should have `OPTION STYLE = MIX` for the tools to copy the files
- Example of a single file without options:
 - FILES
 - Blackbox.ngc
- Example of multiple file selections without options:
 - FILES
 - blackbox1.ngc, blackbox2.ngc, blackbox3.edn

Example of a BBD File with multiple file selections

C_FAMILY	C_RPM	C_FPU_TYPE	FILES
virtex2	true	full	opb_fpu_full.edf
virtex2	true	lite	opb_fpu_lite.edf
virtex2p	true	full	opb_fpu_full.edf
virtex2p	true	lite	opb_fpu_lite.edf
virtex	false	full	opb_fpu_full_noram32x1d.edf, ram32x1ds.edf
virtex	false	lite	opb_fpu_lite_noram32x1d.edf, ram32x1ds.edf

File Usage

- Three ways to integrate your own IP into XPS:
 - As a black box
 - Synthesized with XST or a third-party synthesis tool
 - Requires MPD and BBD files
 - MPD file should have `OPTION STYLE = MIX`
 - As a source
 - Synthesized with the rest of the processor system
 - Uses XST
 - Requires MPD and PAO files
 - Mix
 - Uses netlist and source files
 - Requires MPD, PAO, and BBD files
 - MPD file should have `OPTION STYLE = MIX`

Outline

- PLB Bus and Interfacing
- XPS Directory Structure
- XPS Peripheral Device Files
- **IP Delivery in the EDK**
- Create and Import Peripheral Wizard



IP Cores

See IP Catalog or Xilinx web for Complete Listing of free and evaluation IP Cores

- Xilinx has created a wide variety of IP cores:
 - Bus infrastructure cores
 - Busses: PLB, OPB
 - Bridges: PLB2OPB , OPB2PLB
 - Communication: High-Speed
 - 10/100 Ethernet MAC, CAN controller, HDLC Interface, Flexray, MOST, USB2
 - Communication: Low-Speed
 - Serial Peripheral Interface, IIC Interface, UART 16550, UART lite
 - DMA and Counter
 - Fixed interval timer, watchdog timer, central DMA controller
 - Memory Controllers for
 - Block RAM, DDR/DDR2/SDRAM (multi-port available), SRAM/Flash (multi-port available), Compact Flash
 - General Purpose I/O
 - General Purpose I/O (GPIO)
 - Interprocessor Communication
 - Mailbox, MUTEX



IP Core Information

Data sheet provided for each core (right-click on core in IP catalog to access)

- The size of each core is available in the data sheet
- For example, the `opb_ethernetlite_v1_01_b` data sheet contains the following table:

Table 6: Ethernet Lite MAC Performance and Resource Utilization for Virtex-4 (XC4VLX40-10-ff1148)

Parameter			Device Resources				f _{MAX} (MHz)
C_DUPLEX	C_RX_PING_PONG	C_TX_PING_PONG	Slices	Slice Flip-Flops	BRAMS	LUTs	
1	0	0	367	308	2	528	113
1	0	1	471	310	3	606	115
1	1	0	449	310	3	563	110
1	1	1	439	312	4	610	107
0	1	1	511	371	4	769	107

Processor System Size

- The Processor IP Calculator is an online tool that helps you easily estimate the processor IP core size usage
- www.xilinx.com/ipcenter/processor_central/ppcip/calc.htm
- Try it out!

Step 1: Select the Processor IP core you plan to use in your processor system. Some cores offer min and max values based on no options selected for core or all options selected for core, respectively. Some cores also offer the ability to select multiple instances of the core.

Step 2: At the bottom of the tool, click on "Calculate Total Logic Cell Count" to determine the approximate number of LUTs in your processor system and the target Virtex-II Pro FPGA.



	MIN	MAX
MicroBlaze Soft Processor	<input type="radio"/>	<input type="radio"/>
None	<input type="radio"/>	<input type="radio"/>
Infrastructure IP Cores		
OPB Arbiter & Bus Structure, 2M x 2S	<input type="radio"/> MIN	<input type="radio"/> MAX
OPB Arbiter & Bus Structure, 4M x 4S	<input type="radio"/> MIN	<input type="radio"/> MAX
OPB Arbiter & Bus Structure, 8M x 8S	<input type="radio"/> MIN	<input type="radio"/> MAX
None	<input type="radio"/>	<input type="radio"/>



Outline

- PLB Bus and Interfacing
- XPS Directory Structure
- XPS Peripheral Files
- IP Delivery in the EDK
- **Create and Import Peripheral Wizard**



Create and Import Peripheral Wizard

- The wizard helps you create your own peripheral and then import it into your design
- The wizard generates the necessary core description files into the user-selected directory
- You can start the wizard after creating a new project or opening an existing project in XPS
- The user peripheral can be imported directly through the wizard by skipping the creation option
 - Ensure that the peripheral complies with Xilinx implementation of the IBM CoreConnect bus architecture standard

Starting the IP Wizard

Create and Import Peripheral Wizard - Welcome

Xilinx Embedded Processing Solutions

Welcome to the Create and Import Peripheral Wizard

This wizard will help you create and import a user peripheral for use in processor systems developed using the EDK.

