PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide

UG443 (v1.2) December 14, 2007





Xilinx is disclosing this Document and Intellectual Property (hereinafter "the Design") to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED "AS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring failsafe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems ("High-Risk Applications"). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2007 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision	
8/29/07	1.0	Initial Xilinx release.	
9/11/07	1.1	Added PPC405-PPC440 System Migration chapter.	
12/14/07	1.2	Modified for EDK 10.1i.	

Preface: About This Guide

Guide Contents	5
Additional Resources	5
Conventions	5
Typographical	5
Online Document	6

Chapter 1: PLB v4.6 Overview

Introduction
Overview of PLB v4.6
Motivation and Advantages7
PLB v4.6 Basics
New Features
Feature Restrictions
Signal and Specification Differences11
Overview of PLBV46 IPIF Modules
PLBV46 Slave IPIF Modules
PLBV46 Master IPIF Modules 17
Master/Slave Mixed Data Width Considerations

Chapter 2: PPC405 - PPC440 System Migration

Introduction				
System Migration Methodology				
System Hardware Migration				
System Software Migration				
Existing PPC405 OPB/PLB v3.4 System				
PowerPC 440/XPS System				
System Migration Inside an EDK System				
Migrating Buses/Processor				
Migrating The Memory Controller				
Migrating Ethernet Solution				
Migrating Interrupt Controller to DCR INTC				
Migrating XPS Peripherals				
Modifying Clocking/ Reset Inside The System				
Migrating software from Virtex-4 PowerPC-405 to Virtex-5 PowerPC-440 64				
Instruction Set Changes				
Boot vector				
Caches 65				
Memory Management 65				
OCM 66				
DCR				
Timers				
APU				
Debugging				

Chapter 3: PPC405 - PPC405/PLB v4.6 System Migration

Chapter 4: MicroBlaze System Migration

Chapter 5: Migration of User IP Slave Cores

Introduction	127
Overview of Create IP Wizard for OPB/PLBV34 Slaves	127
Create IP Wizard Slave Services	127
OPB and PLBV34 IPIFs	127
Overview of Create IP Wizard using PLB v4.6	128
Create IP Wizard Slave Services	128
PLBV46 Slave IPIFs	128
Overview of OPB/PLBV34 Slave Cores Created by Create IP Wizard	128
Creating PLBV46 Slave Cores with Create IP Wizard	128
Generating The PLBV46 Slave Cores Using Create IP Wizard	128
Parameters for PLBV46 Slave Cores	145
Modifying the Existing User Logic	146
Modifying the Existing User Logic	146 146

Chapter 6: Migration of User IP Master/Slave Cores

Introduction							149
--------------	--	--	--	--	--	--	-----

Overview of Create IP Wizard for OPB/PLBV34 Master/Slaves	149
Create IP Wizard Master Services OPB/PLBV34 IPIFs	149 149
Overview of Create IP Wizard for PLB v4.6 Master/Slaves	150
Create IP Wizard Master Services	150
PLBV46 Master/Slave IPIFs	150
Overview of OPB/PLBV34 Master/Slave Cores Created By Create IP Wizard	150
Creating the PLBV46 Master/Slave Cores Using Create IP Wizard	151
Generating the PLBV46 Master/Slave Cores	151
Parameters for PLBV46 Master/Slave Cores	165



Preface

About This Guide

Guide Contents

This manual contains the following chapters:

Chapter 1, "PLB v4.6 Overview"

Chapter 2, "PPC405 - PPC440 System Migration,"

Chapter 3, "PPC405 - PPC405/PLB v4.6 System Migration"

Chapter 4, "MicroBlaze System Migration"

Chapter 5, "Migration of User IP Slave Cores"

Chapter 6, "Migration of User IP Master/Slave Cores"

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/literature.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example	
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100	
Courier bold	Literal commands that you enter in a syntactical statement	ngdbuild design_name	



Convention	Meaning or Use	Example	
Helvetica bold	Commands that you select from a menu	$File \to Open$	
	Keyboard shortcuts	Ctrl+C	
	Variables in a syntax statement for which you must supply values	ngdbuild design_name	
Italic font	References to other manuals	See the <i>Development System</i> <i>Reference Guide</i> for more information.	
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.	
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.	ngdbuild [option_name] design_name	
Braces { }	A list of items from which you must choose one or more	lowpwr ={on off}	
Vertical bar	Separates items in a list of choices	lowpwr ={on off}	
Vertical ellipsis • •	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN'	
Horizontal ellipsis	Repetitive material that has been omitted	allow block block_name loc1 loc2 locn;	

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Plue text	Cross-reference link to a location	See the section "Additional Resources" for details.
blue text	in the current document	Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the Virtex-II Platform FPGA User Guide.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.



Chapter 1

PLB v4.6 Overview

Introduction

This chapter provides an overview of the Xilinx PLB v4.6 interconnect as it applies to the migration of User IP cores and systems. The full specification of the Xilinx PLB v4.6 interconnect is not provided in this document, but the differences from existing OPB/PLB v3.4 interconnects are highlighted and an overview of the PLBV46 IP Interface (IPIF)s is provided. It is assumed that the reader has working knowledge of the existing Xilinx implementations of OPB/PLB v3.4 interconnects.

Overview of PLB v4.6

This section will provide an overview of the Xilinx PLB v4.6 interconnect, but will not provide the detailed specification. Please refer to SP026 PLBV46 Interface Simplifications for more details.

Motivation and Advantages

The prime motivation to develop another interconnect for embedded systems and migrate all Processor IP Cores and systems to this interconnect scheme is to increase system performance. Years of experience developing the existing Xilinx embedded solution provided insight into the features and modifications necessary to provide an interconnect solution with improved performance while still preserving the development infrastructure.

Another motivation was improving the ease of use when developing Embedded Systems. Migrating the Xilinx Processor IP Catalog to PLB v4.6/XPS reduces the number of different cores available that provide the same basic function and allows the same IP and system architecture to be used across processors. This greatly simplifies the system design and implementation task because the user doesn't have to determine which cores can be in which configuration and with the specific processor of choice.

PLB v4.6 Basics

Address and Data Buses

PLB v4.6 supports a 32-bit address bus. The read and write data busses are split and can either be 32-bit, 64-bit, or 128-bit data width. Though the arbitration logic supports a 2-deep address pipeline, this is currently not implemented in the Xilinx PLB v4.6/XPS soft IP slaves.



Address and Data Phases

PLB v4.6 transactions are split into an address phase and a data phase. The address phase is when the requested address and transaction qualifiers are driven to all slaves. It starts with the assertion of PLB_PAvalid and terminates with the assertion of Sl_AddrAck. Address Phases may overlap (in time).



Figure 1-1: PLBV46 Address Phase and Data Phase

The data phase is when the actual data transfer occurs. A read data phase and a write data phase can occur simultaneously. The data phase starts with the assertion of Sl_AddrAck and ends with the assertion of Sl_wrComp or Sl_rdComp. A read data phase and write data phase may overlap.

Transactions

PLB v4.6 supports single data beat, cacheline, and fixed length burst of 2 to 16 data beats. Word (32-bit), double-word (64-bit), and quad-word (128-bit) data widths are allowed for burst transactions. Fixed Length Burst requests with data beat counts from 17 to 256 and transfer widths of bytes or half words will be ignored by Xilinx soft IP Slaves. Cacheline transactions are 4-word, 8-word, or 16-word transfers. Target-word first support requires circular address generation in the Slave.

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 Cacheline
- "0010" = 8-word Cacheline
- "0011" = 16-word Cacheline
- ♦ "0100" "0111", "1110" "1111" = Reserved
- "1010" = Word Burst Transfer
- "1011" = Double Word Burst Transfer

- "1100" = Quad Word Burst Transfer
- "1101" = Octal Word Burst Transfer

When PLB_Size is indicating a Burst Transfer, the length of the burst is specified by PLB_BE. Note that a zero value on PLB_BE when specifying the burst length specifies an indeterminate length burst which is not supported in the Xilinx PLB v4.6 interconnect.

Timeouts

The PLB v4.6 implements an address phase timeout of 16 clocks. The timeout counter (located in the arbiter) can be suspended by assertion of Sl_Wait by the targeted slave. PLB v4.6 data phases have no timeout limit, however, the PLBV46 IPIFs implement a timeout counter from 0 to 511 clocks.

Aborts

Transactions can not be aborted in the Xilinx PLB v4.6 interconnect.

New Features

The Xilinx PLB v4.6 interconnect optimizes the existing PLB v3.4 interconnect by removing unneeded features to reduce resource utilization and lead to higher Fmax.

Point-to-Point Bus Topology

Xilinx PLB v4.6 interconnect supports a Point-to-Point bus topology for configurations that contain a single master and single slave. This feature eliminates the need for the slave to perform address decode, therefore, the resources required by the slave are reduced. Also, the latency of the transaction is reduced since the address phase simplifies to a single clock due to the elimination of arbitration and address decode. Latency is further reduced due to the elimination of interface registers in the master and slave. Since the master and slave outputs only connect to a single load, the need for interface registers is removed.



Figure 1-2: Reduced Address Phase when using Point-2-Point Mode



Feature Subsets

Modes were added to streamline IP to use only the features needed by that IP. Two feature subsets of the IBM 128-bit Processor Local Bus Architectural Specification (v4.6) are implemented: the Baseline feature subset and the Performance feature subset.

Note: The PLB v.4.6 Interface signal set is the same for peripherals which implement either of the feature subsets.

Baseline Feature Subset

The Baseline feature subset is mainly targeted at PLB v4.6 slaves that contain low-speed data and/or control registers. The peripherals that would use the Baseline feature subset are always 32-bit devices which never support burst transactions. Data steering is never required. In summary, the Baseline feature subset provides the following:

- Single unit transfers (no bursts or cache-line)
- 32-bit address (4 GB space)
- 32 bit wide data transfers
- Interface is required to connect to 32, 64 or 128-bit wide data buses (per IBM 128bit Processor Local Bus Architectural Specification (v4.6))

Performance Feature Subset

The Performance feature subset was developed to support high-bandwidth masters and slaves. There is no restriction on the data width of the performance peripherals. They can be 32, 64, or 128 bits wide. However, data mirroring (masters) or data steering (slaves) logic may be required when communicating with narrower devices. Bursts transactions up to 16 beats are supported. Master conversion cycles are supported and allow burst transactions to narrower slaves. In summary, the Performance feature subset provides the following:

- Fixed-length bursts up to 16 data beats
- Cache-line transfers
- 32, 64 or 128 bit wide data transfers
- Connects to same or larger width buses
- Master data mirroring for writes to narrower slaves
- Slave data steering for reads to narrower masters

Feature Restrictions

To optimize the resource utilization and performance of Xilinx PLB v4.6 peripherals, unused and unneeded features of the full IBM 128-bit Processor Local Bus Architectural Specification (v4.6) have been eliminated and restrictions placed on the implementation of some features.

The following features of the IBM 128-bit Processor Local Bus Architectural Specification (v4.6) are not supported:

- Requests for bursts longer than 16 data-beats
- Indeterminate length burst transfers
- Aborts (suppressed in PPC processor)

The following feature restrictions are in place:

- Master burst requests to non-bursting (baseline) slaves are prohibited
- Fixed-length bursts cannot be prematurely terminated by a master or a slave
- Only memory transfer types (M_type[0:2]="000") are supported

Note: The above lists are not all-inclusive. Please refer to SP026 PLBV46 Interface Simplifications for complete details.

Signal and Specification Differences

The Xilinx PLB v4.6 interconnect signal set is basically the same as the existing Xilinx PLB v3.4 interconnect signal set. The basic protocols and transactions are also the same as the existing Xilinx PLB v3.4 basic protocols and transactions. Therefore, for users familiar with the existing Xilinx PLB v3.4 interconnect, there is little or no learning curve.

This section will highlight the major differences in the implemented signals and transactions from the IBM 128-bit Processor Local Bus Architectural Specification (v4.6).

Note: This section will not exhaustively list all differences and will not be all-inclusive. Please refer to SP026 PLBV46 Interface Simplifications for complete details.

Upper Address Bus and Parity Signals

The IBM 128-bit Processor Local Bus Architectural Specification (v4.6) specification added an upper address bus and optional parity and parity enable signals for the address busses, byte enable busses, and data busses but are not supported in Xilinx soft IP:

- Mn_UABus(0:31), PLB_UABus(0:31)
- Mn_ABusPar, Mn_ABusParEn, Mn_UABusPar, Mn_UABusParEn
- PLB_ABusPar, PLB_ABusParEn, PLB_UABusPar, PLB_UABusParEn
- Mn_BEPar, Mn_BEParEn, PLB_BEPar, PLB_BEParEn
- Mn_wrDBusPar, Mn_wrDBusParEn, PLB_wrDBusPar, PLB_wrDBusParEn
- PLB_MnRdDBusPar, PLB_MnRdDBusParEn, Sl_rdDBusParEn

In addition, parity error signals were also added:

• Sl_ABusParErr, Mn_rdDBusParErr

Note: These signals are not supported by Xilinx soft IP.

Interrupt Indicators

A set of signals to allow interrupts to be signalled between masters and slaves is specified in the IBM 128-bit Processor Local Bus Architectural Specification (v4.6) but are not supported in Xilinx soft IP:

PLB_MIRQ(n), Sl_MIRQ(0:n)

Note: These signals are not supported by Xilinx soft IP.



Master Bus Timeout

A timeout indicator is now available for each master on the PLB v4.6 interconnect. This allows the master to distinguish between when a transaction has timed out and a transaction that completes with an error. The master timeout signal is sent from the arbitration logic:

PLB_MnTimeout

Pending Request Signals

The Pending Request signals are now split into pending requests for read and write transactions but are ignored by Xilinx soft IP:

- PLB_pendReq => PLB_rdPendReq, PLB_wrPendReq
- PLB_pendPri(0:1) => PLB_rdPendPri(0:1), PLB_wrPendPri(0:1)

Note: These signals are ignored by Xilinx soft IP

Error Indicators

Error indicators are split into separate read and write signals in PLB v4.6 and are supported by Xilinx soft IP:

Sl_MErr(0:n) => Sl_MRdErr(0:n), Sl_MWrErr(0:n)

PLB_Mn_Err => PLB_MRdErr(n), PLB_MWrErr(n)

Compressed, Guarded, and Ordered Signals

The compressed, guarded, and ordered signals have been removed. This information is now conveyed via the Transfer Attribute bus, but are not supported by Xilinx soft IP:

- Mn_compress, Mn_guarded, Mn_ordered => Mn_TAttribute(0:15)
- PLB_compress, PLB_guarded, PLB_ordered => PLB_TAttribute

Note: These signals are not supported by Xilinx soft IP.

Xilinx Soft IP Support Summary

The Xilinx implementation of the PLB v4.6 interconnect in soft IP doesn't include support for all of the signals specified in the specification. Here's a summary of the support limitations:

- Optional parity signals not supported
- Only memory transfers supported
- M_type, PLB_type = "000"
- Transfer attributes not supported
- Pending request input status signals are present on the Master/Slave interfaces, but are ignored
- Slave to Master interrupts are not supported

Overview of PLBV46 IPIF Modules

This section will provide an overview of the PLBV46 IPIF modules. For more detailed information, the user is encouraged to reference the IP Data sheet for the PLBV46 IPIF of interest.

There are four PLBV46 IPIF modules available:

- ◆ PLBV46 Slave Single
- PLBV46 Slave Burst
- PLBV46 Master Single
- ◆ PLBV46 Master Burst

These four PLBV46 IPIFs provide a slave and master version of the PLB v4.6 baseline and the PLB v4.6 performance feature subsets. Note that the master and slave functions are now completely separated, unlike the existing OPB and PLBV34 IPIF modules. All of the PLBV46 master and slave IPIF modules provide support for point-to-point configurations.

The services provided by the existing OPB and PLBV34 IPIF modules are no longer included in any of the PLBV46 IPIF modules. The services are available as separate libraries and are still easily accessible for inclusion in User IP cores through the Create / Import IP Wizard.

PLBV46 Slave IPIF Modules

PLBV46 Slave Single IPIF Module

The PLBV46 Slave Single IPIF module implements the PLB v4.6 baseline feature subset for a slave peripheral. Its main target use is for register accesses within user IP. Figure 1-3 shows the block diagram for the PLBV46 Slave Single IPIF module.



Figure 1-3: PLBV46 Slave Single Block Diagram

The PLBV46 Slave Single IPIF module supports a 32-bit address bus and a 32-bit data bus. Note that 64-bit and 128-bit data widths are not supported, therefore, a core utilizing this IPIF can only have a data-width of 32-bit. However, cores utilizing this IPIF can connect to PLBV46 busses of 64-bits and 128-bits as required by the Xilinx PLB v4.6 specification.

Only single read and write data transfers are supported by the PLBV46 Slave Single IPIF module. There is no support for burst transfers with this IPIF. If a burst transaction is

addressed to a slave utilizing this IPIF, the address acknowledge will never be asserted causing a bus timeout.

The PLBV46 Slave Single IPIF is the only PLBV46 IPIF that supports allowing the User IP to run from a separate, slower clock than the PLB v4.6 bus clock. (This feature is not available in the PLBV46 Slave Burst, PLBV46 Master Single, or PLBV46 Master Burst IPIF modules). The PLBV46 Slave Single IPIF supports a 1:1 and a 2:1 clock ratios which are controlled by the parameter C_BUS2CORE_CLK_RATIO as shown in Table 1-1.

Table 1-1: PLBV46 Slave Single C_BUS2CORE_CLK_RATIO

C_BUS2CORE_CLK_RATIO Value	Resulting Bus to Core Clock Ratio
1	1:1
2	2:1

Note: C_BUS2CORE_CLK_RATIO is not configurable through the Create IP Wizard.

When C_BUS2CORE_CLK_RATIO = 2, synchronization registers are included in the PLBV46 Slave Single IPIF module. Note, however, that the PLBV46 Slave Single IPIF module will not create the clock for the User Logic, this clock must be connected directly to the User Logic. It will not route through the IPIF and the User Logic should not connect to the Bus2IP_Clk output from the IPIF. Also note that the User Logic clock and the PLB v4.6 bus clock must be edge synchronous, i.e., must be outputs from the same DCM.

PLBV46 Slave Burst IPIF Module

The PLBV46 Slave Single IPIF module implements the PLB v4.6 performance feature subset for a slave peripheral. Its main target use is for higher-throughput slaves such as memory controllers and bridges. Figure 1-4 shows the block diagram for the PLBV46 Slave Burst IPIF module.



Figure 1-4: PLBV46 Slave Burst Block Diagram

The PLBV46 Slave Burst IPIF module supports a 32-bit address bus and a 32-bit, 64-bit, or 128-bit data bus. Cores utilizing this IPIF can connect to PLB v4.6 busses of 32-bits, 64-bits and 128-bits as required by the Xilinx PLB v4.6 specification.

Single read and write data transfer, fixed length burst transfers (up to 16 data beats), and cacheline transfers are all supported by the PLBV46 Slave Burst IPIF module. The cacheline read transfer response can be configured to be either target-word first or line word first.

PLBV46 Slave IPIF Parameters

The PLBV46 Slave IPIF modules have parameters that are not available in the existing OPB and PLBV34 IPIF modules. These parameters are discussed in detail in the Migration of User IP Slave Cores chapter. The user is encouraged to refer to the data sheets for the PLBV46 Slave IPIF modules for more details.

Data width parameters should be given special consideration as to the system that this IP will be used in. When these parameters are of mis-matched values, additional logic within the slave IP may be required in order to support burst run length expansion, byte-enable muxing, and read data bus steering. Please refer to the section, "Master/Slave Mixed Data Width Considerations" of this chapter for more information.

PLBV46 Slave IPIC Signal Comparison

The following tables show comparisons of the IPIC signal sets for the existing OPB and PLBV34 IPIF modules and the PLBV46 Slave IPIF modules. Please refer to the data sheets for the PLBV46 Slave IPIF module of interest for more details and descriptions of these signals.

The signals from the IPIF modules to the User Logic (Bus2IP) are compared in Figure 1-5.

OPB	PLBV34	PLBV46
Bus2IP_Data	Bus2IP_Data	Bus2IP_Data
Bus2IP_Addr	Bus2IP_Addr	Bus2IP_Addr
Bus2IP_RNW	Bus2IP_RNW	Bus2IP_RNW
Bus2IP_BE	Bus2IP_BE	Bus2IP_BE
Bus2IP_Burst	Bus2IP_Burst	Bus2IP_Burst (Only for PLBV46 Slave Burst)
Bus2IP_WrReq	Bus2IP_WrReq	Bus2IP_WrReq (Only for PLBV46 Slave Burst)
Bus2IP_RdReq	Bus2IP_RdReq	Bus2IP_RdReq (Only for PLBV46 Slave Burst)
Bus2IP_CS	Bus2IP_CS	Bus2IP_CS
Bus2IP_CE	Bus2IP_CE	
Bus2IP_RdCE	Bus2IP_RdCE	Bus2IP_RdCE
Bus2IP_WrCE	Bus2IP_WrCE	Bus2IP_WrCE
	Bus2IP_IBurst	
	Bus2IP_Abort	
		Bus2IP_BurstLength (Only for PLBV46 Slave Burst)

UG443_1_5_082007

Figure 1-5: Comparison of Slave IPIC Signals (Bus2IP)



Since only fixed length bursts are supported by PLBV46 Slave Burst IPIF modules, the signal, Bus2IP_BurstLength, has been added to supply the User Logic of the IP with the length of the current burst transaction. Knowledge of the transaction burst length may provide the opportunity to optimize the User Logic.

The signals from the User Logic to the IPIF module are compared in Figure 1-6.

OPB	PLBV34	PLBV46
IP2Bus_Data	IP2Bus_Data	IP2Bus_Data
IP2Bus_WrAck	IP2Bus_WrAck	IP2Bus_WrAck
IP2Bus_RdAck	IP2Bus_RdAck	IP2Bus_RdAck
IP2Bus_Retry	IP2Bus_Retry	
IP2Bus_Error	IP2Bus_Error	IP2Bus_Error
IP2Bus_ToutSup	IP2Bus_ToutSup	
IP2Bus_PostedWrInh		
	IP2Bus_Busy	
	IP2Bus_AddrAck	IP2Bus_AddrAck (Only for PLBV46 Slave Burst)
	IP2Bus_Bterm	

UG443_1_6_082007

XILINX

Figure 1-6: Comparison of Slave IPIC Signals (IP2Bus)

IP2Bus_Retry had limited use in existing IP. It originated in the OPB IPIF so that User IP could control the Sln_Retry signal on the OPB. Assertion of this signal terminated the OPB transaction without transferring data and informed the Master that the transaction could be tried again later. The PLBV34 and PLBV46 buses do not have a retry signal. It was present on the IPIC of the PLBV34 IPIFs simply for compatibility with the OPB IPIC. Most processor IP did not utilize this signal, therefore, it was removed from the PLBV46 IPIC signal set. If the User IP Logic requires a method of communicating to the master that it is busy with another operation, this is best implemented by means of a status register.

IP2Bus_ToutSup also originated in the OPB IPIF so that User IP could control the Sln_ToutSup signal on the OPB to suppress the 16-clock timeout counter in the OPB arbiter. This signal is not present on the PLBV34 or PLBV46 buses. It was present in the PLBV34 IPIC, however, to suppress the 64-clock timeout counter present in the PLBV34 IPIF to terminate a data phase transaction that wasn't acknowledged by the User IP. The PLBV46 IPIF now contains a 128-clock data phase timeout counter. With the much longer timeout value, it was deemed unnecessary to provide a suppression capability, therefore, this signal is not present in the IPIC for PLBV46.

IP2Bus_PostedWrInh added huge complexities for OPB IPIFs that didn't seem to merit the slight additional flexibility in user IP, therefore, this signal is no longer available.

IP2Bus_Busy was predominantly used in Master/Slave combinations to indicate to the slave attachment that the master attachment was utilizing the slave attachment and therefore it could not accept any bus transactions. For PLBV46 IPIF modules, the master and slave modules are now completely separate. If there are portions of the User IP logic that need to be protected from bus transactions during certain operations of the accompanying master logic, the User Logic must accommodate this in the User Logic

design as there is no longer a busy indicator from the User Logic to the PLBV46 Slave IPIF module.

IP2Bus_Bterm has been removed because the early termination of a burst is no longer allowed.

IPBus_AddrAck allows the User Logic to request addresses separately and ahead of acknowledging data. This is only useful during burst transactions and therefore this signal is only supported in PLBV46 Slave Burst IPIF modules.

PLBV46 Master IPIF Modules

PLBV46 Master Single IPIF Module

The PLBV46 Master Single IPIF module implements the PLB v4.6 baseline feature subset for a master peripheral. Its main target use is for register accesses within user IP. Figure 1-7 shows the block diagram of the PLBV46 Master Single IPIF module.



Figure 1-7: PLBV46 Master Single Block Diagram

The PLBV46 Master Single IPIF module supports a 32-bit address bus and a 32-bit data bus. Note that 64-bit and 128-bit data widths are not supported, therefore, a core utilizing this IPIF can only have a data-width of 32-bit. However, cores utilizing this IPIF can connect to PLBV46 busses of 64-bits and 128-bits as required by the IBM 128-bit Processor Local Bus Architectural Specification (v4.6).

Only single read and write data transfers can be initiated by the PLBV46 Master Single IPIF module. Burst transfers and cacheline transfers can not be initiated with this IPIF.

The PLBV46 Master Single IPIC signals used to initiate transfers with the existing OPB and PLBV34 IPIF modules have been replaced with a reduced LocalLink interface. This reduced interface contains source and destination ready signals and read and write data busses. It does not include frame delimiter signals since only single transfers can be initiated.

PLBV46 Master Burst IPIF Module

The PLBV46 Master Burst IPIF module implements the PLB v4.6 performance feature subset for a master peripheral. Its main target use is for high performance, high data throughput masters. Figure 1-8 shows the block diagram of the PLBV46 Master Burst IPIF module.



Figure 1-8: PLBV46 Master Burst Block Diagram

The PLBV46 Master Burst IPIF module supports a 32-bit address bus and a 32-bit, 64-bit, or 128-bit data bus. Cores utilizing this IPIF can connect to PLB v4.6 busses of 32-bits, 64-bits and 128-bits as required by the IBM 128-bit Processor Local Bus Architectural Specification (v4.6).

Single read and write data transfers and fixed length burst transfers (up to 16 data beats) can all be initiated by the PLBV46 Master Burst IPIF module to PLB v.4.6 slaves of the same or different sizes. With PLBV46 Master Burst, the User IP reads and writes from the PLB Master via the Xilinx LocalLink Interface protocol which is a different protocol than PLBV34/OPB IPIFs.

Existing OPB and PLB v3.4 master/slave cores now require 2 separate IPIFs for the master and slave interface. For example, PLBV46 Slave Single can be used for slave registers to setup master transactions and the PLBV46 Master Burst can be used for the Master. The Master read/write transactions no longer involves the slave attachment. No internal slave transactions are required to support the master transactions. Therefore, the master no longer supplies an IP2IP address.

Indeterminate length bursts are not supported with PLBV46 Master Burst. The User Logic must only generate fixed length bursts of up to 16 data beats. The length is specified in bytes and a multiple of the Native Dwidth/8. In addition, transactions cannot be aborted. With unaligned addresses, the User IP has to issue single data beat requests until address alignment is established. Critical slave registers that control Master operations must be considered since there is no IP2Bus_Busy signal from the slave.