Xilinx Embedded Processing Solutions

XILINX

The Create and Import Peripheral Wizard can be started after creating a project and using **Hardware** → **Create or Import Peripheral ...** or opening an existing project or using **Start** → **Programs** → **Xilinx ISE Design Suite 10.1** → **EDK** → **Accessories** → **Create and Import Peripheral Wizard**



Select the Flow and Directory

1 Select Create Peripheral flow

2 Select the target directory – project directory or user repository

The image shows two screenshots of the 'Create and Import Peripheral' wizard. The first screenshot, labeled '1', shows the 'Peripheral Flow' section. It has a title 'Peripheral Flow' and a subtitle 'Indicate if you want to create a new peripheral or import an existing peripheral.' Below this is a paragraph: 'This tool will help you create templates for a new EDK compliant peripheral, or help you import an existing interface files and directory structures required by EDK will be generated.' There are two radio buttons under 'Select flow': 'Create templates for a new peripheral' (selected) and 'Import existing peripheral'. Below this is a 'Flow description' section with the text: 'This tool will create HDL templates that have the need to implement the body of the peripheral.' There are also 'Options' with a checkbox 'Load an existing .cip settings file (saved)'. A flow diagram on the left shows three steps: 'Create Templates' (highlighted in green), 'Implement/Verify', and 'Import to XPS'. A stick figure is shown thinking. The second screenshot, labeled '2', shows the 'Repository or Project' section. It has a title 'Repository or Project' and a subtitle 'Indicate where you want to store the new peripheral.' Below this is a paragraph: 'A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in a project, the peripheral will be available to the project without importing it.' There are two radio buttons: 'To an EDK user repository (Any directory outside of your EDK installation path)' and 'To an XPS project' (selected). Below these are text boxes for 'Repository:' and 'Project:' with the value 'C:\xup\embedded\labs\lab3'.

The project directory assigned as the target directory will allow the peripheral to be available to the project without importing it. User repository will allow multiple projects to access the same peripheral by importing it in a project

Selecting a Peripheral Name and Bus

3

Provide the peripheral name and version

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 to identify the peripheral in the project.

Name:

Version: 1.00.a

Major revision: Minor revision: Hardware/Software compatibility revision:

Description:

4

Select the bus to which the peripheral will attach

Create Peripheral

Bus Interface

Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

Processor Local Bus (PLB v4.6)

Fast Simplex Link (FSL)

ATTENTION

Refer to the following documents to get a better understanding of how user peripheral interconnect and OPB/PLB v3.4 interconnect) and the FSL interface.

NOTE - Select the bus interface above and the corresponding link(s) will appear in the CoreConnect Specification.

[CoreConnect Specification](#)

Selecting Various Functionalities

5

Select the functionality

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
- Read/Write FIFO
- Interrupt control
- User logic software register
- User logic memory space
- 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

6

Configure the Slave Interface

Slave Interface

Configure the slave interface of your peripheral

The IPIF slave library provides a quick way to implement a slave interface decoding over various ranges as configured by the user and implements the (IP InterConnect - interface between user logic and IPIF).

Slave performance

Slave peripherals support single beat read/write data transfers by default have the burst transfer support turned on - this feature provides higher data rate protocol for PLB Fixed Length Burst operations.

Burst and cache-line support

Data width

The native bit width of the internal data bus may be less than or equal to (and can be 32, 64, or 128-bit for slaves supporting burst). To conserve I/O system that may interact with your peripheral.

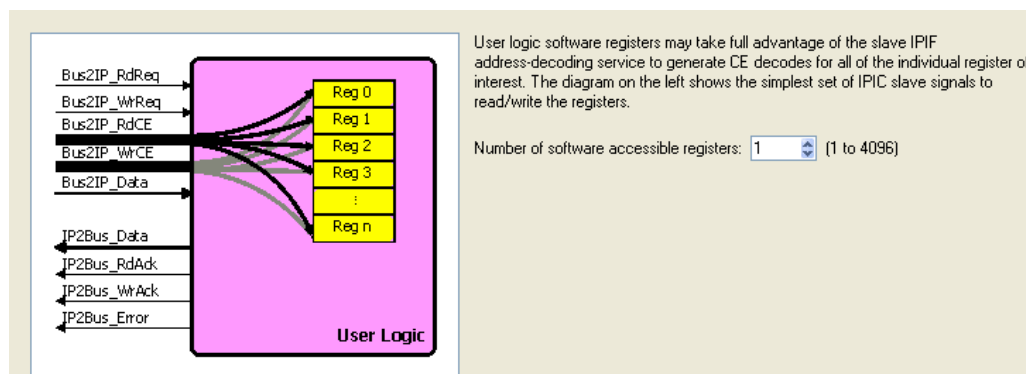
Native data width: 32 bit



Select Software Registers

7

Configure the SW accessible registers



Select IPIF Signals

8

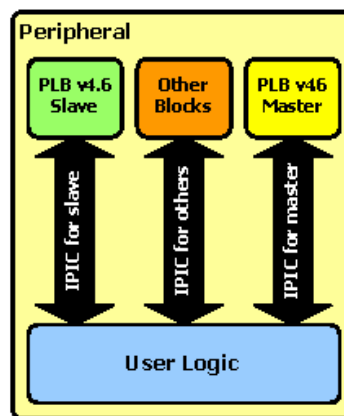
Select the IPIF signals available to the user logic

IP Interconnect (IPIF)

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 interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (II present, some are pre-selected based on the IPIF services you required, and you can choose other options).

Note: all IPIF 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

Select Optional Bus Functional Model

9

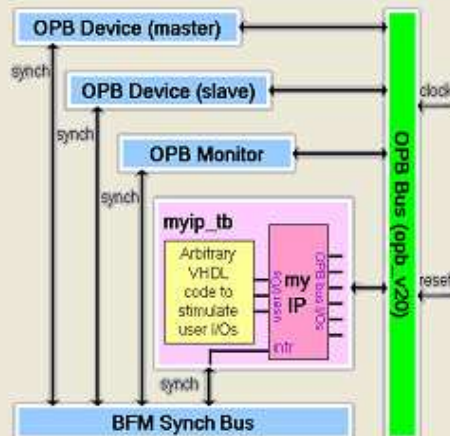
Optional Bus Functional Model Simulation Template

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 toolkit. The link below shows how:

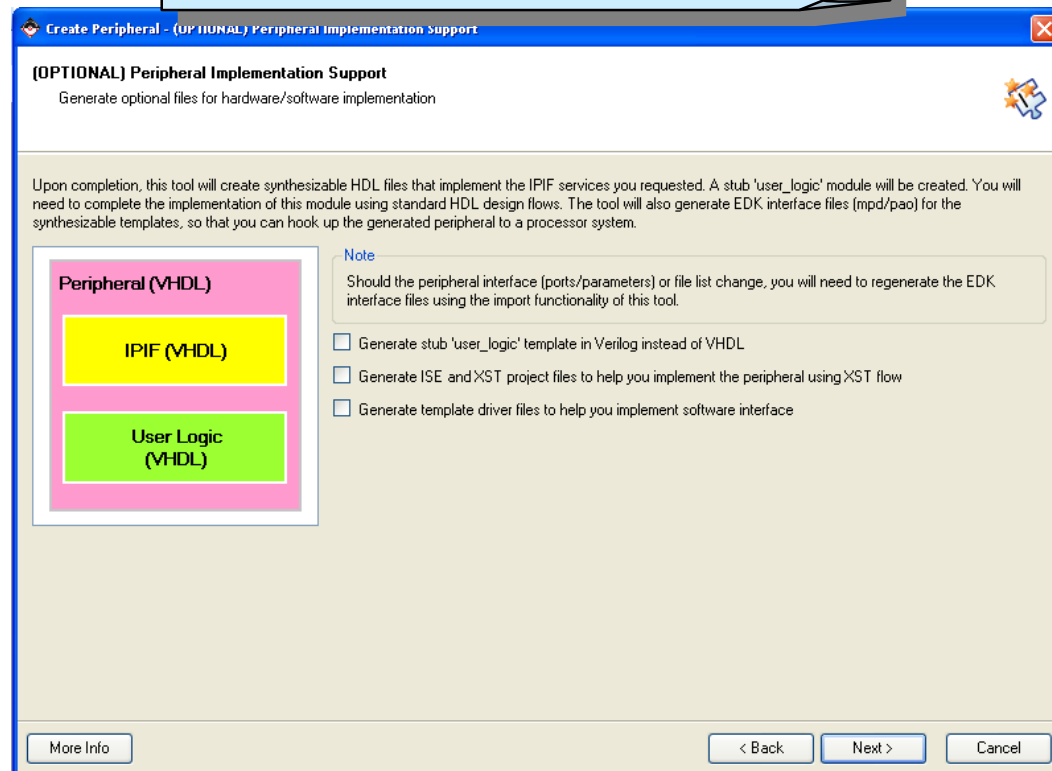
[BFM Toolkit Installation Instructions](#)