No master services like DMA are included with the PLBV46 Master IPIFs. The user has the option of adding LocalLink interface(s) to the User Logic connecting to SDMA which is described later inside the Migration of DMA Solutions chapter.

Conversion cycles are supported when transferring data to smaller data-width slaves. See section, "Master/Slave Mixed Data Width Considerations", for more details.

PLBV46 Master IPIF Parameters

Data width parameters should be given special consideration as to the system that this IP will be used in. These parameters are discussed in detail in the Migration of User IP Master/Slave Cores chapter. When these parameters are of mis-matched values, additional logic within the master PLBV46 IPIF may be required in order to support burst run length expansion and conversion cycles. Please refer to the section, "Master/Slave Mixed Data Width Considerations" of this chapter for more information. Note, however, that this automatic inclusion of the conversion cycle logic and burst run length expansion logic based on the can be turned off via a parameter.

PLBV46 Master IPIC Signal Comparison

The PLBV46 Master IPIC ports are much different from the OPB and PLBV34 IPIC Ports. The PLBV46 Master IPIC now contains a command interface, a read LocalLink interface, and a write LocalLink interface. The command interface is used to setup the transfer and the read and write LocalLink interfaces are used for the actual data transfer. Please refer to the data sheets for the PLBV46 Master IPIF module of interest for more details and descriptions of these signals.

The signals for the IPIC Command Interface are seen in Table 1-2.

Port Name	I/O	Description
IP2Bus_MstRd_Req	Ι	User logic read request.
IP2Bus_MstWr_Req	Ι	User logic write request.
IP2Bus_Mst_Addr	Ι	User logic request byte enables.
IP2Bus_Mst_BE	Ι	User logic request length in bytes for fixed length burst transfers.
IP2Bus_Mst_Type	Ι	User logic request type indicator. 0 - single data beat, 1 - fixed length burst.
IP2Bus_Mst_Lock	Ι	User logic bus lock request.
IP2Bus_Mst_Reset	Ι	Optional user logic reset request.
Bus2IP_Mst_CmdAck	0	Command acknowledge status.
Bus2IP_Mst_Cmplt	0	Command complete status.
Bus2IP_Mst_Error	0	Command error status.
Bus2IP_Mst_Rearbitrate	0	User logic should ignore this signal
Bus2IP_Mst_Cmd_Timeout	0	Command timeout status.

Table 1-2:	PLBV46 Master Burst and PLBV46 Master Single IPIC Command
Interface S	Signals



Some of the command interface signals have similar functionality to the existing OPB and PLBV34 IPIC signal as shown in Figure 1-9 and Figure 1-10.

OPB	PLBV34	PLBV46
Bus2IP_MstWrAck	Bus2IP_MstWrAck	
Bus2IP_MstRdAck	Bus2IP_MstRdAck	
Bus2IP_MstRetry	Bus2IP_MstRetry	
Bus2IP_MstError	Bus2IP_MstError	Bus2IP_Mst_Error
Bus2IP_MstTimeout	Bus2IP_MstTimeout	Bus2IP_Mst_Timeout
Bus2IP_MstLastAck	Bus2IP_MstLastAck	
Bus2IP_IPMstTrans	Bus2IP_IPMstTrans	
		Bus2IP_Mst_CmdAck
		Bus2IP_Mst_CmpIt
		Bus2IP_Mst_Rearbitrate

DS443_1_9_082007

Figure 1-9: Comparison of Master IPIC Signals (Bus2IP)

OPB	PLBV34	PLBV46
IP2Bus_Addr	IP2Bus_Addr	IP2Bus_Mst_Addr
IP2Bus_MstBE	IP2Bus_MstBE	IP2Bus_Mst_BE
IP2IP_Addr	IP2IP_Addr	
IP2Bus_MstWrReq	IP2Bus_MstWrReq	IP2Bus_MstRd_Req
IP2Bus_MstRdReq	IP2Bus_MstRdReq	IP2Bus_MstWr_Req
IP2Bus_MstBurst	IP2Bus_MstBurst	IP2Bus_Mst_Type
IP2Bus_MstBusLock	IP2Bus_MstBusLock	IP2Bus_Mst_Lock
	IP2Bus_MstNum	
		IP2Bus_Mst_Length
		IP2Bus_Mst_Reset

DS443_1_10_082007

Figure 1-10: Comparison of Master IPIC Signals (IP2Bus)

For the actual transfer of data, the LocalLink interfaces are used. Since the PLBV46 Master Single IPIF module can only initiate single transactions and can not initiate burst transactions, this interface signal set is reduced from the signal set for the PLBV46 Master Burst IPIF module.

The PLBV46 Master Burst IPIC Read LocalLink Interface Signals are seen in Table 1-3 and the IPIC Write LocalLink Interface Signals are seen Table 1-4. The PLBV46 Master Single IPIC Read Data Interface and IPIC Write Data Interface are shown in Table 1-5 and Table 1-6.

Note: For a write master transaction, the User Logic is the source and the Master is the destination. For a read master transaction, the User Logic is the destination and the Master is the source.

Port Name	I/O	Description
Bus2IP_MstRd_d	0	Read data output to user logic.
Bus2IP_MstRd_REM	0	LocalLink remainder indicator.
Bus2IP_MstRd_sof_n	0	Active low signal indicating the starting data beat of a read LocalLink transfer.
Bus2IP_MstRd_eof_n	0	Active low signal indicated the ending data beat of a read LocalLInk transfer.
Bus2IP_MstRd_src_rdy_n	0	Active low signal indicating that the data value asserted on the Bus2IP_MstRd_d bus is valid.
Bus2IP_MstRd_src_dsc_n	0	Active low signal that the Source needs to discontinue the transfer.
IP2Bus_MstRd_dst_rdy_n	Ι	Active low signal indicating that the data value asserted on Bus2IP_MstRd_d Bus is being asserted by the destination.
IP2Bus_MstRd_dst_dsc_n	Ι	Active low signal indicating that the Destination (User logic) needs to discontinue the transfer.

Table 1-3: PLBV46 Master Burst IPIC Read LocalLink Interface Signals

 Table 1-4:
 PLBV46 Master Burst IPIC Write LocalLink Interface Signals

Port Name	I/O	Description
IP2Bus_MstWr_d	Ι	Write data input from User logic.
IP2Bus_MstWr_REM	Ι	LocalLink remainder indicators.
IP2Bus_MstWr_sof_n	Ι	Active low signal indicating the starting data beat of a write LocalLink transfer.
IP2Bus_MstWr_src_rdy_n	Ι	Active low signal indicating that the data value asserted on IP2Bus_MstWr_d Bus is valid.
IP2Bus_MstWr_src_dsc_n	Ι	Active low signal indicating that the data value asserted on IP2Bus_MstWr_d bus is being asserted by the Destination.
Bus2IP_MstWr_dst_rdy_n	0	Active low signal indicating that the data value asserted on IP2Bus_MstWr_d bus is being asserted by the Destination.
Bus2IP_MstWr_dst_dsc_n	0	Active low signal indicating that the Write LocalLink Destination(Master) needs to discontinue the transfer.



Port Name	I/O	Description
Bus2IP_MstRd_d	0	Read data output to User Logic.
Bus2IP_MstRd_src_rdy_n	0	Active low signal indicating that the data value asserted on the Bus2IP_MstRd_d Bus is valid.

Table 1-5:	PLBV46 Master	Single IPIC Read	d Data Interface Signals
------------	---------------	------------------	--------------------------

Table 1-6: PLBV46 Master Single IPIC Write Data Interface Signals

Port Name	I/O	Description
IP2Bus_MstWr_d	Ι	Write data input from the User Logic.
IP2Bus_MstWr_dst_rdy_n	0	Active low signal indicating that the data value asserted on the IP2Bus_MstWr_d Bus has been transferred on the PLB.

Example state machines used in the User Logic to interface to these LocalLink interfaces are created by the Create/Import IP Wizard.

Master/Slave Mixed Data Width Considerations

Since a 128-bit data width masters can exist and most PLB v4.6 slaves are 32-bit data width, it is important to understand the system implications of having mixed data widths masters and slaves.

Burst Run Length

The data width of a burst transaction is determined by the value of the PLB_Size signals. The length of a burst transaction is determined by the value of the PLB_BE signals. The width of the PLB_Size vector is the same for masters and slaves regardless of their data widths. However, since the PLB_BE vector at each peripheral is sized as the data width of the peripheral / 8 since for single transfer types these signals indicate the valid data byte lane. Therefore, the size of the PLB_BE vector can be different at peripherals of different data widths. For this reason, the Xilinx PLB v4.6 interconnect limits the length of a burst to 16 data beats so that this length can always accurately represented on the PLB_BE vector for all devices. (The smallest data width allowed for peripherals in the system is 32-bits which dictates a PLB_BE vector of 4 bits which adequately represents a fixed length burst of 16 data beats).

Note, however, that the PLB_BE vector represents the number of data beats as indicated by the PLB_Size vector. For a 128-bit data width master, the value of the PLB_BE vector can represent the length of burst of 128-bit wide data beats. If this transaction is received by a 32-bit data width slave, the PLB_BE vector value no longer accurately represents the amount of data in the transaction.

Therefore, when a slave is connected to a wider data-width master, the slave is required to dynamically adjust their burst run length. Likewise, when a master is connected to a narrower data-width slave, the master is required to dynamically adjust the burst length if the size of the fixed length burst exceeds the native data width of the target slave.

Slaves are also required to adjust the number of data beats needed for cacheline transfers when the transaction involves mixed slave/master data widths.

Burst Run Length Example:

An example of a 128-bit PLB v4.6 master requesting a fixed length burst of 16 doublewords to a 32-bit data-width slave is described below:

- 128-bit PLB v4.6 Master requests a 16 beat burst of double words of a 32-bit Slave
 - PLB_Size = "1011", PLB_BE = "1111"
- 2 words per double word * 16 data beats = 32 words being requested
- 32 data beats required from Slave, but PLB_BE only indicates 16
- Slave must adjust burst run length accordingly to return 32 words
 - Can't use PLB_BE value to determine burst run length
- Master must also adjust burst run length to know how many data acknowledges needed to complete the burst transaction so that the burst signal is asserted for the proper number of clocks

Conversion Cycles

Masters are encouraged to support Conversion cycles (master accessing narrower data width slaves) if the requested byte enables can cross the native data width alignment of the target slave. When slaves are connected to a wider data-width master or a wider data-width PLB v4.6 interconnect, the slaves must implement a mux which based on the values of the lower address bits, selects the appropriate slice of the larger PLB_BE vector to assign to the PLB_BE vector of the size associated with the slave's data width. This is shown for a 32-bit data-width slave connecting to a 128-bit data-width PLB v4.6 in Figure 1-11 with the exception that the byte enable mux is shown external to the slave in this figure and in the Xilinx PLB v4.6 implementation, this mux is part of the slave device.





Figure 1-11: Byte Enable Muxing 32-bit Data-width Slave, 128-bit Data-width PLBV46 bus

Conversion Cycle Example:

An example of a 128-bit data-width master performing conversion cycles to a 32-bit data-width slave is described below:

- 128-bit Master requesting a single read of two bytes from address 0x07 from a 32bit slave:
 - PLB_ABus = 0x07, PLB_BE = 0x0180 (0000 0001 1000 0000)
- BEs cross the boundary of the 32-bit slave
- Slave is required to mux BEs based on PLB_ABus(28:29)
- Slave BEs = 0001, Slave returns 32-bits from address 7
- Master detects slave is 32-bits, generates conversion cycle:
 - PLB_Abus = 0x08, PLB_BE= 0x0080 (0000 0000 1000 0000)
- Slave BEs = 1000, Slave returns 32-bits from address 8

Read Data Bus Steering

When a slave is accessed by a smaller data-width master for a read transaction, the slave must properly place the data on the correct byte lanes of the read data bus so that the master can access the read data. This is called read data bus steering. Figure 1-12 shows the connections between a 128-bit data-width slave and a 32-bit data-width master.





Read Data Bus Steering Example:

An example of a 32-bit data-width master requesting a read from a 128-bit data-width slave is described below:

- 32-bit master requesting a read from a 128-bit slave from address 0x0C
- Data from 0x0C would appear on Sl_RdData(96:127)
- Due to 32-bit master size, slave steers data to Sl_RdData(0:31)

Mixed Master/Slave Data-width Support in Xilinx IP

PLBV46 Master Burst and PLBV46 Slave Burst IP will also support burst run length adjustment and data steering and mirroring. These cores can also support conversion cycles.





Chapter 2

PPC405 - PPC440 System Migration

Introduction

This chapter describes the migration process of a Xilinx embedded processor system from a PPC405 OPB/PLB v3.4 system to a PowerPC[™] 440 with PLB v4.6/XPS system.

The following high-level migration steps will be discussed:

- PPC440 processor block
- PLB v4.6 bus instances are added to the system where OPB/PLB v3.4 buses are removed
- Clocking/Reset scheme
- Equivalent XPS cores (from OPB/PLB v3.4) are connected to PLB v4.6 instances
- Ethernet solution
- Existing memory controllers are migrated to either the PPC440MC DDR2 or MPMC

The original system is built for a VirtexTM-4FX PowerPC 405 system similar to a system for the ML410 board and the migrated system is built for a Virtex-5FXT PowerPC 440 system with a generic board.

System Migration Methodology

System Hardware Migration

The method that is recommended for doing the system hardware migration is outlined below.

- Migrate the processor from the PPC405 to the PowerPC 440
- Separate out the main memory, slave and master/slave cores of the existing PPC405 system
- Migrate PLB DDR2 or MCH OPB DDR2 memory controllers to the equivalent PPC440MC DDR2 memory controller
 - If an existing SDRAM or DDR memory controller is inside the system, the MPMC memory controller is used
- Master cores
 - These are connected to either the SPLB0 or SPLB1 or MPLB PLB v4.6 bus ports on the processor block depending on functionality of the master
 - The maximum number of masters connected to SPLB0 or SPLB1 is 4 each (8 total)



- Slave cores
 - These are connected to the MPLB bus port on the processor block
 - The maximum number of slaves connected to MPLB is 16
- Utilize the DMA controller or the XPS Central DMA, as appropriate depending on the system
- Determine clocking requirements for the PowerPC 440 system and setup the PLL and clock generator
- Determine the reset requirements and the interrupt connections

System Software Migration

The basic considerations for migrating the user applications from the PPC405 to the PowerPC 440 system are listed below. A more detailed section covering software migration is discussed later in this chapter.

- Set software platform settings correctly
- Review cache requirements and cache sizes if necessary
- Set compiler options appropriately for each SW application

Existing PPC405 OPB/PLB v3.4 System

For reference, an example PPC405 system is shown in Figure 2-1. To highlight typical steps in the migration process, this system is shown with some of the common cores found in an embedded system.



Figure 2-1: PPC405 Example System

PowerPC 440/XPS System

The migrated PowerPC 440 system is shown in Figure 2-2.



Figure 2-2: PowerPC 440 Migrated System

Overview of System Migration

Processor Migration

Within the EDK project, the PowerPC 440 Virtex-5 is added to the system.

The PPC440 processor block allows the system to connect PLB Masters, PLB Slaves, DCR Master/Slaves, DMA devices through the crossbar. PLB Slave peripherals connect to the MPLB port and PLB Masters connect to the SPLB0 and SPLB1 ports on the processor block. Either the PPC440MC DDR2 or MPMC with the MIB PIM can connect to the PPC440MC on the processor block.

The MPLB and PPC440MC bus interfaces looks like a slave connections to the PowerPC 440 processor block.

SPLB0 and SPLB1 bus interfaces look like master connections to the PowerPC 440 processor block.

The four LocalLink/DMA interfaces provide DMA to the MPLB and/or PPC440MC inside the system. DMA operations are accomplished through the LocalLink protocol.

Refer to the *Virtex-5 Embedded Processor Block for PowerPC 440 Designs Reference Guide* for more details about the PowerPC 440.

Addressing Inside the PowerPC 440 Processor Block

There are two slave ports on the PowerPC 440 processor block, the Memory Controller (PPC440MC) port and the MPLB port that are configured through the EDK PPC440 wrapper.

There are two master ports, SPLB0 and SPLB1. Master PLB v4.6/XPS devices should be connected to either one of these ports.



The memory address ranges for each master port to the slave ports has to be specified. For SPLB0 and SPLB1:

- Set the base address and high address for the PPC440MC
- Set the base address and high address for the MPLB

Set the address for each SPLB0 and SPLB1 through the EDK GUI as shown in Figure 2-3.

ľ	PowerPC Addressee Bus Featur	es Cache Memory (Controller Reset DMA APU	Buses	
	C_SPLB0C_SPLB0NUM_MPLB_ADDR_RM	IG	1		
	C_SPLB0_RNG_MC_BASEADDR	0x0000000	C_SPLB0_RNG_MC_HIGHADDR	0x01FFFFFF	Memory address range
	Range0 MPLB Base Addr	0x4000000	Range0 MPLB High Addr	0x7FFFFFF	PLB Slaves address range
	Range 1 MPLB Base Addr	Oxfffffff	Range1 High Addr	0x0000000	
	Range2 MPLB Base Addr	Oxfffffff	Range2 MPLB High Addr	0x0000000	
	Range3 MPLB Base Addr	Oxfffffff	Range3 MPLB High Addr	0x0000000	
					DS443_6_3_090607

Figure 2-3: Setting Address Ranges for SPLB0 and MC

In this example, the MPLB is configured for only one address range. Masters connected to SPLB0 and SPLB1 will see one address range for memory and one address range, in this system, for slaves:

- Memory, via addresses 0x0000_0000 0x0FFF_FFFF
- Other PLBV46 slaves, via addresses 0x4000_0000 0x7FFF_FFFF

The user is allowed to configure up to four independent ranges for the slave peripherals connected to the MPLB port. This is beneficial during migration when main memory is in conflict with MPLB addresses. An example of this is when PLB DDR2 is migrated to PPC440MC DDR2. In this case, the address space of PLB DDR2 would be in the MPLB address space if all peripherals were mapped to one address space. The address ranges need to set identically for SPLB0 and/or SPLB1.

Figure 2-4 shows an address mapping example for migrating from the PPC405 to the PowerPC 440 In this example, the PLB DDR2 memory controller is replaced by the PPC440MC DDR2 so that is the address range for the MC. The PLB BRAM Controller is replaced by the XPS BRAM Controller so it's address is mapped to the MPLB range 0. The PLB GPIO is replaced with the XPS GPIO so it's address is mapped to MPLB range 1.



Figure 2-4: Address Mapping Example

Interconnect

The PLB v3.4 and OPB instances are deleted and PLB v4.6 instances are added for the MPLB, SPLB0, and SPLB1 as needed. In addition, the PLB2OPB and OPB2PLB bridges are removed.

Connecting Peripherals to MPLB, SPLB0, SPLB1

A PLB v4.6 bus instance is used on the MPLB to connect slave peripherals to the system.

PLB v4.6 bus instances are used for SPLB0 and SPLB1. Two master peripherals could be on the same bus but there are pros and cons for each scenario.

- Two master PLBs (using SPLB0 and SPLB1)
 - Higher performance but additional resources
- One master PLB (using either SPLB0 or SPLB1)
 - Lower resource utilization but lower performance
 - Would need arbitration logic for multiple masters

It is also possible to connect both masters to the MPLB (slave) bus to realize an even lower resource utilization, but it would adversely affect the performance of the system and would need arbitration logic. In this case, masters on the MPLB could only initiate transactions to slaves on the MPLB and not to the PPC440MC port.

PLB v4.6 DWIDTH Settings

Platgen sets the DWIDTH parameter for each bus instance and all the peripherals connected to each bus to be the maximum value of the NATIVE_DWIDTH parameter among all the peripherals connected to the bus. Per the IBM 128-bit Processor Local Bus Architectural Specification (v4.6), the PLB v4.6 bus must be at least as wide as the widest



connected master/slave interface. All narrower masters/slaves adapt their connections to connect to the wider bus using bit mirroring and (for slaves) ByteEnable muxing. Masters would use conversion cycles. The NATIVE_DWIDTH for all PLB v4.6 ports on the PowerPC 440 is 128 bits and is invariant. This will always cause platgen to set the DWIDTH of all buses connected to any of these PowerPC 440 ports to 128 bits. During operation, any of the PowerPC 440 PLB v4.6 interfaces will accept all transactions from, and issue compatible transactions to, other peripherals of any NATIVE_DWIDTH (32, 64, or 128 bit).

Memory Controller Migration

The DDR2 memory controller has been replaced with the PPC440MC DDR2 memory controller which connects directly to the Memory Interface Block (MIB) on the PPC440 processor block. By having the dedicated port for memory transfers, the PLB v4.6 bus is free of direct processor-to-memory transactions.

For DDR or SDRAM memory controller system migration, MPMC is connected to the MIB through a port configured for the MIB Port Interface Module (PIM).

Interrupt Controller

The OPB Interrupt Controller has been replaced with the DCR Interrupt Controller (INTC) which connects directly to the DCR port on the processor block. The XPS INTC core could have also been used which connects to the MPLB PLB v4.6 bus instance.

For the example migrated system shown, the DCR INTC was chosen to demonstrate the use of the DCR port on the processor block. A DCR bus will have to be added into the system to connect the DCR INTC to the processor block.

There are two DCR ports on the processor block, the SDCR port which is used to connect master devices on the DCR bus, and the MDCR which is used to connect slave devices. The DCR INTC connects to the MDCR port.

The address ranges for the DCR need to be set up in both the PowerPC 440 configuration and the DCR INTC configuration.

The address range set in the PowerPC 440 configuration sets the address range for the DCR registers inside the processor block. The address range set for the DCR INTC configuration sets the address range for the peripheral.

XPS Peripherals

This section covers the general migration from an OPB/PLB v3.4 inside the system to a XPS core. This is applicable to slave and master/slave cores.

- 1. The former PLB v3.4 or OPB core is deleted.
- 2. The XPS core is added to the system.
- 3. Parameters and ports are connected in the same manner as the PLB v3.4 or OPB equivalent.

The slave or master/slave interfaces are connected to the PLB v4.6 shared bus instance

In this case, The XPS BRAM and XPS UART16550 cores replace the PLB BRAM and OPB UART16550 cores, respectively. These slave devices are connected to the MPLB port PLB v4.6 bus instance which connects all XPS slave cores in the system.

The OPB PCI core is replaced by the PLBV46 PCI core. This core has a connection to the slave PLB v4.6 bus for configuring the core and also has a master connection. The master connection can be connected to SPLB0 or SPLB1 (master) buses to enhance performance.

Another consideration for the PLBV46 PCI core is that, unlike the OPB PCI core, it does not have an internal SGDMA. Therefore, to handle SGDMA operations for the PCI, an external DMA Controller must be added into the system. The XPS Central DMA core is used to provide DMA to the system.

The XPS Central DMA has a connection to the slave PLB v4.6 bus for configuring the slave registers for DMA operations and a master connection. The master connection can be connected to one of the SPLB0 or SPLB1 (master) buses to enhance performance.

System Clocking

The PowerPC 440 processor block requires PLL outputs to drive some of the clocks. The PLL adds delays to certain clocks to match internal clock delays of the PowerPC 440 processor block. Setting the value for C_CLKOUTn_DESKEW_ADJUST to PPC will automatically add the proper delay for the outputs that drive MPLB, SPLB0, SPLB1, MC (MIB), and APU clocks.

The PLL module has six available outputs. The output frequencies of each of the clock outputs can be set independently as multiples of the input clock. The parameters C_CLKFBOUT_MULT and C_CLKOUTn_DIVIDE set the output frequency for each clock output.

The PowerPC 440 Processor Block clocking structure is shown in Figure 2-5. The PLL delays are shown going to the PLB I/F (PLB Clock), the MIB I/F (MI Clock), and the APU (FCM Clock).

Also shown in Figure 2-5 is an example clocking ratio scheme for the processor block clocks.



Figure 2-5: Processor Block Clock Diagram
Reset Structure

The PowerPC 440 reset structure is similar to a PPC405 system. The proc_sys_reset module is using the latest version. The PPC440 processor block has the same reset bus interface as the PPC405 system that connects to the proc_sys_reset module.

It is recommended to use separate resets to each bus from the Bus_Struct_Reset output of the proc_sys_reset module. For master and slave peripherals connected to PLB v4.6 instances, the peripherals get vectorized resets for each master and slave connection from the bus core.

Ethernet Controller Migration

The migration of the PLB TEMAC to the XPS LL TEMAC controller varies depending on the features configured for the PLB TEMAC core.

If the PLB TEMAC core in the PPC405 PLB v3.4 system was configured to include the Scatter-gather DMA feature, the migrated system will require a DMA Controller with Scatter-gather DMA capability to be added. The DMA controller on the processor block provides Scatter-gather DMA capability to either the MPLB or PPC440MC ports. The XPS LL TEMAC connects to this DMA controller via the LocalLink port.

If the PLB TEMAC core in the PPC405 PLB v3.4 system was configured to use FIFO direct mode, the migrated system will require the addition of the XPS LL FIFO core. The XPS LL TEMAC connects to the XPS LL FIFO via the LocalLink port. The FIFO data is then accessed through the PLB v4.6 slave connection on the XPS LL FIFO.

XPS LL TEMAC instantiates either a hard TEMAC or a soft TEMAC into the core. For a V4FX/V5FXT system, XPS LL TEMAC will instantiate the hard TEMAC. XPS LL TEMAC has two hard TEMACs available inside the core.

Migration of the JTAG Controller

An updated version of the JTAG Controller has been developed for use with the PowerPC 440 processor block. It is jtagppc_cntlr version 2.01.a This will directly replace the original PPC405 JTAG Controller.

System Migration Inside an EDK System

There are two basic methods for migrating from a PPC405 system to the PowerPC 440 system. One way is to start an EDK project from scratch and add in processor and the equivalent buses and peripherals. The other method is to migrate by starting with the original system and replacing the processor, buses and peripherals and making the appropriate internal/external connections and parameter changes. The following general steps can be a reference for doing either migration method.

Before migrating to the PowerPC 440 processor inside the EDK project, the Virtex-5FXT device can be selected through the EDK Project Options GUI. This allows the user to pick the specific device size, package, and speed grade.

In addition, migrating from a Virtex-4 device to a Virtex-5 device requires modifications to be made to the UCF file for LOC constraints if using an existing system as a starting place.

www.xilinx.com



Migrating Buses/Processor

Migrating the Bus

Adding/Removing Bus Instances

Remove the opb, plb, plb2opb and opb2plb instances (Delete instance and its internal ports) inside the System Assembly View/Bus Interfaces.

In addition, add 3 PLB v4.6 instances by expanding the Bus Bridge tree node. Right click on **Processor Local Bus (PLB) 4.6** and select **Add IP**. The buses are for the MPLB, SPLB0, and SPLB1. The instances created are plbv46_0, plbv46_1, and plb_v46_2.

Add the DCR bus by expanding the Bus Bridge tree node. Right click on **Device Control Register (DCR) Bus 2.9** and select **Add IP**. This creates the dcr_v29_0 instance.