Select Optional Implementations Support

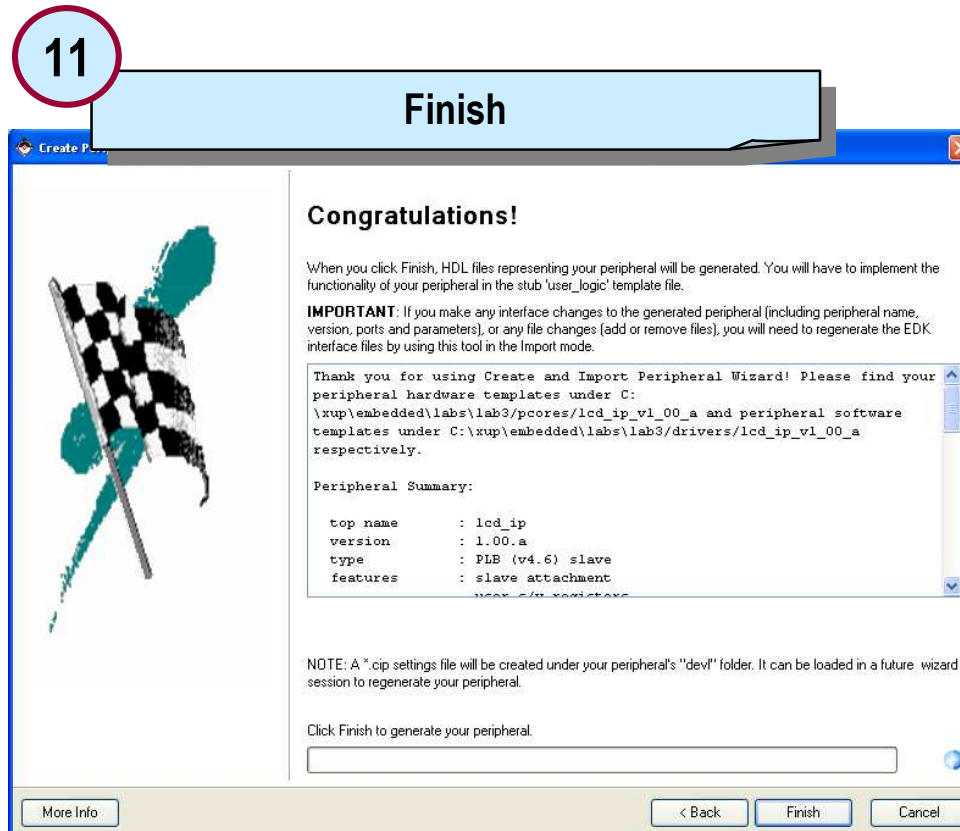
10

Optional Implementation Tools Support

- You can select to generate HDL in Verilog
- You can select to generate project file so you can synthesize using XST and use ISE implementation tools
- You can select to generate software drivers



Generate Peripheral Template

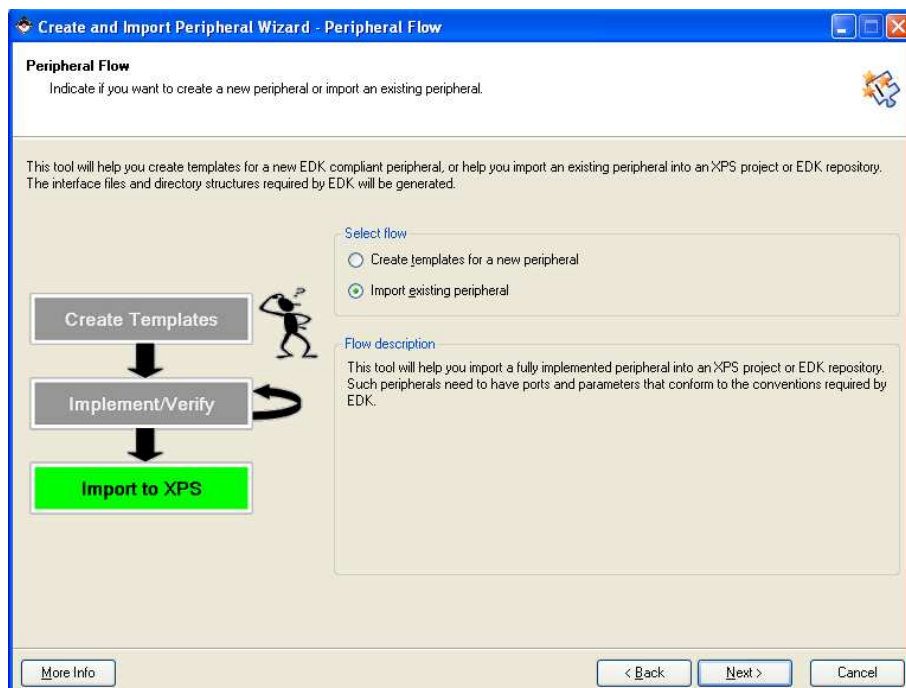


Since the project directory was assigned as the target directory the peripheral will appear in the IP Catalog under **Project Local pcores** folder

Importing a Peripheral

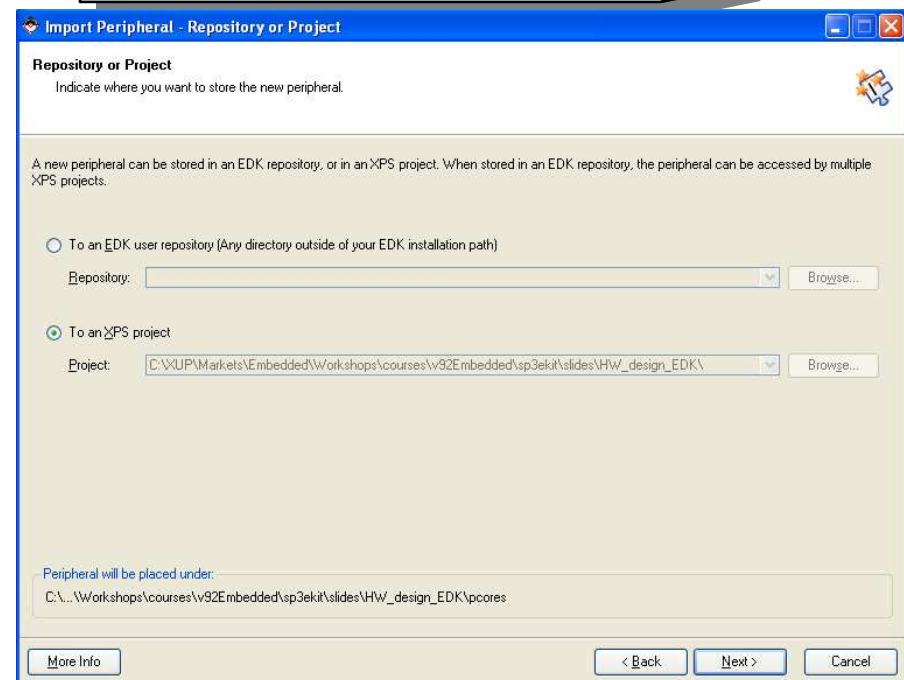
1

Select Import Peripheral flow



2

Identify the Project Directory or Repository



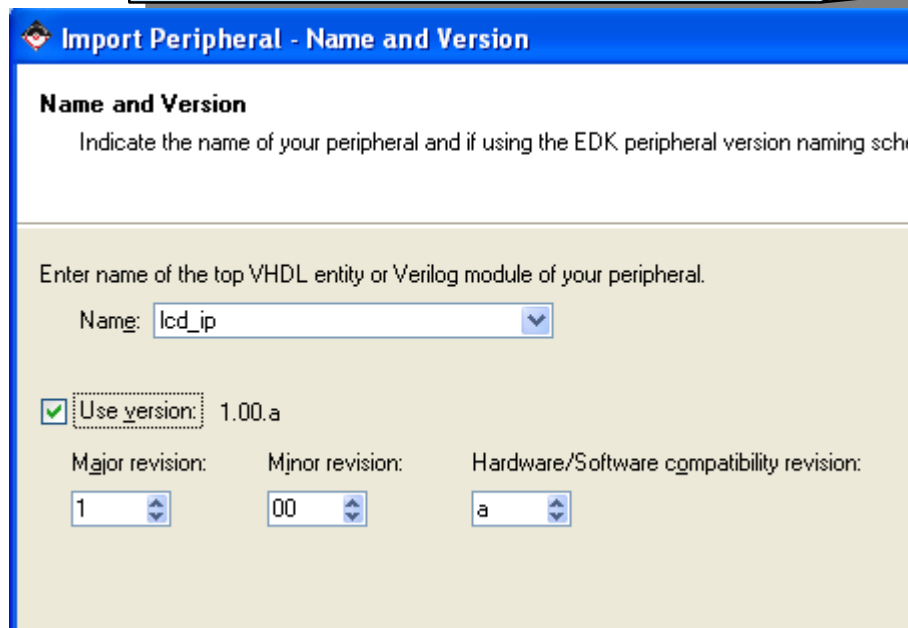
- This step is not needed if the peripheral was created in the current project directory
- Use this step to import peripherals (make a local copy) created in a shared environment