Migrating the Processor

Remove the former **ppc405_0** instance by right clicking on **ppc405_0** inside the **System Assembly View/Bus Interfaces** and clicking on **Delete Instance...** (Delete instance and its internal ports).

Add the PowerPC 440 Virtex-5 to the system by expanding the Processor or the Global Peripheral Repository tree node inside the IP Catalog tab. Right click on **PowerPC 440** Virtex-5 and click on **Add IP**. This creates the ppc440_virtex5_0 instance.

Setting Parameters

Right click on **ppc440_virtex5_0** inside the System Assembly View and select **Configure IP...**.

Inside the Addresses tab, set the addresses for the DCR, SPLB0 and SPLB1.

The Base and High Address for the Memory Controller (MemCon) is automatically set by the tools.

Set the Internal DCR Register Base Address and Internal DCR Register High Address to **0b000000000** and, **0b001111111**, respectively. This sets the address range for the DCR registers inside the processor block.

SPLB0 and SPLB1 address parameters are set the same.

Check SPLB0 Decodes, MPLB Addresses, and SPLB1 Decodes MPLB Addresses. This allows masters on either SPLB0 and SPLB1 to access slave peripherals inside the MPLB address range. If unchecked, masters could only access address range inside the PPC440MC.

Set Number of MPLB Addr Ranges for both SPLB0 and SPLB1 to 2.

Set MemCon Base Addr and High Addr to **0x0000000** and **0x0FFFFFFF**. Set Range0 Base Addr and High Addr to **0x40000000** and **0x7FFFFFFF**. Set Range 1 Base Addr and High Addr to **0x80000000** and **0xFFFFFFF**. Slave peripherals on the MPLB should have an address range between 0x40000000 and 0xFFFFFFFF. This applies to both SPLB0 and SPLB1. The above configuration is shown in Figure 2-6.

					HDL Toggle Nam	es Datasheet Res
idresses	Bus Features Cache	Memory Controller	DMA Reset	APU Misc I	Buses	
Power	PE **					
MemCon —						
Base Addre	ess of Memory	0x0	0000000 High	Address of Memory		0x0FFFFFFF
DCR						
Internal DC	R Register Base Address	ороос	Inter	nal DCR Register High	Address	0b001111111
SPLBO						
SPLB0 De	codes MPLB Addresses		Number of MPL	B Addr Ranges	2	
L	MemCon	Range 0	Range 1	Range 2	Range 3	
Base Addr	0x00000000	0x40000000	0x8000000	Oxfffffff	Oxffffffff	
High Addr	UXUFFFFFFF	Ux/FFFFFFF	UXFFFFFFF	0x0000000	0x0000000	
SPLB1						
SPLB1 De	codes MPLB Addresses		Number of MPL	B Address Ranges	2	
	MemCon	Range 0	Range 1	Range 2	Range 3	
Base Addr	0x0000000	0x40000000	0x8000000	Oxffffffff	Oxffffffff	
High Addr	0x0FFFFFFF	0x7FFFFFFF	OxFFFFFFFF	0x0000000	0x0000000	

UG443_6_6_091007

Figure 2-6: PowerPC 440 Addresses Tab

www.xilinx.com



The control and conflict DCR registers for the PPC440MC memory controller are set in the Memory Controller tab.

The memory controller will be added later in the chapter with the following settings. The memory controller has a 64-bit data width and is configured for 13 bits for row, 10 bits for column, and 2 bits for bank. In addition, the address offset is log2(data_width/8) which is 3 in this case. This information to set the address mapping, after which the Row Conflict Mask and Bank Conflict Mask registers are set.

The address mapping structure for the memory is:

[Bank Address] [Row Address] [Column Address] [Address Offset]

The setup for this example is shown in Figure 2-7.

Bank Add	ress	Row Address	Column Address	,	Offse Addres		
31	25 24	13	12	2	1	0	
				UG443_6	7 09	1007	

Figure 2-7: Example Address Map

Set Mask Used to Determine a Row Conflict to **0x003FFE00** and Mask Used to Determine a Bank Conflict to **0x00C00000**. These values are derived from placing 1s (ones) inside the bit locations in Figure 2-7 for the row and bank bits. Row bits are between bits 25 to 13 and bank bits are between bits 27 to 26.

Set the Control and Configuration for the MC Interface to **0xF810008F**. To review and set this register, see the Refer to the *Virtex-5 Embedded Processor Block for PowerPC 440 Designs Reference Guide*.

The above parameters are shown in Figure 2-8.

c440_virt	ex5_0 : ppc44	0_virtex	5_v1_00_a							
									HDL Toggle Name	rs Datasheet Resto
Addresses	Bus Features	Cache	Memory Controller	DMA	Reset	APU	Misc	Buses		
-MemCon- Mask U	sed to Determine a	Row Confl	ct							0x003FFE00
Mask U	sed to Determine a	Bank Conf	ict							0x00C00000
Control a	and Configuration f	orthe MC I	iterface							0xF810008F
Max Nu	mber of Quad-word	ls per Burst	thru Xbar to MC Interfa	асе						8 💙
-MemCon /	Arbitration									
Memory	Control Interface /	Arbitration M	ode							0 🗸
Second	ary Arbitration Prior	ity for SPLE	0, DMA0 and DMA1							1 🗸
Second	ary Arbitration Prior	ity for SPLE	1, DMA2 and DMA3							0 🖌
Second	ary Arbitration Prior	ity for all Da	ta Reads from CPU							2 🗸
Second	ary Arbitration Prior	ity for all Da	ta Writes from CPU							3 🗸
Second	ary Arbitration Prior	ity for all Ins	truction Fetches from	CPU						4
										OK Cance

Figure 2-8: PowerPC 440 Memory Controller Tab

In the DMA tab, set the Number of DMA Channel to **1**. The DMA Channel is used for the LocalLink interface on the XPS LL TEMAC.



Setting Bus Interfaces

In the Bus Interfaces Tab, expand the ppc440_virtex5_0 instance, make the selections shown in Figure 2-9.

Select the following bus interfaces:

- MDCR bus connection is connected to dcr_v29_0
- JTAGPPC bus connection is connected to jtagppc_0_JTAGPPC0
- MPLB bus connection is connected to plb_v46_0
- SPLB0 bus connection is connected to plb_v46_1
- SPLB1 bus connection is connected to plb_v46_2



UG443_6_9_091007

Figure 2-9: PowerPC 440 Bus Interfaces

Migrating The Memory Controller

Remove the DDR_SDRAM_64Mx32 instance by right clicking on **DDR_SDRAM_32Mx64** inside the System Assembly View/Bus Interfaces and clicking on **Delete Instance...**, then click on **Delete instance and its internal ports**.

Add the PPC440MC DDR2 to the system by expanding the Memory and Memory Controller or Global Peripheral Repository 0 tree node inside the IP Catalog tab. Right click on **PPC440MC_DDR2** and click on **Add IP**. This creates the ppc440_mc_ddr2_0 instance.

See the PPC440MC DDR2 documentation for IOSTANDARDS required for certain pins for the UCF which might be different from the previous memory controller.

Configuring the PPC440MC DDR2 core

Setting Parameters

Right click on ppc440mc_ddr2_0 in the System Assembly View and select Configure IP...

In the All tab, set the following parameters:

- Set C_DDR_BAWIDTH to 2
- Set C_DDR_DWIDTH to **64**
- Set C_DDR_CAWIDTH to **10**
- Set C_DDR_DM_WIDTH to 8
- Set C_DQ_BITS to 8
- Set C_DQS_BITS to **3**
- Set C_DDR_RAWIDTH to **13**
- Set C_MIB_MC_CLOCK_RATIO to 1
- Set C_MEM_BASEADDR are C_MEM_HIGHADDR to **0x00000000** and **0x0FFFFFF**, respectively
- Set C_MEM_CLK_PERIOD_PS to 5000
- Set C_NUM_IDELAYCTRL to **3**
- Set C_IDELAYCTRL_LOC to **IDELAYCTRL_X0Y5-IDELAYCTRL_X0Y4-IDELAYCTRL_X0Y3**

PPC440MC DDR2 Bus Interface

In the Bus Interfaces Tab, expand the ppc440mc_ddr2_0 instance. The PPC440MC bus interface is connected to ppc440_virtex5_0_PPC440MC as shown in Figure 2-10.



Figure 2-10: PPC440MC DDR2 Bus Interfaces

PPC440MC DDR2 Ports

The PPC440MC DDR2 ports are connected similar to the PLB DDR2 ports. The external ports for DQ and DQS external ports the Name and Net name must be the same. Discussion of connecting the system clocking needed for the memory controller is discussed later in this chapter.

Migrating Ethernet Solution

Adding/Removing Ethernet

Remove the TriMode_MAC_GMII instance by right clicking on **TriMode_MAC_GMII** inside the System Assembly View/Bus Interfaces and clicking on **Delete Instance...**, then click on **Delete instance and its internal ports**.

Add XPS LL TEMAC to the system by expanding the Communication High-Speed tree node inside the IP Catalog tab. Right click on **XPS LocalLink Tri-mode Ethernet Mac** and click on **Add IP**. This creates the xps_ll_temac_0 instance.

Setting Parameters

Right click on **xps_II_temac_0** inside the System Assembly View, then select **Configure IP...**.

The XPS LL TEMAC is configured to use a single Hard TEMAC by unselecting **Enable TEMAC1**. The GMII interface is used by setting Physical Interface Type to **GMII**. Because the system is for the Virtex-5 FPGA, the Type of TEMAC is set to **V5 Hard**. The above selections are shown in Figure 2-11.

User	System	Buses		Toggle Names Datasheet	Reston
NI .			Type of TEMAC	V5 Hard	~
			Include IO and BUFG as Needed for the PHY Interface Selected		~
			Physical Interface Type	GMII	*
			Ratio of PLB Bus Clock to Core Clock	1	*
			PHY Address for TEMAC 0	00001	
			TX FIFO Depth of TEMAC0	4096B	~
			Enable TX Checksum Official for TEMAC0		
			RX FIFO Depth of TEMAC0	4096B	~
			Enable RX Checksum Offload for TEMAC0		
			Enable TEMAC 1		
			PHY Address for TEMAC 1	00010	
			TX FIFO Depth of TEMAC1	4096B	~
			Enable TX Checksum Offload for TEMAC1		
			RX FIFO Depth of TEMAC1	4096B	~
			Enable RX Checksum Offload for TEMAC1		

Figure 2-11: XPS LL TEMAC Parameters



Setting Bus Interfaces

The slave interface SPLB of the XPS LL TEMAC core is connected to the plb_v46_0 inside the system as shown in Figure 2-12.

The LocalLink bus interface from the XPS LL TEMAC, xps_ll_temac_0_LLINK0, is connected to the LLDMA0 bus interface on the ppc440_virtex5_0, as shown in Figure 2-12.



UG443_6_12_091007

Figure 2-12: XPS LL TEMAC Bus and LocalLink Interfaces

XPS LL TEMAC Ports

The XPS LL TEMAC GMII ports are connected similar to PLB TEMAC GMII ports. The connection of the system clocking for the XPS LL TEMAC is discussed later in this chapter.

Migrating Interrupt Controller to DCR INTC

Add XPS LL TEMAC to the system by expanding the Clock, Reset and Interrupt tree node inside the IP Catalog tab. Right click on **DCR Interrupt Controller** and click on **Add IP**. This creates the dcr_intc_0 instance.

Setting Bus Interfaces

The SDCR interface from the DCR INTC is connected to the **dcr_v29_0**.

Setting Ports

Expand the tree node for ppc440_virtex5_0 and dcr_intc_0. In the Irq port on the dcr_intc_0, click on **Make New Connection** which creates the dcr_intc_0_Irq. In the EICC440EXTIRQ input signal from the ppc440_virtex5_0 is connected to dcr_intc_0_Irq.

Other interrupts inside the system should be connect to the Intr port inside the dcr_intc_0.

Migrating XPS Peripherals

In the migrated system, the PLB BRAM, OPB INTC, OPB GPIO, OPB PCI and OPB UART Lite cores are replaced with the equivalent XPS BRAM, XPS INTC, XPS GPIO, PLBV46 PCI and XPS UART Lite, respectively.

The addition of these cores is not discussed.

Modifying Clocking/Reset Inside The System

The processor system reset core is replaced with v2_00_a version of the core. In addition, the DCMs instantiations in the system are replaced with the PLL Module and the clock generator core.

Processor System Reset

Adding Processor System Reset to Migrated System

In the System Assembly View/Bus Interfaces, delete the reset_block instance (Delete instance and its internal ports).

Add the latest version of the processor system reset by expanding the Clock Reset and Interrupt tree node inside the IP Catalog tab. Right click on **Processor System Reset Module** and click on **Add IP**. This creates the proc_sys_reset_0 instance.

In the System Assembly View, right click on **proc_sys_reset_0**, then select **Configure IP**



Set External Active High to **0**, Number of Bus Structure Reset Registered Outputs to **4**, and Number of Peripheral Reset Registered Outputs to **2**. The above parameters are shown in Figure 2-13.

llog Burss	7100_3ys_1636(_)Z_00_4	HDL Toggle Names Datasheet Restore
All		
	Number of Clocks Before Input Change is Recognized On The External Reset Input	4
	Number of Clocks Before Input Change is Recognized On The Auxiliary Reset Input.	4
	External Reset Active High	0
	Auxiliary Reset Active High	1
	Number of Bus Structure Reset Registered Outputs	4
	Number of Peripheral Reset Registered Outputs	2

UG443_6_13_091007

Figure 2-13: Processor System Reset Parameters

PPLL D C R • BIF Filters P Bus Interfaces Ports Addresses Name Bus Connection IP Type IP Version BBB 1 00 a MFCM ppc440_virtex5_0_MFCM MFCB No Connection V SDCR No Connection ¥ LLDMA3 No Connection ¥ LLDMA2 ¥ No Connection LI DMA1 No Connection × proc_sys_reset_0_RESETPPC0 RESETPPC \triangleright LLDMA0 xps_II_temac_0_LLINK0 v MDCR × JTAGPPC jtagppc_cntlr_0_JTAGPPC0 ~ PPC440MC ppc440_virtex5_0_PPC440MC MPLB ġ-ġ-🌞 plb_v46_0 Y ò-ò ۵ 4 SPLB0 plb_v46_1 ¥ ò dcr_v29_0 dcr_v29 1.00.a *, olb_v46_0 ∞* plb_v46 1.00.a ◇ plb_v46_1 plb_v46 1.00.a plb_v46 1.00.a xps_bram_if_c 1.00.a 🕀 🗢 xps_bram_if_cntlr_1_bram bram_block 1.00.a 🗄 🗢 ddr2_init xps_gpio 1.00.a Ò Ó 🗄 🗢 LEDs_8Bit 1.00.a ò xps_gpio 🕀 🧼 dcr_intc_0 dcr_intc 1.00.b proc_sys_reset 2.00.a pll_module 1.00.a ₽ ● PCI32_BRIDGE 1.00.a plbv46_pc → pci_arbiter_0 pci_arbiter 1.00.a 1.00.a xps_ll_tema O, -0 ppc440mc_ddr2 1.00.a 1.00.a xps_uart16550 0 € → jtagppc_cntlr_0 2.01.a Ł jtagppc_cntlr 1.00.a xps_central_dma <u>i - i - i</u> clock_generator_0 clock_generator 1.00.a System Assembly View

In the Bus Interfaces Tab, expand the ppc440_virtex5_0 tree node. Connect RESETPPC to proc_sys_reset_0_RESETPPC0 as shown in Figure 2-14.

Figure 2-14: PowerPC 440 Reset Bus Interface



UG443 6 14 091007



In the Ports Tab, expand the tree node for proc_sys_reset_0.

Set Peripheral_Reset to dcr_rst, ll_rst, Ext_Reset_in to sys_rst_s, Bus_Struct_Reset to plbv46_0_rst, plbv46_1_rst, plbv46_2_rst & mc_rst. In addition, set Dcm_locked to dcm_all_locked. The dcm_all_locked input signal is defined in the clocking section.

The above port connections are not shown in Figure 2-15.

Bus Interfaces Ports	Addresses				1	Filters (Applied)	🚓 Add External Port
Name	Net	Direction	Range	Class	Frequency	Reset Polarity	IP Type
🗄 🧼 External Ports							
				PROCESSOR			ppc440_virtex5
				BUS			dcr_v29
				BUS			plb_v46
				BUS			plb_v46
				BUS			plb_v46
# > xps_bram_if_cntlr_1				PERIPHERAL			xps_bram_if_cntlr
xps_bram_if_cntlr_1_b				PERIPHERAL			bram_block
🗄 🧼 ddr2_init				PERIPHERAL			xps_gpio
🕀 🧼 LEDs_8Bit				PERIPHERAL			xps_gpio
⊕ → dcr_intc_0				PERIPHERAL			dcr_intc
⊕ I_module_0				PERIPHERAL			pll_module
PCI32_BRIDGE				PERIPHERAL			plbv46_pci
pci_arbiter_0				PERIPHERAL			pci_arbiter
xps_l_temac_0				PERIPHERAL			xps_II_temac
ppc440mc_ddr2_0				PERIPHERAL			ppc440mc_ddr2
xps_uart 16550_0				PERIPHERAL			xps_uart16550
itagppc_cntlr_0				PERIPHERAL			jtagppc_cntlr
xps_central_dma_0				PERIPHERAL			xps_central_dma
🗄 🧼 clock_generator_0				PERIPHERAL			clock_generator
proc_sys_reset_0				PERIPHERAL			proc_sys_reset
- MB_Reset	No Connection	✓ 0		RST			
MB_Debug_Sys_Rst	No Connection	× 1		RST			
Aux_Reset_In	No Connection	~		RST			
Peripheral_Reset	dcr_rst & II_rst	∨ 0	[0:C_NUM_PERP_RST-1]	RST			
Dcm_locked	dcm_all_locked	∨ I					
Bus_Struct_Reset	plbv46_0_rst & plbv46_1_rst & plbv46_2_rst & mc_rst	∨ 0	[0:C_NUM_BUS_RST-1]	RST			
Slowest_sync_clk	No Connection	× I		CLK			
Ext_Reset_In	sys_rst_s	× I		RST			
System Assembly View							
						110	3443 6 15 09100

Figure 2-15: Processor System Reset Connections

Expand the tree nodes for plb_v46_0, plb_v46_1, and, plb_v46_2 in the System Assembly View/Port. Connect the SYS_Rst in the bus instances to **plbv46_0_rst**, **plbv46_1_rst**, and **plbv46_2_rst**, respectively.

For master and slave peripherals connected to the PLB v4.6, the peripherals get vectorized resets for each master and slave connection from the bus core.

Expand the tree node for ppc440_mc_ddr2_0 and set the mc_mcreset port to mc_rst.

Expand the tree node for xps_ll_temac_0 and set the LlinkTemac0_RST port to **ll_rst**.

Expand the tree node for dcr_intc_0 and set the DCR_Rst to dcr_rst.

Clock Generator and PLL Module

Overview of Existing Clocking Scheme

At the outset, the system contains 2 cascading DCMs. The first DCM provides the 100 MHz clock for the bus. The 200 MHz clock provides the processor clock, PLB DDR2 Device_Clk0, and the RCLK to the OPB PCI. In addition, the first DCM provides the 125 MHz clock for the Hard TEMAC, the 25 MHz clock for the Cal_Clk for PLB DDR2, and the 33 MHz clock for the PCLK to the OPB PCI.

The second DCM provides the 200 MHz Device_Clk90_in_n for the PLB DDR2 memory controller.

With the PLB DDR2 clocks, inverters are needed for the Device_Clk0 and Device_Clk90_in_n from the DCMs to provide the PLB DDR2 Device_Clk0_n and Device_Clk90_in_n clocks.

Clocking Requirements of Migrated System

The migrated system uses the PPC440 processor block with a processor frequency of 400 MHz and the PPC440MC with a frequency of 200 MHz. The cross bar is set to 200 MHz. In addition, the MPLB, SPLB0, SPLB1 and DCR ports are set to 100 MHz.

In addition, the PPC440MC DDR2 and PLBV46 PCI needs a 200 MHz clock for the IDELAY Controllers, while the PPC440MC DDR2 needs a 200 MHz CLK0 and CLK90 clocks. Also, a 125 MHz clock is needed for the Hard TEMAC, and a 33 MHz clock is needed for PCI.

The clocks for the PLBV46 PCI core are not discussed.



The above system clocking is shown in Figure 2-16.



Figure 2-16: System Clocking Block Diagram

In the System Assembly View/Bus Interfaces, delete the dcm_0 and dcm_1 instances (Delete instance and its internal ports).

Configuring PLL Module Inside The Migrated System

Add the PLL Module by expanding the Clock Reset and Interrupt or the Global Peripheral Repository tree node inside the IP Catalog tab. Right click on **PLL for PPC440 Clocks** and click on **Add IP**. This creates the pll_module_0 instance inside the system.

Right click on pll_module_0 in the System Assembly View and select Configure IP

The PLL Module is set for clocking block diagram shown in Figure 2-16.

Set C_CLKFBOUT_MULT to **8**. This provides the module to produce 400, 200, and 100 MHz outputs by divided parameters on the individual ports. The above settings are shown in Figure 2-17

9	HDL Toggle Names Datasheet Rest
C_BANDWIDTH	OPTIMIZED
C_CLKFBOUT_MULT	8
C_CLKFBOUT_PHASE	0.000000
C_CLKIN1_PERIOD	10.000000
C_CLKOUT0_DIVIDE	2
C_CLKOUT0_DUTY_CYCLE	0.50000
C_CLKOUT0_PHASE	0.000000
C_CLKOUT1_DIVIDE	4
C_CLKOUT1_DUTY_CYCLE	0.50000
C_CLKOUT1_PHASE	0.00000
C_CLKOUT2_DIVIDE	8
C_CLKOUT2_DUTY_CYCLE	0.500000
C_CLKOUT2_PHASE	0.00000
C_CLKOUT3_DIVIDE	4
C_CLKOUT3_DUTY_CYCLE	0.500000
C_CLKOUT3_PHASE	0.00000
C_CLKOUT4_DIVIDE	4
C_CLKOUT4_DUTY_CYCLE	0.500000
C_CLKOUT4_PHASE	90.00000
C_CLKOUT5_DIVIDE	1
	OK Canc

Figure 2-17: **PLL Module Multiply Parameter**



XILINX®

l_module_0 : pll_module_v1_00_a	
9	HDL Toggle Names Datasheet Restore
C_BANDWIDTH	OPTIMIZED
C_CLKFBOUT_MULT	8
C_CLKFBOUT_PHASE	0.000000
C_CLKIN1_PERIOD	10.000000
C_CLKOUT0_DIVIDE	2
C_CLKOUT0_DUTY_CYCLE	0.500000
C_CLKOUT0_PHASE	0.000000
C_CLKOUT1_DIVIDE	4
C_CLKOUT1_DUTY_CYCLE	0.500000
C_CLKOUT1_PHASE	0.00000
C_CLKOUT2_DIVIDE	8 🗘
C_CLKOUT2_DUTY_CYCLE	0.500000
C_CLKOUT2_PHASE	0.00000
C_CLKOUT3_DIVIDE	4
C_CLKOUT3_DUTY_CYCLE	0.500000
C_CLKOUT3_PHASE	0.00000
C_CLKOUT4_DIVIDE	4 🗘
C_CLKOUT4_DUTY_CYCLE	0.500000
C_CLKOUT4_PHASE	90.000000
C_CLKOUT5_DIVIDE	1
	OK Cancel
	UG443_6_18_0

Figure 2-18: **PLL Module Divide/Phase Parameters**

Set C_CLKOUT0_DESKEW_ADJUST to **NONE**, C_CLKOUT1_DESKEW_ADJUST to **NONE**, C_CLKOUT2_DESKEW_ADJUST to **PPC**, C_CLKOUT3_DESKEW_ADJUST to **PPC**, and C_CLKOUT4_DESKEW_ADJUST to **PPC**. The above settings are shown in Figure 2-19.

All		HDL Toggle Names Datasheet Resto
C_		1
C_	REF_JITTER	0.100000
C_	RESET_ON_LOSS_OF_LOCK	false
C_	RST_DEASSERT_CLK	CLKIN1
C_	CLKOUT0_DESKEW_ADJUST	NONE
C_	CLKOUT1_DESKEW_ADJUST	NONE
C_	CLKOUT2_DESKEW_ADJUST	PPC
C_	CLKOUT3_DESKEW_ADJUST	PPC
C_	CLKOUT4_DESKEW_ADJUST	PPC
C_	CLKOUT5_DESKEW_ADJUST	PPC
C_	_CLKFBOUT_DESKEW_ADJUST	NONE
C_	CLKIN1_BUF	true
C_	_CLKFBOUT_BUF	true
C_	CLKOUT0_BUF	true
C_	CLKOUT1_BUF	true
C_	CLKOUT2_BUF	true
C_	CLKOUT3_BUF	true
C_	CLKOUT4_BUF	true
C_	CLKOUT5_BUF	false
C_	EXT_RESET_HIGH	0: Active Lc V

Figure 2-19: PLL Module Deskew Parameters



pll_module_0 : pll_module_v1_00_a	E
<u>A</u> I	HDL Toggle Names Datasheet Restore
C_DIVCLK_DIVIDE	1
C_REF_JITTER	0.100000
C_RESET_ON_LOSS_OF_LOCK	false
C_RST_DEASSERT_CLK	CLKIN1
C_CLKOUT0_DESKEW_ADJUST	NONE
C_CLKOUT1_DESKEW_ADJUST	NONE
C_CLKOUT2_DESKEW_ADJUST	PPC
C_CLKOUT3_DESKEW_ADJUST	PPC
C_CLKOUT4_DESKEW_ADJUST	PPC
C_CLKOUT5_DESKEW_ADJUST	PPC
C_CLKFBOUT_DESKEW_ADJUST	NONE
C_CLKIN1_BUF	true
C_CLKFBOUT_BUF	true
C_CLKOUT0_BUF	true
C_CLKOUT1_BUF	true
C_CLKOUT2_BUF	true
C_CLKOUT3_BUF	true
C_CLKOUT4_BUF	true
C_CLKOUT5_BUF	false
C_EXT_RESET_HIGH	0: Active Lcv
	<u></u>
	OK Cancel

Figure 2-20: PLL Module Buffer/Reset Parameters

XILINX[®]

Expand the pll_module_0 tree node in the System Assembly View/Port tab. Make the connections as shown in Table 2-1 and in Figure 2-21.

Port Name	Connection	I/O	Description
CLKFBIN	pll_clkfb_out	Ι	Feedback clock input.
CLKFBOUT	pll_clkfb_out	0	Feedback clock output.
CLKOUT0	proc_sys_clk	0	PowerPC 440 processor clock output.
CLKOUT1	interc_clk	0	Interconnect clock output.
CLKOUT2	sys_clk_s	О	Clock output for MPLB, SPLB0, SPLB1 and DCR.
CLKOUT3	mc_clk	0	MC clock output.
CLKOUT4	mc_clk_90	0	MC clock 90 output.
RST	sys_rst_s	Ι	Asynchronous reset signal.
LOCKED	pll_0_locked	0	An high output when PLL achieves phase alignment.
CLKIN1	dcm_clk_s	Ι	Primary clock input.

 Table 2-1:
 PLL Module Ports and Connections

•	Bus Interfaces	Ports	Addresses						1	♥ Filters (Applied)	🔧 Add External Port
Nar	ne		Net			Direction	Range	Class	Frequency	Reset Polarity	IP Type
÷.	External Ports										
	ppc440_virtex5_	0						PROCESSOR			ppc440_virtex5
	dcr_v29_0							BUS			dcr_v29
	> plb_v46_0							BUS			plb_v46
	> plb_v46_1							BUS			plb_v46
	> plb_v46_2							BUS			plb_v46
	xps_bram_if_cn.	t/r_1						PERIPHERAL			xps_bram_if_cntlr
	xps_bram_if_cn.	tlr_1_b						PERIPHERAL			bram_block
∎	🧼 ddr2_init							PERIPHERAL			xps_gpio
€								PERIPHERAL			xps_gpio
Ð	dcr_intc_0							PERIPHERAL			dcr_intc
.	<pre>> pll_module_0</pre>							PERIPHERAL			pll_module
	CLKOUTDCM5	5	No Connection	1	~	0					
	CLKOUTDCM4	4	No Connection	1	~	0					
	CLKOUTDCM3	3	No Connection	1	~	0					
	CLKOUTDCM2	2	No Connection	1	~	0					
	CLKOUTDCM1	1	No Connection	1	~	0					
	CLKOUTDCM)	No Connection	1	~	0					
	CLKOUT5		No Connection	1	~	0		CLK			
	CLKFBDCM		No Connection	1	~	0					
	LOCKED		pll_0_locked		~	0					
	CLKFBIN		pll_clkfb_out		~						
	CLKFBOUT		pll_clkfb_out		~	0					
	CLKOUT4		mc_clk_90		~	0		CLK			
	CLKOUT3		mc_clk		~	0		CLK			
	CLKOUT2		sys_clk_s		~	0		CLK			
	CLKOUT1		interc_clk		~	0		CLK			
	CLKOUTO		proc_sys_clk		~	0		CLK			
	RST		sys_rst_s		~	1					
	CLKIN1		dcm_clk_s		~			CLK			
Ð	PCI32_BRIDGE	F						PERIPHERAL			plbv46_pci
	pci_arbiter_0							PERIPHERAL			pci_arbiter
	xps_ll_temac_0							PERIPHERAL			xps_II_temac
	ppc440mc_ddr2	2_0						PERIPHERAL			ppc440mc_ddr2
	xps_uart16550_	0						PERIPHERAL			xps_uart16550
٠	itagppc_cntlr_0	_						PERIPHERAL			jtagppc_cntlr
	xps_central_dm	a_0						PERIPHERAL			xps_central_dma
	clock_generato.	r_0						PERIPHERAL			clock_generator
±	proc_sys_reset_	0						PERIPHERAL			proc_sys_reset
Sys	tem Assembly View										

UG443_6_21_091007





Expand the ppc440_0 tree node in the Port tab. Connect CPMPPCS1PLBCLK, CPMPPCS0PLBCLK, CPMDCRCLK, CPMPPCM to sys_clk_s. Connect the CPMINTERCONNECTCLK to interc_clk, CPMMCCLK to mc_clk and CPMC440CLK to proc_sys_clk.

The above connections are shown Figure 2-22.

ne		Net		Dire	ction	Range	Class	Frequency	Reset Polarity	IP Type
noc440 virtex5 (1						PROCESSOR			ppc440 virte
C440TRCTRIGG	SERE	No Connection		× 0		[0:13]	11100200011			pp0110_110
C440TRCTRIGG	SERE	No Connection				[0.10]				
CAAOTRCTRAC	EST	No Connection				10-61				
CAADTRCEVEC		No Connection				[0.0]				
CAANTROCVCLE	-	No Connection				[0.4]				
C440TDCDDAN	CUC	No Connection				10-21				
TPCCAADTPICC	CH3	No Connection				[0.2]				
TRCC440TRIGC	EDIC	No Connection		× 1						
DMA2DVIDO	EDI3	No Connection	1				INTERDURT			
DMA3KXIRQ		No Connection	1				INTERRUPT			
	CIN	No Connection	1	V U			INTERRUPT			
LLDMA3R5 I EN	GIN	No Connection	1	× 1			0117			
CPMDMA3LLCL	ĸ	No Connection	1	× I			CLK			
DMA2RXIRQ		No Connection	1 1	V 0			INTERRUPT			
DMA21XIRQ		No Connection	1 1	v 0			INTERRUPT			
LLDMA2RSTEN	GIN	No Connection	י ו	× I						
CPMDMA2LLCL	K	No Connection	1 1	× I			CLK			
DMA1RXIRQ		No Connection	י ^ו	v 0			INTERRUPT			
DMA1TXIRQ		No Connection	1 1	v 0			INTERRUPT			
LLDMA1RSTEN	GIN	No Connection	n (*	× 1						
CPMDMA1LLCL	K	No Connection	1 1	✓ 1			CLK			
LLDMAORSTEN	GIN	No Connection	1 ľ	× 1						
- PPCEICINTERC	ONN	No Connection	1 1	v 0			INTERRUPT			
EICC440CRITIR	Q	L to H: No Cor	nnection	× I			INTERRUPT			
SPLB1_Error		No Connection	1 i	v 0		[0:3]				
SPLB0_Error		No Connection	1 i	v 0		[0:3]				
C440DBGSYST	EMC	No Connection	1 i	v 0		[0:7]				
DBGC440UNCO	NDD	No Connection	1 i	¥						
DBGC440SYST	EMS	No Connection	1 ·	× I		[0:4]				
DBGC440DEBU	GHA	No Connection	י ו	× I						
PPCCPMINTER	CON	No Connection	י ו	v 0						
C440CPMWDIR	PTR	No Connection	1 ¹	v 0						
C440CPMTIMEF	RRE	No Connection	1	v 0						
C440CPMMSRE	E	No Connection	1	v 0						
C440CPMMSRC	E	No Connection	1	v 0						
C440CPMFITIRE	PTREQ	No Connection	1	v 0						
C440CPMDECIE	RPTR	No Connection	1	v 0						
C440CPMCORE	SLE	No Connection		× 0						
CPMC440TIME	RCLO	No Connection		V I			CLK			
CPMC440CORE	CLO	No Connection					0LIT			
CPMINTERCON	NEC	No Connection								
CPMINTERCON	NEC	No Connection								
CPMC440CLKET	NI NI	No Connection								
DMARRYIRO		No Connection					INTERRUPT			
DMAOTXING		No Connection					INTERRIPT			
EICCAMENTIDO		L to H: dor. int					INTERDURT			
CA40MACUINEC		No Compartie	c_v_iiq				INTERNOFT			
CHAUMACHINEC	RECK	No Connection	1	v U			CLK			
		II_CIK					CLK			
CPMINTERCON	INEC	Interc_clk					CLK			
CPMPPCS1PLB		SYS_CIK_S		× .			CLK			
CPMPPCS0PLB	ULK	sys_clk_s		×			CLK			
CPMMCCLK		mc_clk		× 1			CLK			
CPMDCRCLK		sys_clk_s	1	× I			CLK			
CPMPPCMPLBC	ιK	sys_clk_s	1	×			CLK			
CPMC440CLK		proc_sys_clk	1	× 1			CLK			

UG443_6_22_091007

Figure 2-22: PPC440 Clock Port Connections

Expand the tree node for ppc440_mc_ddr2_0 in the Port tab. Connect mc_mibclk to mc_clk and mi_mcclk90 to mc_clk_90.

The above connections are shown Figure 2-23.

•	Bus Interfaces Ports	Addresses						Filters (Applied)	👷 Add External Port
Name	•	Net		Direction	Range	Class	Frequency	Reset Polarity	IP Type
	External Ports								
	ppc440_virtex5_0					PROCESSOR			ppc440_virtex5
	> dcr_v29_0					BUS			dcr_v29
	plb_v46_0					BUS			plb_v46
	plb_v46_1					BUS			plb_v46
	plb_v46_2					BUS			plb_v46
	xps_bram_if_cntlr_1					PERIPHERAL			xps_bram_if_cntlr
	xps_bram_if_cntlr_1_b.					PERIPHERAL			bram_block
	> ddr2_init					PERIPHERAL			xps_gpio
	> LEDs_8Bit					PERIPHERAL			xps_gpio
	> dcr_intc_0					PERIPHERAL			dcr_intc
	> pll_module_0					PERIPHERAL			pll_module
	PCI32_BRIDGE					PERIPHERAL			plbv46_pci
	>pci_arbiter_0					PERIPHERAL			pci_arbiter
	<pre>xps_II_temac_0</pre>					PERIPHERAL			xps_II_temac
	ppc440mc_ddr2_0					PERIPHERAL			ppc440mc_ddr2
	mc_miwritereadytoac	No Connection	n	✓ 0					
	mc_miwillbebusy	No Connection	n	✓ 0					
	mc_mireaddataparity	No Connectio	n	✓ 0	[(C_DDR_DM_WIDTH*2)-1:0]				
	mc_mibusy	No Connection	n	✓ 0					
	mi_mcwritedataparity	No Connectio	n	×	[(C_DDR_DM_WIDTH*2)-1:0]				
	DDR2_CK_N	fpga_0_DDR	2_SDRAM_32Mx64_DDR_Clkn	✓ 0	[(C_NUM_CLK_PAIRS-1):0]				
	DDR2_CK	fpga_0_DDR	2_SDRAM_32Mx64_DDR_Clk	✓ 0	[(C_NUM_CLK_PAIRS-1):0]				
	DDR2_DM	fpga_0_DDR	2_SDRAM_32Mx64_DDR_DM	✓ 0	[(C_DDR_DM_WIDTH-1):0]				
	DDR2_CKE	fpga_0_DDR	2_SDRAM_32Mx64_DDR_CKE	✓ 0	[(C_NUM_RANKS_MEM-1):0]				
	DDR2_ODT	fpga_0_DDR	2_SDRAM_32Mx64_DDR_ODT	✓ 0	[(C_DDR2_ODT_WIDTH-1):0]				
	DDR2_CS_N	fpga_0_DDR	2_SDRAM_32Mx64_DDR_CSn	✓ 0	[(C_NUM_RANKS_MEM-1):0]				
	DDR2_WE_N	fpga_0_DDR	2_SDRAM_32Mx64_DDR_WEn	✓ 0					
	DDR2_CAS_N	fpga_0_DDR	2_SDRAM_32Mx64_DDR_CASn	✓ 0					
	DDR2_RAS_N	fpga_0_DDR	2_SDRAM_32Mx64_DDR_RASn	✓ 0					
	DDR2_BA	fpga_0_DDR	2_SDRAM_32Mx64_DDR_BankAddr	✓ 0	[(C_DDR_BAWIDTH-1):0]				
	DDR2_A	fpga_0_DDR	2_SDRAM_32Mx64_DDR_Addr	✓ 0	[(C_DDR_RAWIDTH-1):0]				
	DDR2_DQS_N	DDR2_DQS_	N	✓ IO	[(C_DDR_DQS_WIDTH-1):0]				
	DDR2_DQS	DDR2_DQS		✓ IO	[(C_DDR_DQS_WIDTH-1):0]				
	DDR2_DQ	DDR2_DQ		► IO	[(C_DDR_DWIDTH-1):0]				
	mi_mcclk_200	No Connectio	n	× 1					
	mi_mcreset	mc_rst		~		RST			
	mi_mcclk90	mc_clk_90		× 1					
	mc_mibclk	mc_clk							
	> xps_uart 16550_0					PERIPHERAL			xps_uart16550
	itagppc_cntlr_0					PERIPHERAL			jtagppc_cntlr
	<pre>xps_central_dma_0</pre>					PERIPHERAL			xps_central_dma
	clock_generator_0					PERIPHERAL			clock_generator
H <	<pre>proc_sys_reset_0</pre>					PERIPHERAL			proc_sys_reset
Syste	m Assembly View								

UG443_6_23_091007

Figure 2-23: PPC440MC DDR2 Clock Port Connections



Expand the tree nodes for plb_v46_0, plb_v46_1, and, plb_v46_2 in the System Assembly View/Port tab. Connect the PLB_Clk in all bus instances to sys_clk_s.

Expand the tree node for proc_sys_reset_0. Connect Slowest_sync_clk to sys_clk_s.

Expand the tree node for dcr_intc_0. Connect the DCR_Clk to sys_clk_s.

The above connections are shown in Figure 2-24.

	Poits Address	15					Filters (Applied)	Add Externa
ame	Net		Direction	Range	Class	Frequenc	y Reset Polarity	IP Type
 External Ports 								
> opc440_vintex5_	0				PROCESSOR			ppc440_virtex
- 🗢 dcr_v29_0					BUS			dcr_v29
> plb_v46_0					BUS			plb_v46
Bus_Error_Det	No Conne	ction	✓ 0		INTERRUPT			
SYS_Rst	plbv46_0	rst	✓ I		RST			
PLB_Clk	sys_clk_s		V I		CLK			
◇ plb_v46_1					BUS			plb_v46
Bus_Error_Det	No Conne	ction	✓ 0		INTERRUPT			
SYS_Rst	plbv46_1	rst	<u> </u>		RST			
PLB_Clk	sys_clk_s		V I		CLK			
plb_v46_2					BUS			plb_v46
Bus_Error_Det	No Conne	ction	<u> </u>		INTERRUPT			
SYS_Rst	plbv46_2	rst	× I		RST			
PLB_Clk	sys_clk_s		▼ 1		CLK			
xps_bram_if_cntli	<u>r</u> 1				PERIPHERAL			xps_bram_if_
xps_bram_if_cnth	<u>_1_b</u>				PERIPHERAL			bram_block
🗢 ddr2_init					PERIPHERAL			xps_gpio
✓ LEDs_8Bit					PERIPHERAL			xps_gpio
dcr_intc_0					PERIPHERAL			dcr_intc
Irq	No Conne	ction	✓ 0		INTERRUPT			
DCR_Rst	dcr_rst		× 1		RST			
DCR_Clk	sys_clk_s		✓ 1		CLK			
Intr	L to H: No	Connection	<u>I</u> I	[C_NUM_INTR_INPUTS-1:0]	INTERRUPT			
<pre>> pll_module_0</pre>					PERIPHERAL			pll_module
PCI32_BRIDGE					PERIPHERAL			plbv46_pci
· ∽ pci_arbiter_0					PERIPHERAL			pci_arbiter
<pre>>xps_ll_temac_0</pre>					PERIPHERAL			xps_II_temac
ppc440mc_ddr2_	_0				PERIPHERAL			ppc440mc_d
xps_uart16550_0	7				PERIPHERAL			xps_uart1655
itagppc_cntlr_0					PERIPHERAL			jtagppc_cntlr
xps_central_dma	0				PERIPHERAL			xps_central_c
clock_generator_	0				PERIPHERAL			clock_genera
proc_sys_reset_0)				PERIPHERAL			proc_sys_res
MB_Reset	No Conne	ction	✓ 0		RST			
MB_Debug_Sys	_Rst No Conne	ction	🕶 I		RST			
Aux_Reset_In	No Conne	ction	× 1		RST			
Peripheral_Rese	t dcr_rst&l	_rst	✓ 0	[0:C_NUM_PERP_RST-1]	RST			
- Dcm_locked	dcm_all_lo	icked	× 1					
Bus_Struct_Res	et plbv46_0	rst & plbv 46_1_rst & plbv 46_2_rst & mc_rst	V 0	[0:C_NUM_BUS_RST-1]	RST			
	lk sys_clk_s		V I		CLK			
Slowest_sync_c					DCT			

Figure 2-24: PLB v46 Instances Clock Port Connections

Configuring Clock Generator Inside The Migrated System

Add the clock generator by expanding the Clock Reset and Interrupt tree node inside the IP Catalog tab. Right click on **Clock Generator** and click on **Add IP.** This creates the clock_generator_0 instance inside the system.

Right click on **clock_generator_0** in the System Assembly View and select **Configure IP**

The inputs and outputs for the Clock Generator are set up in the Basic tab. Under the Ports, expand the Input & Feedback tree node. Click on **CLKIN**. On the right side of the window, select **Connect to: dcm_clk_s**. In addition, change the Frequency: to **100000000 Hz** as shown in Figure 2-25.

lock G	enerator	can generate required out	put clocks from given input reference	HDL Toggle Names Datasheet	Resto
Basic Step 1:	Ports Overview Specify input cloc	k details	ur system.	g needs. This tool will help you conligun	
Please Port	highlight a clock p s Input & Feedback CLKIN CLKFBIN Outputs Misc	ort in the list below and co Connected to dcm_clk_s	figure its requirements on the right Clock requirement: CLKIN Connected to: dcm_c	side. clk_s	
			Options	ieters in MHS file	
				OK Cancel <u>V</u> alidat	te Clo

Figure 2-25: Clock Generator Input & Feedback

Under Ports Overview, expand the Misc tree node which is near the bottom. Click on **RST**. On the right side of the window, select **Connect to: pll_0_locked**. Do not select **External reset is active high** because the reset of the clock module is in reset until the PLL module achieves phase alignment.

XILINX[®]

Click on **LOCKED**. On the right side of the window, select **Connect to: dcm_all_locked**. This output is connected to an input on the processor system reset module.

The above connections are shown in Figure 2-26.

Clock Ge	nerator		
The clock requiremen clock gene	generator module ts. It serves as a (rator module and	can generate required central clocking resour instantiate or update ir	HDL Toggle Names Datasheet Restore I output clocks from given input reference/feedback clock(s) based on your ce to meet all your system wide clocking needs. This tool will help you configure the n your system.
Basic	Ports Overview		
Step 1: Step 2: Step 2	Specify input cloc Specify the output ighlight a clock p	k details clock requirements ort in the list below and	configure its requirements on the right side.
	ignight a choole p		Clock requirement: RST
Ports		Connected to	
i lr i C	nput & Feedback Outputs Nisc		Connected to: pll_0_locked
	RST	pll_0_locked	External reset is active high
			Options Options Show low-level parameters in MHS file
			OK Cancel Validate Clock



www.xilinx.com

Under the Ports section expand the Outputs tree node and click on **CLKOUT0**. On the right of the window, set the Required frequency: to **100000000 Hz**, Required phase shift : to **0**, and Grouping Information: to **NONE**. Then select **Connect to: New connection...**.

In the dialog box displayed by the system, select **CPMDMAOLLCLK** under ppc440_virtex5_0.

The above connections are shown in Figure 2-27.

lease tupe	e in a net name for the clock generator's port: CLKOLITO	
lease type		
vet name:	clock_generator_u_CLROUTU	
You can als elect the p Componen	so connect the input clock ports of the components in your system to the clock gene ports below which you would like the clock generator's clock port to drive. nt Currently connected to:	erator's clock port. Pleas
	IO_virtex5_0 CPMC440CLK proc_sys_clk CPMINTERCONNECTCLK interc_clk CPMC440TIMERCLOCK CPMDCRCLK sys_clk_s CPMDMACLLCK mc_clk CPMDMA1LLCLK clock_generator_0_CLKOUT0 CPMDMA1LLCLK cPMDMA3LLCLK CPMDMA3LLCLK sys_reset_0 46_0 46_1 46_2 biter_0 _temac_0 odule_0	
-NOTE 1 - Only com like UAR	nponents that have clock ports which require their own clock requirements are listed Ts or GPIOs obtain their clocks from their buses, therefore, they are not listed above	above. Components
-NOTE 2- If a comp disconne	ponent is currently connected and you selected it to be connected with the clock ge acted from its current net.	nerator's port, it will be

Figure 2-27: Clock Generator CLKOUT0 Connections

www.xilinx.com



Under Ports section, expand the Outputs tree node and click on **CLKOUT1**. On the right side of the window, set the Required frequency: to **20000000 Hz**, Required phase shift : to **0**, and Grouping Information: to **NONE**, then select **Connect to: New connection...**.

Select **REF_CLK** under xps_ll_temac_0.

The above connections are shown in Figure 2-28.

lease type in a net name fo	or the clock generator's port: CLKOUT1
let name: clock generato	or 0 CLKOUT1
ou can also connect the ir	put clock ports of the components in your system to the clock generator's clock port. Plea
Component	Currently connected to:
 ppc440_virtex5_0 dcr_intc_0 proc_sys_reset_0 plb_v46_0 plb_v46_1 plb_v46_2 pci_arbiter_0 xps_ll_temac_0 GTX CLK 0 GTX CLK 0 Core_Clk Core_Clk UinkTemac1_ pll_module_0 	clock_generator_0_CLKOUT2 clock_generator_0_CLKOUT1 CLK CLK
NOTE 1 Only components that ha	ve clock ports which require their own clock requirements are listed above. Components
NOTE 1 Only components that have like UARTs or GPIOs obtain	ve clock ports which require their own clock requirements are listed above. Components ain their clocks from their buses, therefore, they are not listed above.
NOTE 1 Only components that have like UARTs or GPIOs obtain NOTE 2 If a component is currently disconnected from its cur	ve clock ports which require their own clock requirements are listed above. Components ain their clocks from their buses, therefore, they are not listed above. y connected and you selected it to be connected with the clock generator's port, it will be ent net.

Figure 2-28: Clock Generator CLKOUT1 Connections

Under the Ports section, expand the Outputs tree node and click on **CLKOUT2**. On the right of the window, set the Required frequency: to **125000000 Hz**, Required phase shift : to **0**, and Grouping Information: to **NONE**, then select **Connect to: New connection...**.

Select **GTX_CLK** under xps_ll_temac_0.

The above connections are shown in Figure 2-29.

Please type in a net name for the clock generator's port: CLKOUT2 Net name: clock_generator_0_CLKOUT2 You can also connect the input clock ports of the components in your system to the clock generator's clock port. Please select the ports below which you would like the clock generator's clock port to drive. Component Currently connected to: @:pop_d40_vites5_0 @:dor_intc_0 @:pop_d40_vites5_0 @:dor_intc_0 @:pob_v46_1 @:pob_v46_1 @:pob_v46_1 @:dor_intc_0 @:pob_v46_2 @:dor_intc_0 @:pob_v46_1 @:dock_generator_0_CLKOUT2 @:pob_v46_1 clock_generator_0_CLKOUT1 @:pob_v46_2 @:dock_generator_0_CLKOUT1 @:pob_v46_1 clock_generator_0_CLKOUT1 @:pot_v46_1 clock_generator_0_CLKOUT1 @:pot_v46_1 clock_generator_0_CLKOUT1 @:pot_v46_1 clock_generator_0_CLKOUT1 @:pot_v46_1 clock_generator_0_CLKOUT1 @:pot_v46_1 clock_generator_0_CLKOUT1 @:pot_v46_1 clock pots which require their own clock requirements are listed above. Components like UARTs or GPUS obtain their clocks from their buses, therefore, they are not listed above. MOTE 2 F a component is current net. OK	🗢 Make New Connection
Component Currently connected to:	Please type in a net name for the clock generator's port: CLKOUT2 Net name: clock_generator_0_CLKOUT2 You can also connect the input clock ports of the components in your system to the clock generator's clock port. Please select the ports below which you would like the clock generator's clock port to drive.
Ppc440_vitex5_0 Gor_intc_0 Proc_sys_reset_0 Ppb_v46_0 Ppb_v46_1 Ppb_v46_2 Poi_poi_v46_1 Poi_poi_v46_2 Poi_poi_v46_1 Doi_v60 Poi_v60 Doi_v60 Doi_v	Component Currently connected to:
NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UARTs or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net. OK Cancel	ppc440_virtex5_0 dor_intc_0 ppoc_sys_reset_0 pbb_v46_1 pbb_v46_1 pbb_v46_2 poi_arbiter_0 xps II temac 0
Only components that have clock ports which require their own clock requirements are listed above. Components like UARTs or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net. OK Cancel	NOTE 1
NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net. OK Cancel	Only components that have clock ports which require their own clock requirements are listed above. Components like UARTs or GPIOs obtain their clocks from their buses, therefore, they are not listed above.
OK Cancel	NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.
	OK Cancel

Figure 2-29: Clock Generator CLKOUT2 Connections

Migrating software from Virtex-4 PowerPC-405 to Virtex-5 PowerPC-440

The Virtex-5 processor block includes the PowerPC 440. The cross bar for the Virtex-5 is architecturally different from the Virtex-4 processor block, which included the PPC405 only. The Virtex-5 processor block keeps the software programming model changes to a minimum. In many cases the user does not have to change any code, but the Virtex-5 block offers new possibilities for system design.

Each of the following sections examines the migration requirements for one particular aspect of programming. For detailed information on using the new features of the PowerPC-440, see the *PPC440 CPU Core User's Manual*.

Instruction Set Changes

The instruction set changes primarily affect assembly language programmers. The instructions discussed here are dcba, dccci, dcread, dlmzb, eieio, icbt, icread, lswx, lswi, mfspr, mtspr, stwcx, sync, tlbia, tlbre, and tlbwe.

The PPC440 has a new, documented instruction called dlmzb that is useful in string operations. Although this instruction was present in some later versions of the PPC405, but was not documented. The GNU compilers supplied with EDK already use this instruction when appropriate.

The behavior of the lswx and lswi instructions is different for certain invalid instruction forms. In the PPC405, if the effective address register is in the range of registers to be written, it is skipped, while in the PPC440, it is overwritten. However, it is an error to specify the effective address register within the range of registers to be written (invalid instruction form).

The behavior of the mfspr and mtspr instructions is different for certain invalid instruction forms. In the PPC405, if the SPR address is not valid, the operation is undefined, but in the PPC440 the operation causes an invalid operation exception.

In the PPC405, the stwcx instruction causes an alignment exception if the effective address is not aligned on a word boundary, but an unaligned word address does not result in an exception in the PPC440. However, the normal usage of the stwcx instruction remains unchanged.

The sync and eieio instructions in the PPC405 are replaced by the msync and mbar instructions, respectively in the PPC440. The new instructions use the same opcodes as the old instructions so assembly code written for the PPC440 should use the new mnemonics, but the existing machine code containing these instructions will work unchanged. EDK users are insulated from these changes by the macros and functions in xio.h, and xio.c.

The icbt instruction has different opcodes in the PPC405 and PPC440 implementations but its behavior is the same. The dcba instruction in the PPC405 was intended to improve performance by providing hints to the hardware for initiating cache line fetches.

The dcba instruction is treated as a no-op in the PPC440 implementation. Any code that uses this instruction will continue to execute normally when moved from the PPC405 to the PPC440 but any performance advantages will be lost.

The operation of the dccci instruction is different between the PPC405 and PPC440 implementations. In the PPC405, dccci invalidates all data cache lines in a congruence class, and is intended to be invoked repeatedly to invalidate the entire cache during initialization. In the PPC440, dccci invalidates the entire cache, and is intended to be invoked once during initialization or power-on-reset.

The dcread and icread functions return data in a different format in the PPC440 but the instructions have the same meaning as in the PPC405.

The tlbia instruction in the PPC405 does not exist in the PPC440. This instruction invalidates all the TLB entries in the PPC405, and places the processor in real-address mode (as opposed to virtual-address mode). The PPC440 is always in virtual address mode so this instruction has been removed. The tlbre, tlbwe, and tlbsx instructions access different TLB registers in the PPC405 or TLB entry fields in the PPC440, but the meaning of the instructions remains the same.