Custom IP Name and Source

3

Enter the top entity name and version name



Import Peripheral - Name and Version

Name and Version
Indicate the name of your peripheral and if using the EDK peripheral version naming scheme

Enter name of the top VHDL entity or Verilog module of your peripheral.

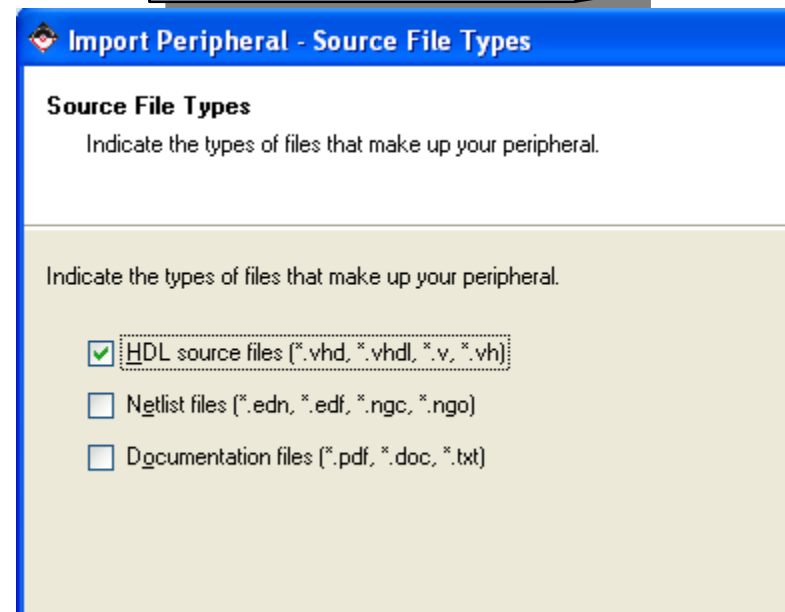
Name: lcd_ip

Use version: 1.00.a

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

4

Select HDL Source files



Import Peripheral - Source File Types

Source File Types
Indicate the types of files that make up your peripheral.

Indicate the types of files that make up your peripheral.

HDL source files (*.vhd, *.vhdl, *.v, *.vh)

Netlist files (*.edn, *.edf, *.ngc, *.ngo)

Documentation files (*.pdf, *.doc, *.txt)

- The name of the peripheral must match the top-level entity name or module name
- The user version name is optional
- Source file types can be a combination of HDL sources, netlists, and documentation files
- The top-level HDL must conform to the CoreConnect bus architecture standard



Custom IP HDL and Location

5

Select language and browse to source files

HDL Source Files
Indicate how this tool

HDL language used to implement your peripheral: VHDL

Use `_data` (*.mpd) collected during a previous invocation of this tool

Browse...

How to locate your HDL source files and dependent library files

Use an XST project file (*.prj)
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.

Browse...

Use existing Peripheral Analysis Order file (*.pao)

Browse...

Browse to your HDL source and dependent library files (*.vhd, *.vhdl, *.v, *.vh) in next step

More Info

< Back Next > Cancel

6

Select HDL source files and libraries

HDL Analysis Information
Indicate the HDL analyze order and the logical libraries your HDL files are compiled into.

Use the buttons on the right to add and remove files, indicate logical libraries and set the HDL analyze order. New sub-HDL libraries will also be imported.

Language	Logical Library	HDL Source File Path
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	proc_common_v2_00_a	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\proc_common_v2_00_a\hdl\vhdl*.vhd
vhdl	plbv46_slave_single_v1_0	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\plbv46_slave_single_v1_0_a\hdl*.vhd
vhdl	plbv46_slave_single_v1_0	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\plbv46_slave_single_v1_0_a\hdl*.vhd
vhdl	plbv46_slave_single_v1_0	C:\Xilinx\EDK92\hw\XilinxProcessorPLib\pcores\plbv46_slave_single_v1_0_a\hdl*.vhd
vhdl	lcd_ip_v1_00_a	C:\XUP\Markets\Embedded\Workshops\courses\92Embedded\sp3ekit\solutions\lat
vhdl	lcd_ip_v1_00_a	C:\XUP\Markets\Embedded\Workshops\courses\92Embedded\sp3ekit\solutions\lat

Add Files...
Add Library...
Remove
Move Up
Move Down

- The HDL language can be VHDL, Verilog, or mixed
- Source files can already be in a project
- Source files can be browsed
- Select the libraries and source files in order of dependency, from lowest to highest



Bus Interface

7

Select bus interface

Import Peripheral - Bus Int

Bus Interfaces
Identify the bus interfaces supported by your peripheral.

A bus interface is a group of related interface ports distinguished by a bus standard (i.e. PLBV46, DCR, or FSL). Select the bus interface(s) supported by your peripheral or indicate if there is no applicable bus interface.

Select bus interface(s)

Processor Local Bus (version 4.6) interface

PLBV46 Master (MPLB)
 Generate burst

PLBV46 Slave (SPLB)

Fast Simplex Link bus interface

FSL Master (MFSL)
 FSL Slave (SFSL)

Device Control Register bus interface

DCR Slave (SDCR)

Note
Xilinx recommends migrating to the new PLB v4.6 bus standard, however, the wizard still supports importing the OPB and PLB v3.4 bus interfaces if you prefer. Please indicate below:

Enable OPB and PLB v3.4 bus interfaces

8

Verify bus port connections

Import Peripheral - Bus Int

SPLB : Port
Define the SPLB bus interface port(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this automatically done the selections for you. Otherwise indicate the ports that correspond to the bus connectors.

Bus Interface Port(s): SPLB

SPLB Bus Connector	Your Port
SPLB_Clk	SPLB_Clk
SPLB_Rst	SPLB_Rst
PLB_abort	PLB_abort
PLB_ABus	PLB_ABus
PLB_UABus	PLB_UABus
PLB_BE	PLB_BE
PLB_busLock	PLB_busLock
PLB_lockErr	PLB_lockErr
PLB_masterID	PLB_masterID

ATTENTION
The Wizard has successfully extracted interface ports for SPLB by applying naming convention.

- If CoreConnect bus architecture naming conventions are followed, the ports are matched; if not, you must assign them

Interrupt Source

9

Define bus interface parameters

Import Peripheral - SPLB : Parameter

SPLB : Parameter
Define the SPLB bus interface parameter(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool automatically done the selections for you. Otherwise check off the values.

Register Space
Parameter determine base address: C_BASEADDR
Parameter determine high address: C_HIGHADDR

Memgry Space

Base Address Parameter	High Address Parameter	Cacheable
------------------------	------------------------	-----------

10

Select interrupt source and sensitivity

Import Peripheral - Identify Interrupt Signals

Identify Interrupt Signals
Identify the interrupt signals on your peripheral.

Indicate the attributes of the interrupt signals by checking the interrupt port name on the left and then clicking on the radio buttons to the right. EDK uses information to automatically connect the interrupt ports of your peripheral.

Select and configure interrupt(s)

lcd

Interrupt sensitivity of port:
 Falling edge sensitive Low level sensitive
 Rising edge sensitive High level sensitive

- Select interrupting signal source, type (level versus edge), and polarity
- This will be presented only if interrupt present

Attributes

11

Select parameter attributes that require special handling

Parameter Attributes
Identify the parameters that require special handling.

Select the parameter on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to system it is instantiated in.

- List User Parameters only -

C_FAMILY

Attributes:

Parameter Name	
Data Type	
Default Value	

12

Select port attributes that require special handling

Port Attributes
Identify the ports that require special handling.

Select the port on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to integrate this periph is instantiated in.