Boot Vector

PPC440 starts at 0xFFFF_FFFC as does the PPC405. PPC440 has a longer booting sequence because of MMU initialization and configuration requirements. The compiler and startup files provided by Xilinx handle the boot sequencing automatically.

Interrupts

In the PPC405 in Virtex-4, the result of an exception is that control is transferred to an instruction at a fixed offset from the address stored in the Exception Vector Prefix Register (EVPR). For any given exception, the predefined 16-bit offset for that exception is concatenated with the 16-bit prefix stored in the EVPR to form the address of the interrupt handler. The PPC440 in Virtex-5 improves on this mechanism by providing 16 Interrupt Vector Offset Registers (IVORs) to hold the specific offset for each exception. When an exception occurs, the 16 bit prefix stored in the Interrupt Vector Prefix Register (IVPR) is concatenated with specific bits of the corresponding IVOR to generate the address of the interrupt handler. The IVORs allow each interrupt handler to be as large or as small as needed, optimizing memory requirements. More details are available in the IBM PPC440x5 CPU Core User's Manual, Chapter 6.

The default IVOR settings generated by EDK correspond to the Virtex-4 PPC405 offsets so user code does not have to change. However, users can rewrite the code to take advantage of the IVORs, if desired.

Caches

The PPC440 in Virtex-5 has 32KB instruction and data caches, while the PPC405 in Virtex-4 had 16KB instruction and data caches. The cache organization and replacement policies are different. While these differences could affect the performance of existing user code, user code changes are not required in most cases, especially if the user code relies on access functions such as XCache_EnableICache and XCache_EnableDCache that are provided by Xilinx. These functions enable/disable 128MB chunks of memory in Virtex-4, and 258 MB chunks of memory in Virtex-5.

Users who are moving their Virtex-4 PPC405 code to Virtex-5 PPC440 should ensure that any transitions between cached and uncached regions of the memory space fall on 256 MB boundaries.

Memory Management

The PPC405 supports two memory modes – a virtual mode where the processor works with virtual addresses and the MMU translates the virtual addresses into real addresses, and a real mode where the processor works with real addresses instead of virtual addresses. The PPC440 eliminates the real mode so the processor is always in virtual

address mode. However, the real mode can be emulated by configuring the MMU and TLB appropriately.

Users of the standalone Xilinx libraries and compilers automatically start with the emulated real mode on the Virtex-5 PPC440, which is similar to the default startup mode on the Virtex-4 PPC405. The RST command in XMD puts the processor in this mode. Memory management changes from the PPC405 to the PPC440 should be transparent to users of 3rd party RTOSes.

OCM

The PPC405 in Virtex-4 has two On Chip Memory (OCM) interfaces that allowed data and instruction memory regions to be accessed directly, bypassing the cache and PLB connections. The OCM interface was designed to provide low latency access to small amounts of local memory. The OCM interface is not available in PPC440 but the behavior of the OCM can be emulated in certain cases by locking a portion of the cache.

The Virtex-5 processor block has some additional interfaces such as the Memory Controller Interface, and the Auxiliary Processor Unit Interface, that support reads or writes to a given address. However, these interfaces are not designed to be OCM replacements.

The Memory Controller Interface is designed to allow the processor to access a single high speed memory controller, but various other devices could use the crossbar to share access to this high speed memory. While the OCM interface in Virtex-4 provides low latency access to memory, the crossbar can add some cycles of latency and arbitration delays to any access from the processor to a memory connected to the Memory Controller Interface.

The APU Interface allows the processor to transfer data from its memory (cached or notcached) or registers, to registers in a coprocessor module. This is very different from loading or storing external memory contents to/from a processor register.

Any system that was designed to use the OCM interface in Virtex-4 will therefore have to be redesigned to work without an OCM. Any memory block that was connected to the OCM interface in Virtex-4 would have to be connected to the Memory Controller Interface or to the MPLB interface in Virtex-5, along with any other memories that are required for the system.

DCR

The DCR register set for the Virtex-5 PPC440 implementation is different from the DCR register set for Virtex-4. One additional difference is that the Virtex-5 implementation provides a new indirect addressing mode, in addition to the direct addressing mode supported by the Virtex-4 implementation. The indirect addressing mode allows software to use less memory. Atomic DCR read/write operation is guaranteed by using the XIO_DCR driver.

Timers

The PPC440 implementation in Virtex-5 provides similar timer functions as the PPC405 implementation in Virtex-4. However the watchdog timer and fixed interval timer periods are different. These changes are transparent to users who access timer functions through Xilinx library functions or 3rd party RTOS functions.

APU

The APU interface and programming model are significantly different between Virtex-4 and Virtex-5. Users of the Xilinx FPU that connects to the APU interface must use a version that is compatible with the Virtex-5 APU.

Debugging

When the PPC440 operates in emulated real mode, code can be debugged using XMD to connect to the hardware, and GDB or SDK as the front end. Debugging in any other operating mode is not available through XMD for the Beta release, but Linux and Vxworks systems can be debugged with the help of third-party debuggers and hardware such as the BDI2000 debug module.







Chapter 3

PPC405 - PPC405/PLB v4.6 System *Migration*

Introduction

This chapter describes the system migration from a PPC405 PLB v3.4 system to a PPC405 PLB v4.6/XPS system. The migrated PPC405 PLB v4.6/XPS system uses the Multi-Ported Memory Controller (MPMC) core for main memories like SDRAM, DDR, and DDR2.

The following high-level migration steps are discussed:

- PPC405 wrapper with PLB v4.6 interface
- PLB v4.6 instances are added to the system where the OPB/PLB v3.4 and bridges are removed
- Clocking/Reset scheme
- Memory controllers are migrated to MPMC which is integrated into system
- Ethernet solution
- Equivalent XPS cores (from OPB/PLB v3.4) are connected to PLB v4.6

The original and migrated systems are built for the ML403 board.

System Migration Methodology

System Hardware Migration

The method recommended flow for doing the system hardware migration is outlined below.

- Migrate the processor from the PPC405 to the PPC405 with a PLB v4.6 interface
- Add PLB v4.6 instances for the system
- Migrate the PLB v3.4 or OPB memory controller to MPMC
- Migrate the clocking/reset scheme
- For peripherals, Master and Slaves cores are connected to the PLB v4.6
 - The maximum number of masters that can be connected to the SPLB is 16
 - The maximum number of slaves that can be connected to MPLB is 16

System Software Migration

No modifications to user application code are necessary when migrating from a PPC405 system to a PC405 PLB v4.6 system.

Existing PPC405 PLB v3.4 System

An existing PPC405 PLB v3.4 system is shown in Figure 3-1. This system will be used as a migration example since it contains common cores that are found inside a typical embedded system.



Figure 3-1: PPC405\PLBV34 System


PPC405 PLB v4.6/XPS System

The migrated PPC405 PLB v4.6/XPS system is shown in Figure 3-2.



Figure 3-2: Migrated PPC405 PLB v4.6 System

Overview of System Migration

Processor Migration

The system's PPC405 Virtex-4 (Wrapper) must use v2_00_a which supports the PLB v4.6 interface.

Added features of this wrapper are dual instruction-side and dual data-side PLB v4.6 interfaces.

The IPLB1 and DPLB1 bus interfaces are used for Point-to-Point connections to the PLB v4.6 PIMs on the MPMC.

The IPLB0 and DPLB0 bus interfaces are both connected to the PLB v4.6 shared bus. The shared bus is connected to a PLB v4.6 PIM port on the MPMC.

Note: The PLB v4.6 PIM port on the MPMC for the shared bus is only required when a master is connected to the shared bus and needs to access the MPMC address range.

A block diagram is shown in Figure 3-3.





Figure 3-3: PPC405 Bus Interfaces Block Diagram

Interconnect

The PLB v3.4 and OPB instances are deleted and the PLB v4.6 instance is added for the shared bus. In addition, the PLB2OPB and OPB2PLB bridges are removed. PLB v4.6 instances are added for the IPLB1 and DPLB1 and the shared buses IPLB0 and DPLB0.

XPS Peripherals

This section covers the general migration from an OPB/PLB v3.4 inside the system to a XPS core. This is applicable to slave and master/slave cores.

- 1. The former PLB v3.4 or OPB core is deleted.
- 2. The XPS core is added to the system.
- 3. Parameters and ports are connected in the same manner as the PLB v3.4 or OPB equivalent.
- 4. The slave or master/slave interfaces are connected to the PLB v4.6 shared bus instance.

Memory Controller Migration

The memory controller for SDRAM, DDR and DDR2 memories in a PLB v4.6 system is the MPMC.

The MPMC can support up to 8 configurable ports. Ports can be configured as XCL, PLB v4.6, SDMA (Soft Direct Memory Access) and NPI (Native Port Interface).

In the example migrated system block diagram, the MPMC is configured to support 4 ports and uses the PLB v4.6 and the SDMA (soft Direct Memory Access) Port Interface Modules (PIMs).

The PLB v4.6 PIMs are used for ports to the IPLB1 (Port 0) and DPLB1 (Port 1) on the PPC405 processor. The PLB v4.6 PIM contains a slave interface. Port 2 allows other masters connected to the PLB v4.6 shared bus to have access to the main memory.

Note: The PLB v4.6 PIM port on the MPMC for the shared bus is only required when a master is connected to the shared bus and needs to access the MPMC address range.

The SDMA PIM is connected to Port 3.

The SDMA PIM LocalLink interface is connected to the XPS LL TEMAC core to provide DMA accesses in and out of memory. The SDMA PIM is connected to the PLB v4.6 bus as a slave and to the XPS LL Temac through a LocalLink interface. The PLB v4.6 slave connection allows masters on the PLB v4.6 bus to setup and control DMA operations.

Ethernet Controller Migration

The migration of the PLB TEMAC to the XPS LL TEMAC controller varies depending on the features configured for the PLB TEMAC core.

If the PLB TEMAC core in the PPC405 PLB v3.4 system was configured to include the Scatter-gather DMA feature, the migrated system will require a DMA Controller with Scatter-gather DMA capability to be added. The SDMA PIM provides Scatter-gather DMA capability to and from the MPMC. The XPS LL TEMAC connects to this DMA controller via the LocalLink port.

If the PLB TEMAC core in the PPC405 PLB v3.4 system was configured to use FIFO direct mode, the migrated system will require the addition of the XPS LL FIFO core. The XPS LL TEMAC connects to the XPS LL FIFO via the LocalLink port. The FIFO data is then accessed through the PLB v4.6 connection of the XPS LL FIFO.

XPS LL TEMAC instantiates either a hard TEMAC or a soft TEMAC into the core. For a V4FX system, XPS LL TEMAC will instantiate the hard TEMAC. XPS LL TEMAC has two hard TEMACs available inside the core.

System Migration Inside an EDK System

Migrating Buses/Processor

Migrating the Bus

Adding/Removing Bus Instances

Remove the opb, plb and plb2opb instances (Delete instance and its internal ports) inside the System Assembly View/Bus Interfaces.

In addition, add 3 PLB v4.6 instances by expanding the Bus Bridge tree node. Right click on **Processor Local Bus (PLB) 4.6** and select **Add IP**. The buses are for the IPLB1, DPLB1, IPLB0 and DPLB0. The instances created are plbv46_0, plbv46_1, and plb_v46_2.

Migrating the Processor

Adding/Removing PPC405 Instances

Remove the former **ppc405_0** instance by right clicking on **ppc405_0** inside the **System Assembly View/Bus Interfaces** and clicking on **Delete Instance...** (Delete instance and its internal ports).

Add the latest version of the processor by expanding the Processor tree node inside the IP Catalog tab. Right click on **PowerPC 405 Virtex-4** and click on **Add IP**. This creates the ppc405_virtex4_0 instance.



Setting Parameters

Right click on ppc405_virtex4_0 inside the System Assembly View and select Configure IP \ldots .

Inside the PowerPC tab select the **Fastest PLB Clock** which is **IPLB1** as shown in Figure 3-4.

405_virt	tex4_0 : ppc4	05_virt	tex4_v2	_00_a		
					Тодд	HDL le Names Datasheet Re
PowerPC	Bus Settings	APU	Buses			
Address -	rPC™					
Internal	DCR Reg Base /	Addr	0b1	11111111	Internal DCR Reg High Addr	0ь000000000
X DF	PLB 1 Base Addr		C)xfffffff	DPLB 1 High Addr	0x00000000
X IP	LB 1 Base Addr		C	xfffffff	IPLB 1 High Addr	0x0000000
PPC405	Features					
C_PVR	LOW		C	рьоооо	C_PVR_HIGH	060000
Enable	the Memory-mana	agement l	Unit (MMU)		Deterministic Hardware Multiplier	r Mode
Disable	Operand Forward	ding for La	oad Instr.		Fastest PLB Clock	IPLB1 🗸
PLB Re	e-synchronization	Bypass				

Figure 3-4: Setting PPC405 Parameters

Setting Bus Interfaces

Inside the Bus Interfaces Tab, expand the ppc405_virtex4_0 instance. Select the following bus interfaces, JTAGPPC bus connection is connected to jtagppc_0_JTAGPPC0, IPLB1 bus is connected to plb_v46_0, DPLB1 bus is connected to plb_v46_1, IPLB0 and DPLB0 bus connections are plb_v46_2. The above selections are shown in Figure 3-5.



Figure 3-5: Setting PPC405 Bus Interfaces



Migrating The Memory Controller

Adding/Removing Memory Controllers

Remove the DDR_SDRAM_64Mx32 instance by right clicking on DDR_SDRAM_64Mx32 inside the System Assembly View/Bus Interfaces and clicking on Delete Instance... Then click on Delete instance and its internal ports.

Add MPMC to the system by expanding the Memory Controller tree node inside the IP Catalog tab. Right click on **Multi-Port Memory Controller** and click on **Add IP**. This creates the mpmc_0 instance.

The PLB DDR memory controller provided a big endian connection to memory devices whereas MPMC provides a little endian connection to memory devices. In turn the user needs to swap the pins to match the proper endian of the MPMC. In addition, it is preferred to use a MIG complaint pinout to obtain the best peformance with MPMC.

Refer to the MPMC documentation for IOSTANDARDS required for certain pins for the UCF which might be different from the previous memory controller.

Configuring The MPMC

Right click on mpmc_0 inside the System Assembly View and select Configure IP

Setting Ports

Inside the **Base Configuration** Tab, four ports will be used on the MPMC. Port 0 is set to the **Port Type** of **PLBV46**. Port 1 is set to the **Port Type** of **PLBV46**. Port 2 is set to the **Port Type** of **PLBV46**. Port 3 is set to the **Port Type** of **SDMA**.

In addition, under Common Addresses, the **Base Addr** is set to **0x00000000** which is the base address of the PLB DDR and the **SDMA Register Base Address** is set to **0x42000000** in this case. All ports of the MPMC are sharing the same base and high address.

The above configuration of the ports is shown in Figure 3-6.

							HDL Toggle Names	Datasheet R
se Configuration	Memory Interface Po	rt Configuration Ad	lvanced					
Port Type Configuration	n							
		MP	MC Modi	ule Interf	ace			
PORT0	PORT1	PORT2	PORT3	PORT4	PORT5	PORT6	PORT7	
1	•	•	1	≜	•	4	1	
PLBV46 V	PLBV46 V	PLBV46 V	SDMA V	INACTIVE V	INACTIVE V	INACTIVE V	INACTIVE V	LeftJustify
Base Addr			0x0000	SDMA R	egister Base Address			0x42000000
The pinout of	MPMC must be comp	patible with MIG (M	lemory Interface G	enerator), please s	ee data sheet of M	PMC for more infor	nation.	
se Configuration: sel	ect the port type for each	h active port and remo	ve unused ports betwe	en active ports				
emory Interface : sele	ect the memory part and o	configure the memory s	ettings					
vt Configuration: set t he Advanced tab con	the required port parame tains additional MPMC.c	ters configuration options ar	nd is intended for adva	nced users only				
		-						

Figure 3-6: MPMC Base Configuration Tab



Inside the **Memory Interface** Tab, selections are made for the DDR memory on the board.

For Memory Part Filter, **Type** is set to **DDR**, **Manufacture** is set to **Infineon**, and **Select a Part** is set to **HYB25D256160BT-7**. The Part Settings, Memory Timing Information and DIMM Settings sections are set automatically based upon the DDR selected inside the Memory Part Selector section.

For the Memory System Settings section, **Memory Clock Period** is set to **10000ps**, and **Memory Data Width** is set to **32**.

The above selections are shown Figure 3-7.

elected Memory Info Part No. HYB25D256160BT-7 Siz	te				
		64MB Type	DDR Base Addr	0x00000000 High Add	r Ox03FFFFF
emory/DIMM Settings Memory Part Se	ttings				
Settings	C	Configuration		Info	
Number of DIMMs	1	CE Width	1 ~	Memory DM Width	4 🗸
Memory Data Width	32 💙	X ODT Width	1.	Memory Addr Width	13
Memory Clock Period (ps)	10000 \$	Clock Width	1 👻	Memory Bank Addr Width	2
ODT Setting Dia	sabled 🗸	CSn Width	1. 🗸	Memory DQS Width	4 ~
Reduced Drive Output		No. of Ranks	1. ~		
		Registered Memory			
			_		

Figure 3-7: MPMC Memory Interface Tab

Inside the Port Configuration Tab, Port 3- SDMA, **MPMC to SDMA Clk Ratio** is set to **1** as shown in Figure 3-8.

se Configuration Memory Interface Port Configurati	on Advanced		
Port 0 - PLBv46		Port 1 - PLBv46	
Native Data Width of PLB	64 🗸	Native Data Width of PLB	64 🗸
Port 2 – PLBv46 Native Data Width of PLB	64 W	Pot 3 - SDMA Enable Completed Er on TX	
		Enable Completed Err on HX Clock Div. of Int. Timer Clk	100
		MPMC to SDMA Clk Ratio	1 💌

Figure 3-8: MPMC Port Configuration Tab

Inside the Advanced/Misc Tab, set the C_NUM_IDELAYCTRL to 2 and C_IDELAYCTRL_LOC to IDELAYCTRL_X0Y2-IDELAYCTRL_X0Y3. These settings are for the ML403 board.

MPMC Bus Interfaces

Ports 0,1, and 2 of the MPMC contains PLB v4.6 PIMs. The **SPLB0**, **SPLB1** and **SPLB2** bus interfaces are connected to the **plb_v46_0**, **plb_v46_1**, and **plb_v46_2** instances respectively as shown in Figure 3-9.

In addition, Port 3 is configured for SDMA. The **SDMA_CTRL3** bus interface is connected to **plb_v46_2** which is slave interface on the SDMA. This is shown in Figure 3-9.



Figure 3-9: MPMC PLBV46 PIM Bus Interfaces

MPMC Ports

The MPMC DDR ports are connected similar to the PLB DDR ports. The external ports for DQ and DQS external ports the Name and Net name must be the same and the external DDR feedback clock is removed. Discussion of connecting the system clocking needed for the memory controller is discussed later in this chapter.

In addition, the SDMA TX and RX interrupts should be connected to interrupt controller inside the system.

Migrating Ethernet Solution

Adding/Removing Ethernet

Remove the **TriMode_MAC_GMII** instance by right clicking on **TriMode_MAC_GMII** inside the **System Assembly View/Bus Interfaces** and clicking on **Delete Instance...** . Then click on **Delete instance and its internal ports**.

Add XPS LL TEMAC to the system by expanding the Communication High-Speed tree node inside the IP Catalog tab. Right click on **XPS LocalLink Tri-mode Ethernet Mac** and click on **Add IP**. This creates the xps_ll_temac_0 instance.

Setting Parameters

Right click on xps_ll_temac_0 inside the System Assembly View and select Configure IP ...

The XPS LL TEMAC is configured to use a single Hard TEMAC by unselecting **Enable TEMAC1.** The GMII interface is used by setting **Physical Interface Type** to **GMII**. Since the system is for VirtexTM-4, the **Type of TEMAC** is set to **V4 Hard TEMAC**. The above selections are shown in Figure 3-10.

User	System	Buses		
All			TYpe of TEMAC	V4 Hard TEM
			Include IO and BUFG as Needed for the PHY Interface Selected	
			Physical Interface Type	GMII 🗸
			Ratio of PLB Bus Clock to Core Clock	1
			PHY Address for TEMAC 0	00001
			TX FIFO Depth of TEMAC0	4096B 🗸
			Enable TX Checksum Offload for TEMAC0	
			RX FIFO Depth of TEMAC0	4096B 🗸
			Enable RX Checksum Offload for TEMAC0	
			Enable TEMAC 1	
			PHY Address for TEMAC 1	00010
			TX FIFO Depth of TEMAC1	4096B 🗸
			Enable TX Checksum Offload for TEMAC1	
			RX FIFO Depth of TEMAC1	4096B 🗸
			Enable RX Checksum Offload for TEMAC1	

Figure 3-10: XPS LL TEMAC Parameters

Setting Bus Interfaces

The slave interface of the XPS LL TEMAC core is connected to the plb_v46_2 instance inside the system, as shown in Figure 3-11.

Port 3 of the MPMC contains the SDMA. The LocalLink bus interface from the XPS LL TEMAC is connected to the SDMA LocalLink bus interface on the MPMC, as shown in Figure 3-11.



Figure 3-11: XPS LL TEMAC Bus and LocalLink Interfaces

XPS LL TEMAC Ports

The XPS LL TEMAC GMII ports are connected similar to PLB TEMAC GMII ports. Discussion of connecting the system clocking for the XPS LL TEMAC is discussed later in this chapter.



Migrating XPS Peripherals

Inside the migrated system, the PLB BRAM, OPB INTC, OPB GPIO, and OPB UART Lite cores are replaced with the equivalent XPS BRAM, XPS INTC, XPS GPIO, and XPS UART Lite, respectively.

Expand the tree node for ppc405_virtex4_0. The EICC405EXTINPUTIRQ input signal should be connected to the IRQ output signal on the interrupt controller inside the system.

Adding these cores are not discussed.

Modifying Clocking/Reset Inside The System

The processor system reset core is replaced with v2_00_a version of the core. In addition, the DCMs instantiations inside the system are replaced with the clock generator core.

Processor System Reset

Adding Processor System Reset to Migrated System

Inside the System Assembly View/Bus Interfaces, delete the reset_block instance (Delete instance and its internal ports).

Add the latest version of the processor system reset by expanding the Clock Reset and Interrupt tree node inside the IP Catalog tab. Right click on **Processor System Reset Module** and click on **Add IP**. This creates the proc_sys_reset_0 instance.

Right click on proc_sys_reset_0 inside the System Assembly View and select **Configure IP**

Set **External Active High** to **0**, Number of **Bus Structure Reset Registered Outputs** set to **4**. The above parameters are shown in Figure 3-12.



proc_s	ys_reset	_0 : proc_sys_reset_v2_00_a	
User	Buses	HDL Toggle Names Datasheet Resto	ne
All		Number of Clocks Before Input Change is Recognized On The External Reset Input 4 Number of Clocks Before Input Change is Recognized On The Auxiliary Reset Input 4 External Reset Active High 0 Auxiliary Reset Active High 1 Number of Bus Structure Reset Registered Outputs 4 Number of Peripheral Reset Registered Outputs 1	
		OK	

Figure 3-12: Processor System Reset Parameters

Inside the Bus Interfaces Tab, expand the ppc405_virtex4_0 tree node. Connect RESETPPC to proc_sys_reset_0_RESETPPC0 as shown in Figure 3-13.



Figure 3-13: PPC405 Reset Bus Interface

Inside the Ports Tab, expand the tree node for proc_sys_reset_0. Set Ext_Reset_in to sys_rst_s. Bus_Struct_Reset to plbv46_0_rst, plbv46_1_rst, plbv46_2_rst, and mpmc_rst. In addition, set Dcm_locked to dcm_all_locked which the input signal is created inside the clocking section.

The above port connections are shown in Figure 3-14.

•	Bus Interfaces	Ports	Addresses					Connection Filters	Add External Port
Nam	e		Net		Di	in Range	Class	Frequency Reset Polarity	IP Type
÷. <	External Ports								
÷ <	ppc405_virtex4_	0					PROCESSOR		ppc405_virtex4
E <	≥ plb_v46_0						BUS		plb_v46
÷ <	≥ plb_v46_1						BUS		plb_v46
E <	plb_v46_2						BUS		plb_v46
	>xps_bram_if_cn	t/r_1					PERIPHERAL		xps_bram_if_cntlr
	plb_bram_if_cnt	tlr_1_bran	,				PERIPHERAL		bram_block
	>mpmc_0						PERIPHERAL		mpmc
E <	≥jtagppc_0						PERIPHERAL		jtagppc_cntlr
E <	≥ LEDs_4Bit						PERIPHERAL		xps_gpio
E	>xps_intc_0						PERIPHERAL		xps_intc
	> RS232_Uart						PERIPHERAL		xps_uartlite
E	>xps_ll_temac_0	1					PERIPHERAL		xps_ll_temac
	proc_sys_reset_	_0					PERIPHERAL		proc_sys_reset
	Peripheral_Res	set	No C	onnection	✓ 0	[0:C_NUM_PERP_RST-1]	RST		
	MB_Reset		No C	onnection	∨ 0		RST		
	MB_Debug_Sy	/s_Rst	No C	onnection	V		RST		
	Aux_Reset_In		No C	onnection	¥		RST		
	Dcm_locked		dcm_	_all_locked	*				
	Bus_Struct_Re	set	plbv4	46_0_rst & plbv46_1_rst & plbv46_2_rst & mpmc_rst	∨ 0	[0:C_NUM_BUS_RST-1]	RST		
	Ext_Reset_In		sys_r	st_s	1	J	RST		
	Slowest_sync_	clk	clock	_generator_0_CLKOUT0	\mathbf{v}		CLK		
± <	clock_generato	r_0					PERIPHERAL		clock_generator
Syste	em Assembly View								

UG443_2_14_082007

XILINX°

Figure 3-14: Processor System Reset Connections

Expand the tree nodes for plb_v46_0, plb_v46_1, and, plb_v46_2 inside the System Assembly View/Port. Connect the **SYS_Rst** in all bus instances to **plbv46_0_rst**, **plbv46_1_rst**, and **plbv46_2_rst** respectively.

For master and slave peripherals connected to the PLB v4.6, the peripherals get vectorized resets for each master and slave connection from the bus core.

Expand the tree node for mpmc_0 and set the MPMC_Rst port to mpmc_rst.

Clock Generator

Overview of Existing Clocking Scheme

Originally, the system contains 2 cascading DCMs. The first DCM provides the 100 MHz clocks for the bus, processor clock, and the PLB DDR CLK0. In addition, the first DCM provides the 125 MHz clock for the Hard TEMAC.

The second DCM provides the 100 MHz phase shifted CLK90 for the PLB DDR memory controller.

With the PLB DDR clocks, inverters are needed for the CLK0 and CLK90 from the DCMs to provide the PLB DDR CLK180 and CLK270 clocks.

Clocking Requirements of Migrated System

The migrated system requires a 100 MHz clock for the processor, and buses. In addition, MPMC needs a 200 MHz clock for the IDELAY Controllers and a 100 MHz CLK0 and CLK90 clocks. No additional phase shift is needed for CLK90 since calibration for MPMC is handled by the IDELAY Controllers. Also, a 125 MHz clock is needed for the Hard TEMAC.

Configuring Clock Generator Inside The Migrated System

Inside the System Assembly View/Bus Interfaces, delete the dcm_0 and dcm_1 instances (Delete instance and its internal ports).

Add the clock generator by expanding the **Clock Reset and Interrupt** tree node inside the IP Catalog tab. Right click on **Clock Generator** and click on **Add IP**. This creates the clock_generator_0 instance inside the system.

Right click on clock_generator_0 inside the System Assembly View and select **Configure IP**

Inside the Basic tab is where the user sets up the inputs and outputs for the Clock Generator.



Under the Ports, expand the Input & Feedback tree node. Click on CLKIN. On the right side of the window, select **Connect to: dcm_clk_s**. In addition, change the **Frequency:** to **100000000 Hz** as shown in Figure 3-15.

			HDL Toggle Names Datasheet
ne clock rves as odule ar	generator modu a central clockin nd instantiate or u	le can generate required out g resource to meet all your s pdate in your system.	tput clocks from given input reference/feedback clock(s) based on your requirements system wide clocking needs. This tool will help you configure the clock generator
<u>B</u> asic	Ports Overview		
Step 1: Step 2: Please	Specify input clo Specify the outp highlight a clock	ck details ut clock requirements port in the list below and co	nfigure its requirements on the right side.
Port		Connected to	Clock requirement: CLKIN
• • •	LCLKFBIN Dutputs Misc		Frequency: 10000000 Hz
			Options Show low-level parameters in MHS file

UG443_2_15_082007

Figure 3-15: Clock Generator Input and Feedback

Under the Ports, expand the Misc tree node which is near the bottom. Click on **RST**. On the right side of the window, select **Connect to: sys_rst_s**. Don't select External reset is active high since the polarity on the reset is active low on the board.

Click on **LOCKED**. On the right side of the window, select **Connect to: dcm_all_locked**. This output is connected to an input on the processor system reset module.

The above connections are shown in Figure 3-16.

Clock G	enerator		
			HDL Toggle Names Datasheet Restore
The clock serves as module ar	c generator module a central clocking nd instantiate or up	e can generate required output presource to meet all your syst odate in your system.	clocks from given input reference/feedback clock(s) based on your requirements. em wide clocking needs. This tool will help you configure the clock generator
<u>B</u> asic	Ports Overview		
Step 1: Step 2:	Specify input cloo Specify the output	ck details It clock requirements	
Please	highlight a clock p	oort in the list below and config	ure its requirements on the right side.
Port	s	Connected to	Clock requirement: RST
	Input & Feedback Outputs Misc [RST LOCKED	sys_rst_s dcm_all_locked	Connected to: sys_rst_s
. tr			Options
			OK Cancel Validate Cloc

UG443_2_16_082007

Figure 3-16: Clock Generator Misc



Under the Ports expand the Outputs tree node and click on CLKOUT0. On the right of the window, set the **Required frequency:** to **100000000 Hz**, **Required phase shift :** to **0**, and **Grouping Information:** to **Group0.** Then select **Connect to: New connection...** .

This brings up a dialog box.

Select Slowest_sync_clk under proc_sys_reset_0.

Select PLB_Clk under plb_v46_0, plb_v46_1, and plb_v46_2.

Select CPMC405CLOCK under ppc405_virtex4_0.

Select **SDMA3_Clk** and **MPMC_Clk0** under **mpmc_0**. This sets the SDMA clock and CLK0 of the MPMC.

Select LlinkTemac0_CLK under xps_ll_temac_0 which selects the LocalLink clock.

The above connections are shown in Figure 3-17.

	Clon	
ease type in a net nar	ne for the clock generator's port: C	LKOUT0
et name: clock_gene	srator_0_CLKOUT0	
ou can also connect th orts below which you v	ne input clock ports of the compon would like the clock generator's clo	ents in your system to the clock generator's clock port. Please select the ck port to drive.
Component	Currently connected to	e
ppc405_virtex4_0		
CPMC405	CLOCK clock_generator_0_CL	KOUTO
	CLK	
⊒ plb_ <u>v46_0</u>		
PLB_Clk	clock_generator_0_CL	KOUTO
plb_v46_1		101170
PLB_Clk	clock_generator_0_CL	KOUTU
plb_v46_2		101170
PLB_Clk	clock_generator_0_CL	KUUTU
proc sys reset 0	and all adaption and a Ci	KOLITA
⊡ Slowest_s	ync_cik clock_generator_0_CL	NUTU
= mpmc_U	M-	
	лк. 11.	
	JK.	
		KOLITA
	ик clock_generator_U_CL	NUTU
	лк. Че	
	лК. Ча	
	11. 11.	
		KOLITA
	k90 clock generator 0 CL	KOUTI
	200MHz clock cenerator 0 CL	KOUT2
	<_zoominz clock_generator_0_CL	NO 012
- xos II temac 0	<	
	0 clock generator 0 CL	KOUT3
	clock generator 0 Cl	KOUT2
	0.00.1_90.00.001_0_00	
Core Clk		
✓ LinkTema	c0 CLK clock generator 0 CL	KOUTO
	c1 CLK	

Figure 3-17: Clock Generator CLKOUT0 Connections



Under the Ports expand the Outputs tree node and click on CLKOUT1. On the right of the window, set the **Required frequency:** to **100000000 Hz**, **Required phase shift :** to **90**, and **Grouping Information:** to **Group0.** Then select **Connect to: New connection...** .

Select **MPMC_Clk90** under **mpmc_0**. This provides the CLK90 of the MPMC.

The above connections are shown in Figure 3-18.

Make New Connection		
Please type in a net name for the	clock generator's port: CLKOUT1	
Net name: clock_generator_0_0	CLKOUT1	
You can also connect the input c ports below which you would like	lock ports of the components in y the clock generator's clock port t	our system to the clock generator's clock port. Please select the o drive.
Component	Currently connected to:	
CPMC405_virtex4_0	clock_generator_0_CLKOUT0	
□ PLB_Clk □ PLB_Clk	clock_generator_0_CLKOUT0	
□ PLB_Clk □ plb_v46_2	clock_generator_0_CLKOUT0	
PLB_Clk ⊫ proc_sys_reset_0	clock_generator_0_CLKOUT0	
Slowest_sync_clk	clock_generator_0_CLKOUT0	
SDMA0_Cik	clock_generator_0_CLKOUT0	
SDMA7_Ck	clock generator 0 CLKOUT0	
MPMC_Clk90	clock_generator_0_CLKOUT1	
MPMC_Clk_200MH	z clock_generator_0_CLKOUT2	
GTX_CLK_0	clock_generator_0_CLKOUT3 clock_generator_0_CLKOUT2	
UinkTemac0_CLK	clock_generator_0_CLKOUT0	
NOTE		
Only components that have clo GPIOs obtain their clocks from t	ck ports which require their own of their buses, therefore, they are no	clock requirements are listed above. Components like UARTs or t listed above.
NOTE 2 If a component is currently cont	nected and you selected it to be (connected with the clock generator's port, it will be disconnected
nom its current rict.		
	ОК	Cancel
		LIG443 2 18 082

Figure 3-18: Clock Generator CLKOUT1 Connections

Under the Ports expand the Outputs tree node and click on CLKOUT2. On the right of the window, set the **Required frequency:** to **200000000 Hz**, **Required phase shift :** to **0**, and **Grouping Information:** to **NONE.** Then select **Connect to: New connection...** .

Select **MPMC_Clk_200MHz** under **mpmc_0**. This provides the 200 MHz clock for the Idelay Controllers.

Select **REF_CLK** under **xps_ll_temac_0**.

The above connections are shown in Figure 3-19.

Make New Connection		×
Please type in a net name for the	clock generator's port: CLKOUT?	
Thease type in a net name for the	CLUCK generators point. CENOUTZ	·
Net name: clock_generator_0_	CLKOUT2	
You can also connect the input or ports below which you would like	lock ports of the components in y the clock generator's clock port t	our system to the clock generator's clock port. Please select the o drive.
Component	Currently connected to:	
ppc405_vitex4_0 CPMC405CLOCK CPMDCRCLK pplb v46.0	clock_generator_0_CLKOUT0	
PLB_Clk	clock_generator_0_CLKOUT0	
PLB_Clk	clock_generator_0_CLKOUT0	
PLB_Clk □ proc_sys_reset_0	clock_generator_0_CLKOUT0	
Slowest_sync_clk	clock_generator_0_CLKOUT0	
SDMA0_Clk	clock_generator_0_CLKOUT0	
SDMA4_Ck		
	clock_generator_U_CLKOUTU	
	clock generator 0 CLKOUT2	ן
		J
GTX CLK 0	clock_generator_0_CLKOUT3	
REFCLK	clock_generator_0_CLKOUT2	
Core_Clk		
Link Temac0_CLK	clock_generator_0_CLKOU10	
NOTE 1	ock ports which require their own (clock requirements are listed above. Components like UARTs or
GPIOs obtain their clocks from	their buses, therefore, they are no	t listed above.
If a component is currently con from its current net.	nected and you selected it to be	connected with the clock generator's port, it will be disconnected
)
	ОК	Cancel
		LIG443 2 19 0820

Figure 3-19: Clock Generator CLKOUT2 Connections



Under the Ports expand the Outputs tree node and click on CLKOUT3. On the right of the window, set the **Required frequency:** to **125000000 Hz**, **Required phase shift :** to **0**, and **Grouping Information:** to **NONE.** Then select **Connect to: New connection...** .

Select GTX_CLK under xps_ll_temac_0.

The above connections are shown in Figure 3-20.

Make New Connection		×						
Please type in a net name for the	clock generator's port: CLKOUT	3						
Net name: clock generator 0 (Net name: clock_generator_0_CLKOUT3							
You can also connect the input of ports below which you would like	lock ports of the components in y the clock generator's clock port	rour system to the clock generator's clock port. Please select the to drive.						
Component	Currently connected to:							
pc405_virtex4_0 CPMC405CLOCK CPMDCRCLK v46 0	clock_generator_0_CLKOUT0							
PLB_Clk □ PLB_Clk □ plb_v46_1	clock_generator_0_CLKOUT0							
PLB_Clk □ plb_v46_2	clock_generator_0_CLKOUT0							
PLB_Clk ⊜ proc_sys_reset_0	clock_generator_0_CLKOUT0							
Slowest_sync_clk	clock_generator_0_CLKOUT0							
SDMA0_Clk 	clock_generator_0_CLKOUT0							
MPMC Ck0	clock generator 0 CLKOUT0							
MPMC_Clk90	clock_generator_0_CLKOUT1							
MPMC_Clk_200MH	lz clock_generator_0_CLKOUT2							
GTX CIK 0	clock generator 0 CLKOUT3	1						
	clock_generator_0_CLKOUT2	-						
UinkTemac0_CLK	clock_generator_0_CLKOUT0							
NOTE 1								
Only components that have clo GPIOs obtain their clocks from	ock ports which require their own their buses, therefore, they are n	clock requirements are listed above. Components like UARTs or ot listed above.						
-NOTE 2								
If a component is currently con from its current net.	nected and you selected it to be	connected with the clock generator's port, it will be disconnected						
	UK	Laricel						
		UG443_2_20_08200						

Figure 3-20: Clock Generator CLKOUT3 Connections



Chapter 4

MicroBlaze System Migration

Introduction

This chapter describes the system migration from a MicroBlaze OPB system to a MicroBlaze PLB v4.6/XPS system. The migrated MicroBlaze PLB v4.6 system uses the Multi-Ported Memory Controller (MPMC) core for main memories like SDRAM, DDR and DDR2.

The following high-level migration steps are discussed:

- 1. MicroBlaze is configured for the PLB v4.6 interface
- 1. PLB v4.6 is added to the system where the OPB is removed
- 2. Clocking/Reset/Debug Scheme
- 3. Memory controllers are migrated to MPMC which is integrated into system
- 4. Ethernet Solution
- 5. Equivalent XPS cores (from OPB) are connected to PLB v4.6

The original and migrated systems are built for the ML505 board.

System Migration Methodology

System Hardware Migration

The method that is recommended for doing the system hardware migration is outlined below.

- Migrate the processor from MicroBlaze with an OPB interface to MicroBlaze with a PLB v4.6 interface
- Add PLB v4.6 instance to the system
- Migrate the OPB memory controller to MPMC
- Migrate the clocking/reset/debug scheme
- For peripherals, Master and Slaves cores are connected to PLB v4.6
 - The maximum number of masters that can be connected to the PLB v4.6 is 16
 - The maximum number of slaves that can be connected to the PLB v4.6 is 16

System Software Migration

No modifications to user application code are necessary when migrating from a MicroBlaze OPB system to a MicroBlaze PLB v4.6/XPS system.



An example MicroBlaze OPB system is shown in Figure 4-1. This system shows common cores that are found inside a typical embedded system.



Figure 4-1: MicroBlaze OPB System

Migrated MicroBlaze PLB v4.6/XPS System

The migrated MicroBlaze PLB v4.6/XPS system is shown in Figure 4-2.



Figure 4-2: MicroBlaze PLB v4.6/XPS System



Overview of System Migration

Processor Migration

The MicroBlaze processor version is replaced with to version 7.00.a which supports a PLB v4.6 interface. Both the IXCL and DXCL bus interfaces are connected to the MPMC through XCL Port Interface Modules (PIMs). MicroBlaze provides a master PLB v4.6 connections to the PLB v4.6 shared bus to allow access to the other peripherals on the shared bus.

Interconnect

The OPB instance is deleted and the PLB v4.6 instance is added inside the system.

XPS Peripherals

This section covers the general migration from an OPB to a XPS core. This is applicable to slave and master/slave cores.

- 1. The former OPB core is deleted.
- 2. The XPS core is added to the system.
- 3. Parameters and ports are connected in the same manner as the OPB equivalent.
- 4. The slave or master/slave interfaces are connected to the PLB v4.6 instance.

In addition, the plbv46_opb_bridge and the opb_plbv46_bridge bridges are available in case an OPB core must be added to the system.

Memory Controller Migration

The memory controller for SDRAM, DDR and DDR2 memories in a PLB v4.6 system is the Multi-Ported Memory Controller (MPMC).

MPMC can support up to 8 configurable ports. Ports can be configured as XCL, PLB v4.6, SDMA (Soft Direct Memory Access), and NPI (Native Port Interface).

In the example migrated system block diagram, the MPMC is configured to support 4 ports and uses the XCL, PLB v4.6 and the SDMA Port Interface Modules (PIMs).

The XCL PIMs on MPMC are for the IXCL (Port 0) and the DXCL (Port 1) bus interfaces from the MicroBlaze processor.

The PLB v4.6 PIM (Port 2) is connected to the PLB v4.6 shared bus inside the system. This allows other masters connected to the PLB v4.6 shared bus to have access to the main memory.

Note: The PLB v4.6 PIM for the shared bus is required if a master exists on the shared bus that needs to access the MPMC address range or if MicroBlaze is not setup for XCL transactions to the MPMC.

The SDMA PIM is connected to Port3.

The SDMA PIM LocalLink interface is connected to the XPS LL TEMAC core to provide DMA transactions in and out of memory. The SDMA PIM is connected to the PLB v4.6 bus as a slave and to the XPS LL Temac through a LocalLink interface. The PLB v4.6 slave connection allows masters on the PLB v4.6 bus to setup and control DMA operations.



Ethernet Controller Migration

The migration of the OPB Ethernet controller to the XPS LL TEMAC controller varies depending on the features configured for the OPB Ethernet core.

If the OPB Ethernet core in the MicroBlaze OPB system was configured to include the Scatter-gather DMA feature, the migrated system will require a DMA Controller with Scatter-gather DMA capability to be added. The SDMA PIM for the MPMC provides Scatter-gather DMA capability to and from the MPMC. The XPS LL TEMAC connects to this DMA controller via the LocalLink port.

If the OPB Ethernet core in the MicroBlaze - OPB system was configured to use FIFO direct mode, the migrated system will require the addition of the XPS LL FIFO core. The XPS LL TEMAC connects to the XPS LL FIFO via the LocalLink port. The FIFO data is then accessed through the PLB v4.6 connection of the XPS LL FIFO.

The XPS LL TEMAC could instantiate either a hard TEMAC or a soft TEMAC. Unlike OPB Ethernet, XPS LL TEMAC will instantiate/use the Hard TEMAC for V5LXT. XPS LL TEMAC has two hard TEMACs available inside the core but only one is used for this migration example.

System Migration Inside an EDK System

Migrating Buses/Processor

Migrating the Bus

Remove the mb_opb instance (Delete instance and its internal ports) inside the System Assembly View/Bus Interfaces.

In addition, add a PLB v4.6 instance by expanding the Bus Bridge tree node. Right click on **Processor Local Bus (PLB) 4.6** and select **Add IP**. The buses are for the PLB v4.6 shared bus inside the system. The instances created is plbv46_0.

Migrating the Processor

Adding/Removing MicroBlaze Instance

Remove the former MicroBlaze instance by right clicking on **microblaze_0** inside the **System Assembly View/Bus Interfaces** and clicking on **Delete Instance...** (Delete instance and its internal ports).

Add the latest version of the processor by expanding the Processor tree node inside the IP Catalog tab. Right click on **MicroBlaze** and click on **Add IP**. This creates the microblaze_0 instance.



Setting Parameters

Right click on microblaze_0 inside the System Assembly View and select Configure IP

Inside the Cache Tab, Enable Instruction Cache is enabled and Enable Data Cache is enabled. Size of the I-Cache in Bytes and Size of the I-Cache in Bytes are both set to 4kB. I-Cache Base Address, D-Cache Base Address and I-Cache High Address, D-Cache High Address are both set to 0x50000000 and 0x5FFFFFFF. The above settings are shown in Figure 4-3.

🔶 microbla	ze_0	: microblaz	e_v7_00	_a							×
							HC Toggle)L Names	Datashe	et Rest	bre
Instruct	ions	Exceptions	Cache	MMU	Debug	Interrupt and	Reset PV	R Bu	Jses		_
Enab	e Instr	uction Cache									
-Instr	iction	Cache Feature									
Siz	e of th	e I-Cache in By	es						4kB	~	
ЭI	I-Cache Base Address 0x5000000					I-Cache High	Address		0x5FFFFFF		
En	able X	linx Cache Link	s for I-Cach	e	V	Enable I-Cach	ne Writes				
Ins	tructor	n Cache Line Le	ngth	4	*	Number of I-C	ache Address	Tag Bits	16	*	
Enab	e Data	a Cache									
Data	Cach	e Feature									
Siz	e of D	Cache in Bytes							4kB	~	
D4	Cache	Base Address		0x5000	00000	D-Cache High	n Address		0x5FFFF	FFF	
En	able X	linx Cache Link	s for D-Cad	he	\checkmark	Enable D-Cad	che Writes				
Da	ta Cac	he Line Length		4	*	NumberofD- Bits	Cache Address	s Tag	16	×	
									ОК	Cance	

UG443_3_3_082007

Figure 4-3: Setting MicroBlaze Cache Tab

Inside the Debug Tab, select **Enable MicroBlaze Debug Module Interface**. This allows MicroBlaze to use debug. The MDM core will be discussed later in this chapter.

Inside the Buses Tab, **Select Processor Local Bus (PLB) interface** is **enabled** as shown in Figure 4-4.

🗢 microblaze_0 : microblaze_v7_00_a	×
HDL Toggle Names Datasheet Restore	
Instructions Exceptions Cache MMU Debug Interrupt and Reset PVR Buses	
Select Processor Local Bus (PLB) interface	
Number of FSL Links	
OK Cancel]





Setting Bus Interfaces

Inside the Bus Interfaces Tab, expand the microblaze_0 instance. Select the following bus interfaces, **IPLB** and **DPLB** bus connections are connected to **plb_v46_0**, **ILMB** is connected to **ilmb**, and DLMB is connected to **dlmb**. The DEBUG bus interface will be connected later in this chapter.

Note: LMB BRAM IF CNTLR must be version v2.10.a or higher for PLB v4.6 systems.

The above selections are shown in Figure 4-5.



Figure 4-5: Setting MicroBlaze Bus Interfaces

Migrating The Memory Controller

Adding/Removing Memory Controllers

Remove the DDR2_SDRAM_32Mx32 instance by right clicking on DDR2_SDRAM_32Mx32 inside the System Assembly View/Bus Interfaces and clicking on Delete Instance... Then click on Delete instance and its internal ports.

Add MPMC to the system by expanding the Memory Controller tree node inside the IP Catalog tab. Right click on **Multi-Port Memory Controller** and click on **Add IP**. This creates the mpmc_0 instance.

The MCH OPB DDR2 memory controller provided a big endian connection to memory devices whereas MPMC provides a little endian connection to memory devices. In turn the user needs to swap the pins to match the proper endian of the MPMC. In addition, it is preferred to use a MIG complaint pinout to obtain the best peformance with MPMC.

Refer to the MPMC documentation for IOSTANDARDS required for certain pins for the UCF which might be different from the previous memory controller.



Configuring The MPMC

Right click on mpmc_0 inside the System Assembly View and select Configure IP

Setting Ports

Inside the **Base Configuration** Tab, four ports will be used on the MPMC. Port 0 is set to the **Port Type** of **XCL**. Port 1 is set to the **Port Type** of **XCL**. Port 2 is set to the **Port Type** of **PLBV46**. Port 3 is set to the **Port Type** of **SDMA**.

In addition, under Common Addresses, the **Base Addr** is set to **0x50000000** which is the base address of the MCH OPB DDR2 in the existing system and the **SDMA Register Base Address** is set to **0x42000000** in this case. All ports of the MPMC are sharing the same base and high address.

The above configuration of the ports is shown in Figure 4-6.

nc_0:mpmc_v3	3_00_a							
							HDL Toggle Name	s Datasheet Re
Base Configuration	Memory Interface	Port Configuration Ad	vanced					
Port Type Configura	ation							
		MPI	NC Mod	ule Interf	ace			
PORTO	PORT1	PORT2	PORT3	PORT4	PORT5	PORT6	PORT7	
†	1	1	ł	↑	≜	≜	↑	
XCL V	XCL	PLBV46	SDMA 🗸	INACTIVE V	INACTIVE V	INACTIVE V	INACTIVE V	LeftJustify
-Common Addresses Base Addr	5		0x500	DOOOO SDMA R	egister Base Address			0x42000000
The pinout o	of MPMC must be c	ompatible with MIG (M	emory Interface G	enerator), please s	ee data sheet of M	PMC for more infon	nation.	
Base Configuration: s	select the port type for	each active port and remov	e unused ports betwe	een active ports				
Memory Interface : se	elect the memory part a	ind configure the memory s	ettings					
Port Configuration: se	et the required port par	ameters 10	d in the second s					
ine nuvanced tab ci	urkains auditional MPN	ic coniguration options an	u is inceriudo for adva	nceu users only				
								OK Can

UG443_3_6_082007

Figure 4-6: MPMC Base Configuration Tab
Inside the Memory Interface Tab, selections are made for the DDR memory on the board.

For Memory Part Filter, **Type** is set to **DDR2**, **Manufacture** is set to **Micron**, and **Select a Part** is set to **MT4HTF3264H-53E**. The Part Settings, Memory Timing Information and DIMM Settings sections are set automatically based upon the DDR2 selected inside the Memory Part Selector section.

For the Memory System Settings section, **Memory Clock Period** is set to **5000ps**, **Memory Data Width** is set to **64**, and **ODT Setting** is set to **75 Ohm**. In addition, set **ODT Width** to **2** and **Clock Width** to **2**. These settings are for a 64-bit DDR2 data width.

The above selections are shown Figure 4-7.

Memory Part Selector	Port Configuration Ad	Style * D	ensity * 🗸 Wid	th \star 💙 Part No. 🛙	T4HTF3264H-53E
Selected Memory Info Part No. MT4HTF3264H-53E	Size	256MB Type	DDR2 Base Addr	0x50000000 High Add	ir Ox5FFFFFF
Settings Number of DIMMs Memory Data Width Memory Clock Period (ps) ODT Setting 7 Reduced Drive Output	1 0 64 W 5000 0 5 Dhm W	Corfiguration	2 ¥ 2 ¥ 1 ¥ 1 ¥	Info Memory DM Width Memory Addr Width Memory Bank Addr Width Memory DQS Width	8 × 13 ÷ 2 ÷ 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5

Figure 4-7: MPMC Memory Interface Tab



🔰 XILINX®

In addition, Port 3 - SDMA, set the MPMC to SDMA Clk Ratio to 2.

mpmc_0 : mpmc_v3_00_a		
Base Configuration Memory Interface Port Configuration Advanced		HøL Toggle Names Datasheet Restore
Port 0-3 Port 4-7		
Port 0 - XCL	Port 1 XCL	
Cache Line Size 4	Cache Line Size	4 💌
Write Transfer	Write Transfer	1 💌
Port 2 - PLBv46	Port 3 SDMA	
Native Data Width of PLB 64	Enable Completed Err on TX	
	Enable Completed Err on RX	
	Clock Div. of Int. Timer Clk	100
	MPMC to SDMA Clk Ratio	2 🗸
		OK Cancel
		UG443 3 8 082007

Figure 4-8: **MPMC Port Configuration Tab**

Inside the Advanced/Misc Tab, set the C_NUM_IDELAYCTRL to 3 and C_IDELAYCTRL_LOC to IDELAYCTRL_X0Y5-IDELAYCTRL_X0Y1-IDELAYCTRL_X0Y0. These settings are for the ML505 board.

MPMC Bus Interfaces

Inside the Bus Interfaces Tab, expand the mpmc_0 instance.

XCL0 and XCL1 are connected to microblaze_0_IXCL and microblaze_0_DXCL bus interfaces on the MicroBlaze processor.

SPLB2 is connected to plb_v46_0.

SDMA_CTLR3 is connected to **plb_v46_0** which is the slave interface of the SDMA. The LocalLink interface is connected later in this chapter for the XPS LL Temac.

The connections are shown in Figure 4-9.



Figure 4-9: MPMC PLB v4.6 PIM Bus Interfaces

MPMC Ports

The MPMC DDR2 ports are connected similar to the MCH OPB DDR2 ports. The external ports for DQ and DQS external ports the Name and Net name must be the same. Discussion of connecting the system clocking needed for the memory controller is discussed later in this chapter.

In addition, the SDMA TX and RX interrupts should be connected to interrupt controller inside the system.



Migrating Ethernet Solution

Adding/Removing Ethernet

Remove the Ethernet_MAC instance by right clicking on **Ethernet_MAC** inside the **System Assembly View/Bus Interfaces** and clicking on **Delete Instance...**. Then click on **Delete instance and its internal ports**.

Add XPS LL TEMAC to the system by expanding the Communication High-Speed tree node inside the IP Catalog tab. Right click on **XPS LocalLink Tri-mode Ethernet Mac** and click on **Add IP**. This creates the xps_ll_temac_0 instance.

Setting Parameters

Right click on xps_ll_temac_0 inside the System Assembly View and select Configure IP ...