- List User Ports only -

lcd

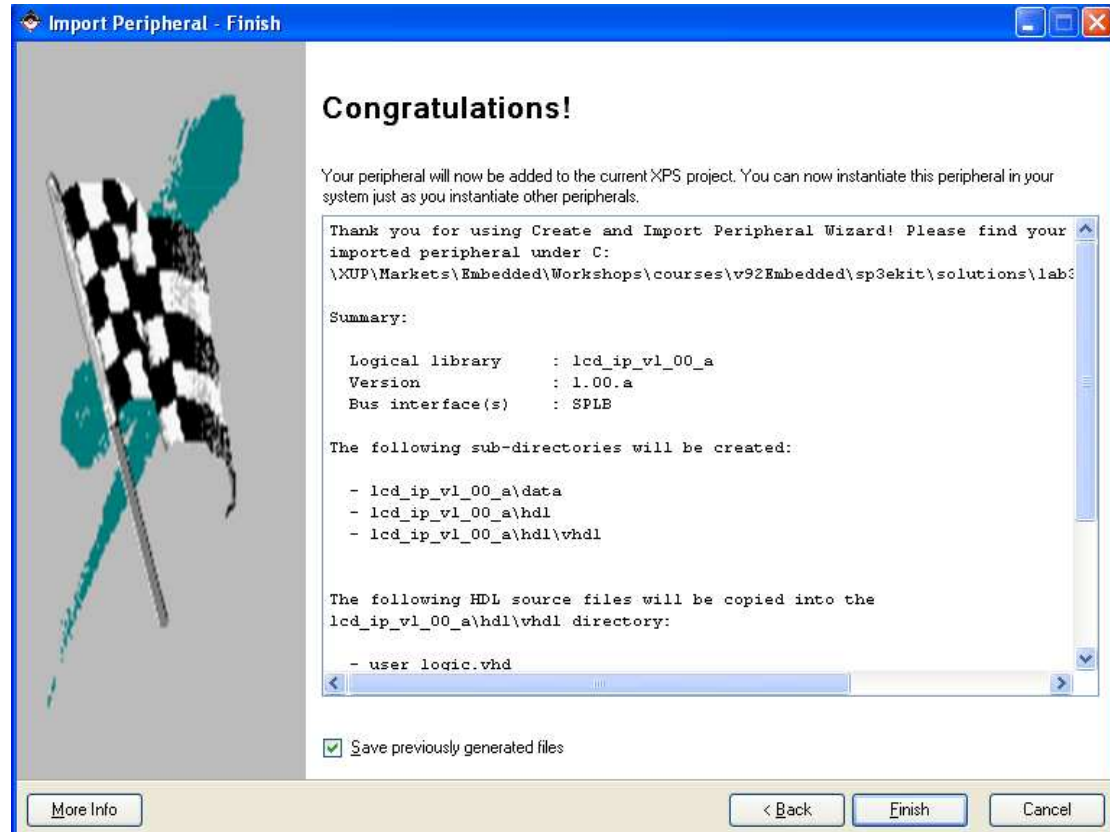
Attributes:

Port Name	lcd
Direction Mode	Output
Default Connection	''
Vector Dimension	[0:6]

- The parameters are listed
- View or change the parameter default values
- The changed value is reflected in the MPD file
- The ports area is listed
- View or change the parameter default values
- The changed value is reflected in the MPD file

Importing Custom IP

- If the netlist and documentation files were selected earlier, the corresponding GUI displays, requesting their locations
- If not, the **Finished** GUI displays
- If the save box is checked, the previously generated files will also be saved



The peripheral will appear under the Peripheral or Project Repository folder in the IP Catalog

Knowledge Check

- What is the process for creating a peripheral of custom IP in XPS?
- What is the process for importing a piece of custom IP into XPS?
- If you are using a third-party synthesis tool to compile your IP, what files are required to integrate that IP into XPS?

Answers

- What is the process for creating a peripheral of custom IP in XPS?
 - Start the **Create and Import Wizard** tool from XPS
 - Select the **Create templates for a new peripheral** option
 - Identify the destination directory location
 - Select the **bus interface**
 - Select **functionality and any interrupts**
 - Define any software registers and address ranges
 - Add additional signals which the peripheral may be using
 - Generate the files
 - Add user logic in user_logic.vhd

Answers

- What is the process for incorporating a piece of custom IP into XPS?
 - Develop your custom IP by using any combination of HDL, netlist, or libraries
 - Ensure that the top-level file conforms to CoreConnect bus architecture requirements
 - Start the Create and Import Wizard tool from XPS
 - Identify the location of libraries, netlists, and source files in order of dependency
 - Select the bus interface
 - Select the source and any type of interrupt
- If you are using a third-party synthesis tool to compile your IP, what files are required to integrate that IP into XPS?
 - MPD and BBD files

Where Can I Learn More?

- Tool documentation
 - *Processor IP Reference Guide*
 - ***Embedded System Tools Guide*** → ***Create/Import Peripheral Wizard***
 - ***Embedded System Tools Guide*** → ***Microprocessor Peripheral Description***
 - ***Embedded System Tools Guide*** → ***Peripheral Analyze Order***
 - ***Xilinx Drivers***
- Support Website
 - EDK Website: www.xilinx.com/edk