The XPS LL TEMAC is configured to use a single Hard TEMAC by unselecting **Enable TEMAC1**. The GMII interface is used by setting **Physical Interface Type** to **GMII**. The existing system used a MII interface since the OPB Ethernet didn't allow for the Hard TEMAC to be instantiated. **Ratio of PLB Bus Clock to Core Clock** is set to **1**. The above selections are shown in Figure 4-10.

er Susten Pusee	DL Names Datasheet Restore
Type of TEMAC	V5 Hard TEM V
Include IO and BUFG as Needed for the PHY Interface Selected	
Physical Interface Type	GMII 🗸
Ratio of PLB Bus Clock to Core Clock	1
PHY Address for TEMAC 0	00001
TX FIFO Depth of TEMAC0	4096B 🗸
Enable TX Checksum Offload for TEMAC0	
RX FIFO Depth of TEMAC0	4096B 💙
Enable RX Checksum Offload for TEMAC0	
Enable TEMAC 1	
PHY Address for TEMAC 1	00010
TX FIFO Depth of TEMAC1	4096B 💙
Enable TX Checksum Offload for TEMAC1	
RX FIFO Depth of TEMAC1	4096B 🗸
Enable RX Checksum Offload for TEMAC1	
- D.	
	OK Cancel

Figure 4-10: XPS LL TEMAC Parameters

Setting Bus Interfaces

The slave interface of the XPS LL TEMAC core is connected to the plb_v46_0 instance inside the system, as shown in Figure 4-10.

Port 3 of the MPMC contains the SDMA. The LocalLink bus interface from the XPS LL TEMAC is connected to the SDMA LocalLink bus interface on the MPMC, as shown in Figure 4-11.





XPS LL TEMAC Ports

The XPS LL TEMAC GMII ports are connected for GMII operation. Discussion of connecting the system clocking for the XPS LL TEMAC is discussed later in this chapter.



Migrating XPS Peripherals

In the migration example, the OPB INTC, OPB GPIO, and OPB UART Lite cores are replaced with the equivalent XPS INTC, XPS GPIO, and XPS UART Lite cores respectively.

Expand the tree node for **microblaze_0**. The Interrupt output port of the Interrupt Controller (IRQ) should be connected to the input Interrupt port on MicroBlaze.

Adding these cores are not discussed.

Modifying Clocking/Reset/Debug Inside The System

The processor system reset module is added inside the system. Also, the debug module is added to the system. In addition, the DCMs instantiations inside the system are replaced with the clock generator core.

Debug Module

Add the **MDM** to the system by expanding the Debug tree node inside the **IP Catalog** tab. Right click on **MicroBlaze Debug Module (MDM)** and click on **Add IP**. This creates the mdm_0 instance.

For MicroBlaze systems, the MDM core is used inside the system for debug.

Since the MDM core can be used on either PLB v4.6 or OPB systems, the parameters, bus connections, and ports are set to PLB v4.6.

Inside the **User/Debug** Tab, **C_MB_DBG_PORTS** is set to **1** since only one processor is used inside the system. The above parameter is shown in Figure 4-12.

💠 mdm_0 : mdm_v1_00_a			
User Buses		HDL Toggle Names D	atasheet Restore
Debug UART FSL Advanced	g ports		1
		ОК	Cancel
			UG443_3_12_082007

Figure 4-12: MDM Parameters User/Debug





XILINX[®]

mdm_0 : mdm_v1_00_	a	
User Buses		HDL Toggle Names Datasheet Restore
User Buses Debug UART FSL Advanced	Enable JTAG UART UART Data size Specifies the Bus Interface for the JTAG UART	8 ▶LBv46
		OK Cancel

Figure 4-13: MDM Parameters User/UART

The individual debug signals for MicroBlaze are grouped to the DEBUG bus interface. The MDM core has separate debug bus interfaces for each debug port. The **DEBUG** bus interface on MicroBlaze is connected to **mdm_0_MBDEBUG_0** on the MDM core as shown in Figure 4-14. In addition, the **SPLB** for the MDM core is connected to the **plb_v46_0** instance inside the system.



Figure 4-14: MDM Bus Interfaces

Since PLB v4.6 doesn't have a debug reset input like OPB, the MDM core has the Debug_SYS_Rst output port for system resets.

Expand the tree node for mdm_0 inside the System Assembly View/Port. Click on New Connection on the Net name for Debug_Sys_Rst. This port is connected to the Processor System Reset Module which is discussed inside the next section.



Processor System Reset

The Processor System Reset Module provides resets to the MicroBlaze system. The same reset scheme is used inside Base System Builder (BSB) for MicroBlaze PLB v4.6 systems.

In a MicroBlaze system, the Processor System Reset Module provides resets to the processor, buses, and peripherals. The inputs to the module is the external system reset, slowest synchronous clock, and the DCM Lock signal of the last DCM inside the system. Refer to the Processor System Reset Module Product Specification for more information.

Inside a MicroBlaze system, the **MB_Reset** of the MicroBlaze processor is connected to the **MB_Reset** output on the Processor System Reset Module.

In addition, the **Debug_SYS_Rst** output from the MDM core is connected to the **MB_Debug_Rst** on the Processor System Reset Module. This connection allows the processor and the buses to reset through the MDM by means of the Processor System Reset Module.

The Bus_Struct_Reset port is connected to the bus resets inside the system.

For master and slave peripherals connected to the PLB v4.6, the peripherals get vectorized resets for each master and slave connection from the bus core.

A block diagram of the reset scheme for the MicroBlaze processor, Processor System Reset Module and the MDM pcore is shown in Figure 4-15.

Note: The slowest synchronous clock, DCM Lock and bus resets connections inside the system are not shown inside the block diagram.



Figure 4-15: Reset Scheme Block Diagram

Adding Processor System Reset to Migrated System

Add the Processor System Reset by expanding the Clock Reset and Interrupt tree node inside the IP Catalog tab. Right click on Processor System Reset Module and click on Add **IP**. This creates the proc_sys_reset_0 instance.

Right click on proc_sys_reset_0 inside the System Assembly View and select Configure IP

Set External Active High to 0, Number of Bus Structure Reset Registered Outputs set to 2. The above parameters are shown in Figure 4-16.

proc_sys_reset_0 : pro	c_sys_reset_v2_00_a	×
User Buses	HDL Toggle Names	Datasheet Restore
	Number of Clocks Before Input Change is Recognized On The External Reset Input Number of Clocks Before Input Change is Recognized On The Auxiliary Reset Input External Reset Active High Auxiliary Reset Active High Number of Bus Structure Reset Registered Outputs Number of Peripheral Reset Registered Outputs	
		DK Cancel

UG443_3_16_082007

Figure 4-16: Processor System Reset Parameters

Inside the **Ports** Tab, expand the tree node for **proc_sys_reset_0**. **Bus_Struct_Reset** to **plbv46_0_rst**, and **mpmc_rst**. For **MB_Reset** set the Net name to **New Connection**. In addition, set **Dcm_locked** to **dcm_all_locked** which the signal is created inside the clocking section. **MB_Debug_Sys_Rst** is connected to **mdm_0_Debug_SYS_Rst** which is an out from the MDM core. Set **Ext_Reset_in** to **sys_rst_s**.

The following port connections are shown in Figure 4-17.

PROCESSOR microbiaze ■ initrobiaze_0 PROCESSOR microbiaze ■ initrobiaze_0 BUS Imb_v10 ■ initrobiaze BUS Imb_v10 ■ initrobiaze BUS Imb_v10 ■ offith BUS Imb_v10 ■ offith BUS Imb_v10 ■ offith BUS Imb_v10 ■ offith PERIPHERAL Imb_bram_fit ■ init_b_crdit PERIPHERAL Imb_bram_fit	ame	Net	Direction	Range	Class	Frequency Reset Pola	rity IP Type
PROCESSOR microblaze_0 Imb BUS Imb_v10 Imb PERIPHERAL Imb_bram_f_ Imb_cartin PERIPHERAL Imb_bram_f_ Imb_fram PERIPHERAL Imb_bram_block Imf_frame_0 PERIPHERAL Imb_bram_block Imf_frame_0 PERIPHERAL mgmc Imf_frame_0 PERIPHERAL mgmc Imf_frame_0 PERIPHERAL mps_intc Imf_frame_0 PERIPH							
b Imb BUS Imb_v10 b ofmb BUS Imb_v10 b ofmb_ordf BUS pb_v46 b ofmb_ordf PERIPHERAL Imb_bram_f b imb_ordf PERIPHERAL Imb_bram_f b imb_ordf PERIPHERAL Imb_bram_f b imb_ordf PERIPHERAL bram_block c md_0 PERIPHERAL bram_block c md_0 PERIPHERAL mdm c md_0 PERIPHERAL mdm c mgmc_0 PERIPHERAL mgmc c wps_intc_0 PERIPHERAL mgmc c wps_intc_0 PERIPHERAL wps_intc c wps_intc_0 PERIPHERAL </td <td>🔍 🦈 microblaze_0</td> <td></td> <td></td> <td></td> <td>PROCESSOR</td> <td></td> <td>microblaze</td>	🔍 🦈 microblaze_0				PROCESSOR		microblaze
b offic off	limb —				BUS		lmb_v10
e pb.y46_0 BUS pb.y46_0 e mb.y46_0 PERIPHERAL Imb.bram_ff	- 🗢 dlmb				BUS		lmb_v10
a dinb, cntri a dinb, cntri b linb, cntri cntri b linb, cntri cntr					BUS		plb_v46
Imb_craft PERIPHERAL Imb_brand PERIPHERAL bran_block Imb_brand PERIPHERAL proc_sist proc_sist proc_sist Imb_craft PERIPHERAL mdn mdn Imb_craft PERIPHERAL mdn mdn Imb_craft PERIPHERAL mdn mdn Imb_craft PERIPHERAL mdn mdn Imb_craft PERIPHERAL mdn monc Imb_craft PERIPHERAL mpnc monc Imb_craft PERIPHERAL mpnc mpnc Imb_craft PERIPHERAL mpnc mpnc Imb_craft PERIPHERAL mpnc mpnc Imb_craft PERIPHERAL mpnc mpnc Imb_craft Imb_craft PERIPHERAL mpc_sit Imb_craft Imb_craft PERIPHERAL mpc_sit Imb_craft Imb_craft Imb_craft mpc_sit Imb_craft Imb_craft Imb_craft mpc_sit Imb_craft Imb_craft Imb_craft mpc_sit Imb_craft Imb_craft Imb_craft Imb_craft Imb_craft Imb_craft Imb_craft Imb_craft Imb_craft Imb_craft Imb_cra	→ dlmb_cntlr				PERIPHERAL		Imb_bram_if_cnt
Image: Section of the section of t	. → ilmb_cntlr				PERIPHERAL		Imb_bram_if_cnt
• # 85322 PERIPHERAL xps_uatitie • mdm_0 PERIPHERAL mdm • mdm_0 PERIPHERAL mpmc • mdm_0 PERIPHERAL mpmc • mdm_0 PERIPHERAL xps_intc. • npmc_0 PERIPHERAL xps_intc. • xps_gnic_0 PERIPHERAL xps_gnic • xps_gnic_0 PERIPHERAL xps_gnic • xps_snest_0 PERIPHERAL xps_gnic • proc_sys_rest_0 PERIPHERAL xps_snest_0 • proc_sys_rest_0 PERIPHERAL xps_snest_0 • proc_sys_rest_0 Itemaccon PERIPHERAL xps_snest_0 • proc_sys_rest_0 Itemaccon PERIPHERAL xps_snest_0 • proc_sys_rest_0 Itemaccon PERIPHERAL xps_snest_0 • Source_sns_mo_ckcl cock_generator_0_CLKOUTO Itemaccon Itemaccon Itemaccon • Source_sns_mo_ckcl cock_generator_0_CLKOUTO Itemaccon RST Itemaccon Itemaccon • Bus_Struct_Reset • bus_vf6.0_rtk impmc_rst Itemaccon RST Itemaccon	- 🗢 lmb_bram				PERIPHERAL		bram_block
Implement Memory PERIPHERAL mdm Implement PERIPHERAL mpmc Implement PERIPHERAL mpmc Implement PERIPHERAL xps_intc Implement Peripheral Proc.sys_inst Implement Peripheral Peripheral Implement Peripheral Peripheral Implement Peripheral Peripheral	- 🗢 RS232				PERIPHERAL		xps_uartlite
⇒ npmc_0 PERIPHERAL mpmc > npmc_0 PERIPHERAL xps_pinc > npmc_0_0 PERIPHERAL xps_pinc > npmc_0_0 PERIPHERAL xps_pinc > npmc_sys_reset_0 PERIPHERAL xps_sitemac > npmc_sys_reset_0 PERIPHERAL xps_sitemac > npmc_sys_reset_0 PERIPHERAL xps_sitemac > Peripheral Reset No Connection MI RST > Doc_sys_reset_0 CLK Sitemac CLK > Sitewest_sync_ck clock_generator_0_CLXOUTO_MI CLK CLK > Sitewest_sync_sk mm_0_Debug_SYS_Rat	- 🗢 mdm_0				PERIPHERAL		mdm
> pag_inc_0 PERIPHERAL xps_intc > xps_golo_0 PERIPHERAL xps_golo > xps_golo_0 PERIPHERAL xps_glic > xps_glic_mac_0 PERIPHERAL xps_glic_mac_0 > xps_glic_mac_0 PERIPHERAL xps_glic_mac_0 > roc_sys_reset_0 PERIPHERAL proc_sys_rest > Petpheral.Reset No Connection Mil > Down_jocked dowl, generator_0_CLKOUTO NI > Slowet_sys_rest pb_v46_0_nt& mpmo_rst MO > Slowet_sys_rest_0 RST RST > Bay_sts_s mdm_0_Debug_SYS_Rt NI > MB_Reset_in proc_sys_rest_0 RST > dowl_generation 0 PERIPHERAL RST > dowl_generation 0 PERIPHERAL Association 0	> mpmc_0				PERIPHERAL		mpmc
PER/PHERAL xps_gpic_0 PER/PHERAL xps_gpic_0 > proc_sys_reset_0 PER/PHERAL xps_l_temac > proc_sys_reset_0 PER/PHERAL xps_l_temac Peripheral_Reset No Connection 0 [0.C_NUM_PERP_RST-1] RST - Aux_Reset_in No Connection 0 [0.C_NUM_PERP_RST-1] RST - Dom_jocked dom_all_ocked 0 . . . - Slowest_sync_ck clock_generator_0_CLKOUTO 0 . . . - Slowest_sync_ck clock_generator_0_CLKOUTO 0 - Bus_Struct_Reset pb_v46_0_rst & mpmo_rst MO - MB_Debug_Sys_Rat MI .					PERIPHERAL		xps_intc
a > pos_ /_temac_0 PERIPHERAL xps_ll_temac_0 a > pos_ sys_reset_0 PERIPHERAL pos_sys_reset_0 Peripheral Reset No Connection vi Aux_Reset_in No Connection vi Dom_locked dom_all_locked vi Stowet_sync_ck clock_generator_0_CLKOUTO will CLK Bus_Struct_Reset pb_v46_0_nt& mpmo_nt vi MB_Debug_Sys_Rat vi RST MB_Reset_in sys_rat_s vi MB_Reset_in proc_sys_reset_0_MB_Reset vi MB_Reset proc_sys_reset_0_MB_Reset vi MB_Reset proc_sys_reset_0_MB_Reset vi	- 🗢 xps_gpio_0				PERIPHERAL		xps_gpio
Proc_sys_reset_0 PERIPHERAL proc_sys_reset Propheral_Reset No Connection 0 [0.C_NUM_PERP_RST-1] RST Dom_locked dom_al_locked i i i Struct_Reset bl_v46_0_rst & mpmc_rst 0 [0.C_NUM_PERP_RST-1] RST Struct_Reset bl_v46_0_rst & mpmc_rst 0 [0.C_NUM_BUS_RST-1] RST Bus_Struct_Reset pb_v46_0_rst & mpmc_rst 0 [0.C_NUM_BUS_RST-1] RST Bus_Reset_in mm_o_lobug_SYS_Rst i RST MB_Reset_in proc_sys_reset_0_MB_Reset 0 RST MB_Reset_in proc_sys_reset_0_MB_Reset 0 RST					PERIPHERAL		xps_ll_temac
Pack Reset No Connection Image: Constraint of the system	proc_sys_reset_0				PERIPHERAL		proc_sys_reset
Aux_Reset_In No Connection MI RST Dcm_locked dcm_all_ocked MI CLK Stowet_sync_ck clock_generator_0_CLKOUTO MI CLK Bus_Struct_Reset pb_v46_0_rst & mpmc_rst MO MB_Debug_Sys_Rst mdm_0_Debug_SYS_Rst I MB_Reset pt_vst_s MI MB_Reset_In prc_sys_rest_0_MB_Reset NO MB_Reset pto_c_sys_rest_0_MB_Reset NO	Peripheral_Reset	No Connection	0	[0:C_NUM_PERP_RST-1]	RST		
Dom_jocked dom_all_jocked Image: Control of Control o	Aux_Reset_In	No Connection	1		RST		
Slovest_sync_ck clock_generator_0_CLKOUTO CLK Bus_Struct_Reset plb_v46_0_st & mpmc_st 0 [0.C_NUM_BUS_RST-1] RST MB_Debug_Sys_Fat I RST RST I RST MB_Reset_In sys_rst_s I RST I RST MB_Reset_for proc_sys_reset_0_MB_Reset I RST I	Dcm_locked	dcm_all_locked					
Bus_Struct_Reset pb_v46_0_rst&mpmc_rst MO [0.C_NUM_BUS_RST-1] RST MB_Debug_Sys_Rst mdm_0_Debug_SYS_Rst MI RST RST Ext_Reset_In sys_rst_s mod RST RST MB_Reset proc_sys_reset_0_MB_Reset NO PERIPHERAL Abok senset	Slowest_sync_clk	clock_generator_0_CLKOUT0	1		CLK		
MB_Debug_Sys_Rat mdm_0_Debug_SYS_Rat RST Ext_Reset_in sys_rst_s I MB_Reset proc_sys_reset_0_MB_Reset I RST RST	Bus_Struct_Reset	plb_v46_0_rst & mpmc_rst	0	[0:C_NUM_BUS_RST-1]	RST		
Ext_Reset_In sys_rst_s RST MB_Reset proc_sys_reset_0_MB_Reset O RST PERIPHERAL dock generative	MB_Debug_Sys_Rst	mdm_0_Debug_SYS_Rst			RST		
MB_Reset proc_sys_reset_0_MB_Reset ♥0 RST	Ext_Reset_In	sys_rst_s	1		RST		
PERIPHERAL dock gener	MB_Reset	proc_sys_reset_0_MB_Reset	· 0		RST		
	clock_generator_0				PERIPHERAL		clock_generator

UG443_3_17_082007

Figure 4-17: Processor System Reset Port Connections



Expand the tree node for **microblaze_0**, inside the **System Assembly View/Port**. Connect the **MB_RESET** to **proc_sys_reset_0_MB_Reset**.

Expand the tree node for plb_v46_0, inside the System Assembly View/Port. Connect the SYS_Rst to plbv46_0_rst.

Expand the tree node for mpmc_0 and set the MPMC_Rst port to mpmc_rst.

The above connections are shown in Figure 4-18.

Bus Interfaces	Ports Addresses			Connection Filt	ers 🚓 Add External Port Show All
Name	Net	Direction	Range	Class Freque	ncy Reset Polarity IP Type
🗄 🧼 External Ports					
🖃 🥯 microblaze_0				PROCESSOR	microblaze
- MB_Halted	No Connection	✓ 0			
- DBG_STOP	No Connection	✓ I			
INTERRUPT	L to H: No Connection	on 💌 I		INTERRUPT	
MB_RESET	proc_sys_reset_0_N	1B_Reset 💌 I		RST	
🕀 🧼 ilmb				BUS	lmb_v10
🗊 🧼 dlmb				BUS	lmb_v10
🖃 🧼 plb_v46_0				BUS	plb_v46
Bus_Error_Det	No Connection	✓ 0		INTERRUPT	
PLB_Clk	clock_generator_0_	CLKOUTO 🔽 I		CLK	
SYS_Rst	plb_v46_0_rst	▼ 1		RST	
🕀 🧼 dlmb_cntlr				PERIPHERAL	Imb_bram_if_cntlr
🕀 🧼 ilmb_cntlr				PERIPHERAL	lmb_bram_if_cntlr
🕀 🧼 Imb_bram				PERIPHERAL	bram_block
🕀 🧼 RS232				PERIPHERAL	xps_uartite
🕀 🧼 mdm_0				PERIPHERAL	mdm
□ mpmc_0				PERIPHERAL	mpmc
MPMC_Rst	mpmc_rst	× I		RST	
⊕ ∞ xps_intc_0				PERIPHERAL	xps_intc
🕀 🗢 xps_gpio_0				PERIPHERAL	xps_gpio
⊕ ∞ xps_ll_temac_0				PERIPHERAL	xps_ll_temac
Desc_sys_reset_0				PERIPHERAL	proc_sys_reset
∃ <> clock_generator_	0			PERIPHERAL	clock_generator
System Assembly View					
					UG443_3_18_08200

Figure 4-18: Processor System Reset System Connections



Clock Generator

Overview of Existing Clocking Scheme

Originally, the system contains 2 cascading DCMs. The first DCM provides the 100 MHz clocks for the bus, and processor clock. In addition, the first DCM provides the 200 MHz clock for the next DCM that provides the DDR2 clocking. In addition, the first DCM provides the 25 MHz for the Cal_Clk for the DDR2 memory controller.

The second DCM provides the 200 MHz CLK0 and CLK90 for the MCH OPB DDR2 memory controller.

With the MCH OPB DDR2 clocks, inverters are added by BSB inside the system which are deleted inside the migrated system.

Clocking Requirements of Migrated System

The migrated system requires a 100 MHz clock for the processor and bus. MPMC requires a 200 MHz clock for the IDELAY Controllers and a 200 MHz CLK0 and CLK90 clocks. For a GMII interface for XPS LL Temac, a 125 MHz clock is needed.

Configuring Clock Generator Inside The Migrated System

Inside the **System Assembly View/Bus Interfaces**, delete the **dcm_0** and **dcm_1** instances (Delete instance and its internal ports).

Add the clock generator by expanding the **Clock Reset and Interrupt** tree node inside the **IP Catalog** tab. Right click on **Clock Generator** and click on **Add IP**. This creates the clock_generator_0 instance inside the system.

Right click on **clock_generator_0** inside the **System Assembly View** and select **Configure IP**

Inside the **Basic** tab, this is where the user sets up the inputs and outputs to the Clock Generator.

Under the **Ports**, expand the **Input & Feedback** tree node. Click on **CLKIN**. On the right side of the window, select **Connect to: dcm_clk_s**. In addition, change the **Frequency:** to **100000000 Hz** as shown in Figure 4-19.

Clock Generator	
The clock generator module can generate re It serves as a central clocking resource to me module and instantiate or update in your syste	HDL Toggle Names Datasheet Restore quired output clocks from given input reference/feedback clock(s) based on your requirements. et all your system wide clocking needs. This tool will help you configure the clock generator rm.
Basic Ports Overview	
Step 1: Specify input clock details Step 2: Specify the output clock requirement	nts
Please highlight a clock port in the list belo	V and configure its requirements on the right side.
Ports Connected to	
CLKIN dcm_clk_s	Connected to: dcm_clk_s
CLKFBIN	
	Frequency: 10000000 Hz
RST sys_rst_s	
LOCKED dcm_all_locked	
	Options
	Show low-level parameters in MHS file
	OK Cancel <u>V</u> alidate Cloc
	LIC/1/2 2 10

Figure 4-19: Clock Generator Input and Feedback

Under the **Ports**, expand the **Misc** tree node which is near the bottom. Click on **RST**. On the right side of the window, select **Connect to: sys_rst_s**. Do not select **External reset is active high** since the polarity on the reset is active low on the board.

Click on **LOCKED**. On the right side of the window, select **Connect to: dcm_all_locked**. This output is connect to an input on the processor system reset.

The above connections are shown in Figure 4-20.

Ports Connected to Image: Ports Connected to <						Toggl	HDL Names		Restor
Basic Ports Pease highlight a clock details Step 1: Specify input clock details Step 2: Specify the output clock requirements Please highlight a clock port in the list below and configure its requirements on the right side. Ports Connected to Image: Image	erves as	generator module a central clocki	e can generate req ng resource to me	uired output cloc et all your system	ts from given input refere wide clocking needs. Thi	nce/feedback c s tool will help yo	lock(s) based	on your required	uirements
Step 1: Specify input clock details Step 2: Specify the output clock requirements Please highlight a clock port in the list below and configure its requirements on the right side. Ports Connected to Image:	tule and	Ports Overview	odate in your syste	n.					
Step 1: specify the output clock requirements Please highlight a clock port in the list below and configure its requirements on the right side. Ports Connected to Image: Ports Connected to: Image: Ports Sys_rst_s Image: Ports Connected to: Image: Ports Connected to: Image: Ports Connected to: Image: Ports Connected to: Image: Ports	Step 1: (ok dataile						
Please highlight a clock port in the list below and configure its requirements on the right side.	Step 2: 2	Specify input Cloc Specify the output	ut clock requirement	its					
Please highlight a clock port in the list below and configure its requirements on the right side. Ports Connected to Clock requirement: RST Connected to: sys_rst_s Cornected to: sys_rst_s External reset is active high Definition Options Show low-level parameters in MHS file									
Ports Connected to Imput & Feedback Connected to: sys_rst_s Imput & Feedback Imput & Feedback Imput & Feedback <td< td=""><td>'lease h</td><td>iighlight a clock p</td><td>port in the list below</td><td>v and configure it</td><td>clock requirements on the righ</td><td>t side. ст</td><td></td><td></td><td></td></td<>	'lease h	iighlight a clock p	port in the list below	v and configure it	clock requirements on the righ	t side. ст			
Input & Feedback Outputs Misc I.OCKED dcm_all_locked Options Show low-level parameters in MHS file	Ports		Connected to		Clock requirement: K	51			
Outputs Misc RST sys_rst_s LOCKED dcm_all_locked Options Show low-level parameters in MHS file	🕀 In	nput & Feedback			Connected to:	va rat a			~
Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of the system Image: State of	E 0	lutputs				<u></u>			
Options Show low-level parameters in MHS file	⊡ · M	lisc Det	eve ret e		External reset i	s active high			
Options		LOCKED	dom all looked			o doute night			
Options			uchi ali lockeu						
Options			dcm_ai_locked						
Options Show low-level parameters in MHS file			dcm_ail_locked						
Options Show low-level parameters in MHS file			dcm_ail_iockeu						
Options Show low-level parameters in MHS file			acm_an_tocked						
Options									
Options									
Show low-level parameters in MHS file			ucin_aii_lockedj						
<u>Snow low-level parameters in MHS file</u>			ucin_aii_lockedj		Ontions				
			ucin_aii_lockedj		Options		6.61		
					Options	arameters in MH	IS file		
					Options	arameters in MH	IS file		
					Options	arameters in MH	IS file		
					Options	arameters in MH	IS file		
					Options	arameters in MH	IS file		
					Options	arameters in MH	IS file		

Figure 4-20: Clock Generator Miscellaneous

Under the **Ports** expand the **Outputs** tree node and click on **CLKOUT0**. On the right of the window, set the **Required frequency:** to **100000000 Hz**, **Required phase shift :** to **0**, and **Grouping Information:** to **None.** Then select **Connect to: New connection...** .

This brings up a dialog box and select Slowest_sync_clk under proc_sys_reset_0.

Select PLB_Clk under plb_v46_0.

Select LMB_Clk for both ilmb and dlmb.

Select SDMA3_Clk under mpmc_0. This sets the SDMA clock.

Select LlinkTemac0_CLK under xps_ll_temac_0 which selects the LocalLink clock.

Select Slowest_sync_clk under proc_sys_reset_0.

The above connections are shown in Figure 4-21.

Please type in a	net name for the	clock generator's port: CLKOUT0	
Net name: cloc	k_generator_0_0	CLKOUTO	
You can also co ports below whic	nnect the input cl h you would like	ock ports of the components in yo the clock generator's clock port to	ur system to the clock generator's clock port. Please selec drive.
Component		Currently connected to:	
	B_Clk	clock_generator_0_CLKOUT0	
	B_Clk	clock_generator_0_CLKOUT0	
	B_Clk	clock_generator_0_CLKOUT0	
	MA0_Clk MA1_Clk MA2_Clk		
	MA3_Clk MA4_Clk MA5_Clk MA5_Clk MA5_Clk	clock_generator_0_CLKOUT0	
	MC_CIK0	clock_generator_0_CLKOUT1	
	MC_CIk_200MH MC_CIk_200MH	z clock_generator_0_CLKOUT1	
□ xps_ll_tema □ GT □ RE □ DC	c_O X_CLK_O FCLK LK	clock_generator_0_CLKOUT3 clock_generator_0_CLKOUT1	
. Uir	kTemac0_CLK kTemac1_CLK	clock_generator_0_CLKOUT0	
□ proc_sys_re	set_0 west_sync_clk	clock generator 0 CLKOUTO	
NOTE 1 Only compone or GPIOs obta	mest_sync_cik	cicck_generator_U_CLKOUIU	lock requirements are listed above. Components like UAR not listed above.
If a componer disconnected	t is currently conr from its current ne	nected and you selected it to be co et.	onnected with the clock generator's port, it will be

Figure 4-21: Clock Generator CLKOUT0 Connections



XILINX[®]

Select MPMC_Clk_200MHz and MPMC_Clk0 under mpmc_0.

Select **REF_CLK** under **xps_ll_temac_0**.

The above connections are shown in Figure 4-22.

ou can also connect the input	clock ports of the components in your system to the clock generator's clock port. Pleas	se select t
orts below which you would like	the clock generator's clock port to drive.	
ilmb		
umenter LMB_Clk	clock_generator_0_CLKOUT0	
□ LMB_Clk ⊫.plb_v46_0	clock_generator_0_CLKOUT0	
PLB_Clk	clock_generator_0_CLKOUT0	
- SDMA0_CK SDMA1_Ck SDMA2_Ck		
SDMA3_CIK	clock_generator_0_CLKOUT0	
SDMA6_Ck		
MPMC_CIK0	clock_generator_0_CLKOUT1 clock_generator_0_CLKOUT2	
MPMC_Clk_200M	<u>lz_clock_generator_0_CLKOUT1</u>	
□ xps_ll_temac_0	clock_generator_0_CLKOUT3_	
	clock_generator_0_CLKOUT1	
UinkTemac0_CLK	clock_generator_0_CLKOUT0	
⊡ proc_sys_reset_0 □ Slowest_sync_clk	clock_generator_0_CLKOUT0	
NOTE 1		
Only components that have of or GPIOs obtain their clocks fr	ock ports which require their own clock requirements are listed above. Components lik om their buses, therefore, they are not listed above.	ce UARTs
NOTE 2		
If a component is currently co disconnected from its current	nected and you selected it to be connected with the clock generator's port, it will be net.	

Figure 4-22: Clock Generator CLKOUT1 Connections

Under the **Ports** expand the **Outputs** tree node and click on **CLKOUT2**. On the right of the window, set the **Required frequency:** to **200000000 Hz**, **Required phase shift :** to **90**, and **Grouping Information:** to **Group0**. Then select **Connect to: New connection...** .

Select MPMC_Clk90 under mpmc_0.

The above connections are shown in Figure 4-23.

21 t t ft		
lease type in a net name for the	Clock generators port: CLKOUT2	
Net name: clock_generator_0_0	:LKOUT2	
You can also connect the input c ports below which you would like	ock ports of the components in your system to the clock generator's clock port. Please the clock generator's clock port to drive.	select th
Component	Currently connected to:	
ilmb □ LMB_Ck	clock_generator_0_CLKOUT0	
LMB_Clk	clock_generator_0_CLKOUT0	
PLB_Clk	clock_generator_0_CLKOUT0	
	clock_generator_0_CLKOUT0	
□ SDMA5_Clk □ SDMA6_Clk □ SDMA7_Clk <u>□ MPMC_Clk0</u>	clock_generator_0_CLKOUT1_	
MPMC_Ck90	clock_generator_0_CLKOUT2 z clock_generator_0_CLKOUT1	
	clock_generator_0_CLKOUT3 clock_generator_0_CLKOUT1	
UinkTemac0_CLK	clock_generator_0_CLKOUT0	
Slowest_sync_clk	clock_generator_0_CLKOUT0	
-NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UARTs		
or GPIOs obtain their clocks from their buses, therefore, they are not listed above.		
If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.		
	OK Cancel	

Figure 4-23: Clock Generator CLKOUT2 Connections



XILINX[®]

Select GTX_CLK under xps_ll_temac_0.

The above connections are shown in Figure 4-24.

Net name: dock_generator_0_CLKOUT3	lease type in a net name for the o	slock generator's port: CLKOUT3	
for can also connect the input clock pots of the components in your system to the clock generator's clock port. Please select toots below which you would like the clock generator's clock port to drive. Component Currently connected to: Imb Imb Implex46.0 Implex46.0 Implex46.0 Implex46.0 Implex46.0 Implex46.0 Implex46.0 SDIMA2.0k Implex10k Implex10k Implex	let name: clock_generator_0_C	LKOUT3	
Component Currently connected to: Imb LMB_Cik clock_generator_0_CLKOUT0 Imb Imb Imb Imb Imb clock_generator_0_CLKOUT0 Imple_v46_0 Imb Imb Imple_v46_0 clock_generator_0_CLKOUT0 Imple_v46_0 Imple_v46_0 Imple_v46_0 clock_generator_0_CLKOUT0 Imple_v46_0 Imple_v46_0 Imple_v46_0	'ou can also connect the input clo orts below which you would like t	ock ports of the components in your system to the clock generator's clock port. Please select the clock generator's clock port to drive.	
Imb IMB_CIk clock_generator_0_CLKOUT0 IMB_CIk clock_generator_0_CLKOUT0 Impro_0 SDMA0_CIk SDMA1_Cik SDMA2_Cik SDMA2	Component	Currently connected to:	
IMB_CIk dock_generator_0_CLKOUT0 PlB_CIk dock_generator_0_CLKOUT0 PLB_CIk dock_generator_0_CLKOUT0 SDMA0_CIk SDMA1_Cik SDMA2_Cik SDMA2_Cik SDMA2_Cik SDMA4_Cik SDMA4_Cik SDMA5_Cik SDMA5_Cik	ilmb └── □ LMB_Clk	clock_generator_0_CLKOUT0	
Impo_0 Impo_0 Impo_0 SDMA0_Clk SDMA2_Clk SDMA1_Clk Impo_0 SDMA2_Clk Impo_0 SDMA3_Clk Impo_0 SDMA3_Clk Impo_0 SDMA3_Clk Impo_0 SDMA3_Clk Impo_0 SDMA5_Clk Impo_0 SDMA5_Clk Impo_0 SDMA7_Clk Impo_0 SDKat_Clk <td>⊑ umb_ LMB_Clk ⊑∙plb_v46_0</td> <td>clock_generator_0_CLKOUT0</td>	⊑ umb_ LMB_Clk ⊑∙plb_v46_0	clock_generator_0_CLKOUT0	
SDMA3_Cik clock_generator_0_CLKOUT0 SDMA4_Cik SDMA5_Cik SDMA5_Cik SDMA5_Cik SDMA7_Cik MPMC_Cik.90 MPMC_Cik.90 clock_generator_0_CLKOUT1 MPMC_Cik.200MHz clock_generator_0_CLKOUT1 MPMC_Cik.200MHz clock_generator_0_CLKOUT1 MPMC_Cik_200MHz clock_generator_0_CLKOUT1 MPMC_Cik_200MHz clock_generator_0_CLKOUT1 MPMC_Cik_dem Core_Cik Unit termac_0 Core_Cik Unit termac_1CLK clock_generator_0_CLKOUT0 Unit Termac0_Cik clock_generator_0_CLKOUT0 Unit Termac0_Cik clock_generator_0_CLKOUT0 Unit Termac0_Cik clock_generator_0_CLKOUT0 Unit Termac1_Cik clock_generator_0_CLKOUT0 Stowest_sync_cik clock_generator_0_CLKOUT0 NOTE 1 Only components that have clock posts which require their own clock requirements are listed above. Components like UART or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	PLB_Clk 	clock_generator_0_CLKOUT0	
Image: MPMC_Ckk0 clock_generator_0_CLKOUT1 Image: MPMC_Ck2 MPMC_Ck2 Image: MPMC_Ck2 Image: MPMC_Ck2	SDMA3_CIK SDMA4_CIK SDMA5_CIK SDMA5_CIK SDMA6_CIK SDMA7_CIK	clock_generator_0_CLKOUT0	
Image: MPMC_Clk90 clock_generator_0_CLKOUT2 Image: MPMC_Clk_200MHz clock_generator_0_CLKOUT1 Image: MPMC_Clk_0 Image: MPMC_Clk_0 </td <td> MPMC_CIk0</td> <td>clock_generator_0_CLKOUT1</td>	MPMC_CIk0	clock_generator_0_CLKOUT1	
MPMC_Clk_200MHz clock_generator_0_CLKOUT1 MPMC_Clk_Mem Proc_sys_II temac_0 Exps_II temac_0 <tr< td=""><td>··· 🔲 MPMC_Clk90</td><td>clock_generator_0_CLKOUT2</td></tr<>	··· 🔲 MPMC_Clk90	clock_generator_0_CLKOUT2	
Aps II temac 0 GTX_CLK_0 clock_generator_0_CLKOUT3 REFCLK clock_generator_0_CLKOUT1 DCLK DCLK Ulink Temac0_CLK clock_generator_0_CLKOUT0 Ulink Temac1_CLK Josephine Clock_generator_0_CLKOUT0 Slowest_sync_clk clock_generator_0_CLKOUT0 NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UART or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	MPMC_Clk_200MHz	. clock_generator_0_CLKOUT1	
Image: STX_CLK_0 clock_generator_0_CLKOUT3 Image: REFCLK clock_generator_0_CLKOUT1 Image: DCLK clock_generator_0_CLKOUT0 NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UART or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is current y connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	xps_II_temac_0		
NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UART or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	GTX_CLK_0	clock_generator_0_CLKOUT3	
UlinkTemac0_CLK clock_generator_0_CLKOUT0 UlinkTemac1_CLK clock_generator_0_CLKOUT0 NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UART or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.		clock_generator_0_CLKOUT1	
NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UARTs or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	□ LlinkTemac0_CLK □ LlinkTemac1_CLK □ proc_sys_reset_0	clock_generator_0_CLKOUT0	
NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UART or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	Slowest_sync_clk	clock_generator_0_CLKOUT0	
or GPIOs obtain their clocks from their buses, therefore, they are not listed above. NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	NOTE 1 Only components that have clock ports which require their own clock requirements are listed above. Components like UARTs		
NOTE 2 If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.	or GPIUs obtain their clocks from their buses, therefore, they are not listed above.		
	If a component is currently connected and you selected it to be connected with the clock generator's port, it will be disconnected from its current net.		

Figure 4-24: Clock Generator CLKOUT3 Connections



Chapter 5

Migration of User IP Slave Cores

Introduction

This chapter describes the process for migrating from an existing OPB or PLB v3.4 slave core to a PLB v4.6 slave core and retain the same services and functionality.

The following migration steps will be discussed:

- 1. Review the Create IP Wizard slave services available for the OPB/PLB v3.4 slave core
- 2. Review the Create IP Wizard slave services available for the PLB v4.6 slave core
- 3. Describe services and functionality of the original OPB/PLB v3.4 core created by Create IP Wizard
- 4. Use Create IP Wizard, within the EDK tools, to create an equivalent PLB v4.6 pcore for both PLB v3.4 and OPB slave cores
 - a. Set up services to match the functionality of the original OPB/PLB v3.4 slave core
 - b. Describe parameters dealing with PLB v4.6 slave pcores
- 5. Modify the existing user logic of the pcore to interface with the top level template created by Create IP Wizard

Overview of Create IP Wizard for OPB/PLBV34 Slaves

Create IP Wizard Slave Services

The slave services for OPB/PLBV34 are S/W reset and MIR, User Logic Interrupt Support, User Logic S/W register support, Burst Transaction Support, FIFO, and User Logic Address range support.

For OPB, the slave interface has a data width of 32 bits. For PLBV34, the slave interface has a data width of 64 bits.

OPB and PLBV34 IPIFs

The OPB and PLBV34 IPIFs includes slave services and master services which are enabled or disabled through parameters inside the IPIF instance. The Create IP Wizard enables or disables these parameter based upon user selections inside the Wizard.



Overview of Create IP Wizard using PLB v4.6

Create IP Wizard Slave Services

The slave services are similar to the OPB and PLBV34 slave services. The services are Software Reset, Read/Write FIFO, Interrupt Control, User logic Software Register, and User Logic Memory Space. The user has a option of adding burst and cache-line support inside the slave. Burst and native data width are discussed later in this chapter.

PLBV46 Slave IPIFs

Neither the PLBV46 Slave Single IPIF nor the PLBV46 Slave Burst IPIF support a Soft Reset service, Local IP Interrupt services, RdFIFO services, WrFIFO services or master services. These services are automatically added through the Create IP Wizard in the top level template file created by the Wizard.

PLBV46 Slave IPIFs do not support aborts or indeterminate length bursts. However, optimizations may be possible inside the user logic based upon the Bus2IP_BurstLength signal inside the IPIC.

In the PLBV46 Slave Burst IPIF, when the write buffer is included, the IP2Bus_Error signal has no affect on the bus. This is because the PLBV46 IPIF will acknowledge the data phase on the PLB immediately after the address phase, therefore not allowing the IP time to assert the IP2Bus_Error signal. To catch an IP error during a write transaction, the user must set an interrupt .

The selection of the PLBV46 Slave IPIF depends on the selection of Burst and cache-line support in the Wizard. If Burst and cache-line support is not enabled in the wizard, the PLBV46 Slave Single IPIF is used in the slave core. If Burst and cache-line support is enabled, the PLBV46 Slave Burst IPIF is used in the slave core.

Overview of OPB/PLBV34 Slave Cores Created by Create IP Wizard

The PLBV34 and OPB Slave Example pcores were created with Create IP Wizard, within the EDK tools. Both slave examples use the software reset and MIR support, user logic interrupt support with Device ISC (interrupt source controller) is enabled, user logic Software register support, RdFIFO and WrFIFO with Packet Mode and Vacancy Calculator, and burst transaction support. The setup of these services are the same for the OPB and PLBV34.

For OPB, the write mode to this core disables posted write behavior for normal acknowledge write behavior.

For OPB, four 32-bit registers are used in the user logic.

For PLBV34, four 64-bit registers are used in the user logic.

Creating PLBV46 Slave Cores with Create IP Wizard

Generating The PLBV46 Slave Cores Using Create IP Wizard

The steps in the following section are applied in generating both the migrated OPB and PLBV34 slave cores. Notice the migration considerations for both OPB/PLBV34 in the following steps.

1. Invoke the Create IP Wizard and click on **Next** as shown in Figure 5-1.



Figure 5-1: Create/IP Wizard Welcome



2. In the Create and Import Peripheral Wizard - Peripheral Flow window, under Select Flow, click on **Create templates for a new peripheral** as shown in Figure 5-2, hen click on **Next**.

🗢 Create and Import Peripheral Wizard - Peripheral Flow	
Peripheral How 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 compliant peripheral, or help you import an existing peripheral into an XPS project or ED The interface files and directory structures required by EDK will be generated. Select flow Create Templates Create Templates Timplement/Verify Import to XPS Create templates for a new peripheral Create templates that have the EDK compliant pot/parameter interview Will need to implement the body of the peripheral.	K repository.
More Info	Cancel

UG443_4_2_082007

Figure 5-2: Create/IP Wizard Peripheral Flow

3. In the Create Peripheral - Repository or Project window, select a location for the repository where the cores will be generated as shown in Figure 5-3, then click on **Next**.

🗢 Create Peripheral - Repository or Project	
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 an EDK repository, the peripheral can be acces XPS projects.	sed by multiple
To an EDK user repository (Any directory outside of your EDK installation path)	
Repository: C:\edk_user_repository	Bro <u>w</u> se
○ To an XPS project Project: ♥	Browse
Peripheral will be placed under:	
C:\edk_user_repository\MyProcessorIPLib\pcores	
More Info	Cancel

UG443_4_3_082007

Figure 5-3: Create/IP Wizard Repository



4. In the Create Peripheral - Name and Version window, enter the name of the slave migrated core that will be created as shown in Figure 5-4 for the OPB migration, then click on **Next**.

💠 Create Peripheral - Name and Version	
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: xps_opb_master_example	
Version: 1.00.a	
Major revision: Minor revision: Hardware/Software compatibility revision: 1 00 a	
Description:	
Logical library name:xps_opb_master_example_v1_00_a All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical librar above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDP repositories indicated in the XPS project settings.	y name (
More Info	Cancel

UG443_5_4_082007

Figure 5-4: Create/IP Wizard Name and Version for OPB MIgration

5. In the Create Peripheral - Name and Version window, enter the name of the slave migrated core that will be created for the PLB migration example as shown in Figure 5-5, then click on **Next**.

🗢 Create Peripheral - Name and Version	
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: xps_plb_master_example	
Version: 1.00.a	
Major revision: Minor revision: Hardware/Software compatibility revision: 1 00 a	
Description:	
Logical library name: xps_plb_master_example_v1_00_a	
All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.	name
More Info	Cancel

UG443_4_5_082007

Figure 5-5: Create/IP Wizard Name and Version for PLB Migration



6. In the Create Peripheral - Bus Interace window, under under **To which bus will this peripheral be attached?**, click on **Processor Local Bus (PLB v4.6)** as shown in Figure 5-6, then click on **Next**.

🗢 Create Peripheral - Bus Interface	
Bus Interface Indicate the bus interface supported by your peripheral.	
To which bus will this peripheral be attached? Image: Constraint of the second state of the sec	
ATTENTION Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) buses (include interconnect and OPB/PLB v3.4 interconnect) and the FSL interface. NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface. <u>CoreConnect Specification</u> <u>PLB (v4.6) Slave IPIF Specification for single data beat transfer</u> <u>PLB (v4.6) Slave IPIF Specification for burst data transfer</u> <u>PLB (v4.6) Master IPIF Specification for single data beat transfer</u> <u>PLB (v4.6) Master IPIF Specification for burst data transfer</u> <u>PLB (v4.6) Master IPIF Specification for burst data transfer</u>	ling PLB v4.6
Note Xilinx recommends using the new PLB v4.6 bus standard, however, the wizard still supports the OPB and PLB v3.4 bus interfaces. Enable OPB and PLB v3.4 bus interfaces	
More Info	Cancel

Figure 5-6: Create/IP Wizard Bus Interface

7. In the Create Peripheral - IPIF (IP Interface) Services window, select the services, Software reset, Read/Write FIFO, Interrupt control, and User logic software register as shown in Figure 5-7, then click on Next.

Note that no master services are selected.

Create Peripheral - IPIF (IP Interface) Services	
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) models in the interface between the PLB interconnect and the user logic. Besides the standard functions like add IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation. IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation. IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation. IPIF module, the wizard tool also offers other commonly used services and configuration. IPIF module, the wizard tool also offers other commonly used services and configuration. IPIF module, the wizard tool also offers other commonly used services and configuration. IPIF module, the wizard tool also offers other commonly used services and configuration. IPIF module, the wizard tool also offers other commonly used services and configuration. IPIF module, the wizard tool also offers other complex peripherals included if master service selected). IPIF module, the wizard tool also offers other complex peripherals included if master service selected). IPIF module, the wizard tool also offers other complex peripherals included if master service selected). IPIF module, the wizard tool also offers other complex peripherals included if master service selected). IPIF module, the wizard tool also offers other complex peripherals include if master service selected).	ules, which provide you with a quick way dress decoding provided by the slave ion of the design. for operations like logic control, status /address space access, and etc. (PLB). Ver logic software register User logic memory space als like Ethemet and PCI for egions (PLB master interface will be
More Info	Next > Cancel

UG443_4_7_082007

Figure 5-7: Create/IP Wizard IPIF Services



8. In the Create Peripheral - Slave Interface window, several design considerations are needed for OPB and PLBV34.

If Burst and cache-line support is not enabled, the native data width is fixed at 32 bits, and bursting is not available. If Burst and cache-line support is enabled, the native data width is variable at 32, 64, and 128 bits, and bursting is available.

If the Smallest Master Native Dwidth < Slave Native Dwidth, byte steering logic is included which takes additional resources.

♦ OPB

In a typical OPB system the largest master has a data width of 32 bits. In this case, the user has an option of enabling or disabling Burst and cache-line support because the data width of the bus is 32 bits. The user should base this decision on the functionality of the core. If Burst and cache-line support is enabled, set the inclusion of the Write data buffer.

For this example, check **Burst and cache-line support**, set the **Native data width:** to **32** bit, and set the **Write buffer depth:** to **16** as shown in Figure 5-8, then click on **Next**.

Slave Interface Configure the slave interface of your peripheral
The IPIF slave library provides a quick way to implement a slave interface between the user logic and the PLB v4.6 interconnect. It provides address
decoding over various ranges as configured by the user and implements the protocol and timing translation between the PLB v4.6 interconnect and the IPIC (IP InterConnect . interface between user logic and IPIF).
Slave performance Slave peripherals support single beat read/write data transfers by default. If performance is key to the slave peripheral (i.e. memory controllers), you can have the burst transfer support turned on - this feature provides higher data transfer rates for the PLB Cacheline access and enables the transfer protocol for PLB Fixed Length Burst operations.
Data width The native bit width of the internal data bus may be less than or equal to the PLB slave interface data bus width. To conserve FPGA resources, set the value to be the same as the smallest PLB master in the system that may interact with your peripheral. Native data width: 32 v bit
Write data buffer To increase maximum frequency, the IPIF slave library utilizes a write buffer to allow back-to-back data beat transfers on the PLB bus even when the user logic is not capable of acknowledging the writes this quickly. A small buffer will cause flow control on the PLB v4.6 interconnect, while a large buffer takes up FPGA resources. Setting the depth to 0 will eliminate the write data buffer (some peripherals do not need a write buffer or either the user logic will instantiate a buffer alternatively), but keep in mind that excluding the write buffer may expose the user logic to critical path. Write buffer depth: 16
More Info

Figure 5-8: Create/IP Wizard Slave Interface for OPB Migration

◆ PLBV34

In a typical PLBV34 system the largest master has a data width of 64 bits. In this case, check **Burst and cache-line support**. The user has the option of setting the Native data width based upon needs of the system. In addition, the user can set the Write data buffer.

In this example, set the **Native data width:** to **64** bits and the **Write buffer depth:** to **16** as shown in Figure 5-9, then click on **Next**.

💠 Create Peripheral - Slave Interface		
Slave Interface Configure the slave interface of your peripheral	*	
The IPIF slave library provides a quick way to implement a slave interface between the user logic and the PLB v4.6 interconnect. It provides and decoding over various ranges as configured by the user and implements the protocol and timing translation between the PLB v4.6 interconnect at (IP InterConnect . interface between user logic and IPIF).	Iress and the IPIC	
Slave performance Slave performance Slave peripheral support single beat read/write data transfers by default. If performance is key to the slave peripheral (i.e. memory controllers) have the burst transfer support turned on - this feature provides higher data transfer rates for the PLB Cacheline access and enables the transfer rotocol for PLB Fixed Length Burst operations.	, you can fer	
Data width The native bit width of the internal data bus may be less than or equal to the PLB slave interface data bus width. To conserve FPGA resource value to be the same as the smallest PLB master in the system that may interact with your peripheral. Native data width: 64 v bit	s, set the	
Write data buffer To increase maximum frequency, the IPIF slave library utilizes a write buffer to allow back-to-back data beat transfers on the PLB bus even when the user logic is not capable of acknowledging the writes this quickly. A small buffer will cause flow control on the PLB v4.6 interconnect, while a large buffer takes up FPGA resources. Setting the depth to 0 will eliminate the write data buffer (some peripherals do not need a write buffer or either the user logic will instantiate a buffer alternatively), but keep in mind that excluding the write buffer may expose the user logic to critical path. Write buffer depth: 16		
More Info	Cancel	

Figure 5-9: Create/IP Wizard Slave Interface for PLB Migration



9. In the Create Peripheral - FIFO Service window, make the selections shown in Figure 5-10 to configure the read/write FIFO, then click on **Next**.

🗢 Create Peripheral - FIFO Service	
FIFO Service Configure the read/write FIFO in the IPIF.	*
Read/Write FIFO provides data buffering service, which may automatically utiliz packet mode is turned off and only 4 to 16 words of FIFO memory depth is need Use gacket mode Use gacket mode	tes SRL16 primitives for the memory medium instead of BRAM primitives if led. I Include Write FIFO Use packet mgde Uge vacancy calculation Numbgr of Write FIFO entries: 512 Write FIFO Use packet mgde Use packet mgde Use packet mgde Use vacancy calculation Numbgr of Write FIFO entries: 512 Use packet mgde Use packet mgde Use packet mgde Use vacancy calculation Numbgr of Write FIFO entries: 512 Use packet mgde Use packet m
More Info	< Back Next > Cancel

Figure 5-10: Create/IP Wizard FIFO Service

10. In the Create Peripheral - Interrupt Service window, check **Use Device ISC (Interrupt source controller)** and **Use Device ISC Priority Encoder service**. Select **Level Pass Through (non-inverted)** as the **Capture Mode**, and **1** as the **Number of interrupts generated by user logic:** as shown in Figure 5-11:

Click on Next.

💠 Create Peripheral - Interrupt Service	
Interrupt Service Configure interrupt handling.	
The interrupt control service provides interrupt capture support which capture and user logic into a single interrupt output.	es and coalesces various interrupts generated from IPIF, other design blocks Device ISC Device ISC (Interrupt Source Controller) coalesces all captured internal interrupts into a single output signal. You may eliminate Device ISC if all interrupts come from the user logic. Image: Comparison of the source controller) Priority Encoder Device ISC Priority Encoder (Interrupt ID register) indicates which interrupt source has a pending interrupt. It is useful in aiding the user interrupt service routine to resolve the source of an interrupt. Image: Use Device ISC Priority Encoder service Use Device ISC Priority Encoder service User logic interrupt Number of interrupts generated by user-logic: Image: Level Pass Through (non-inverted) Image: Level Pass Through (non-inverted) Image: Device Ison of the user logic has no additional capture processing applied to it. It is immediately sent to the IP ISC Interrupt
Datasheet	
More Info	< Back Next > Cancel

Figure 5-11: Create/IP Wizard Interrupt Service

11. In the Create Peripheral - User S/W Register window, set the **Number of software** accessible registers: to 4 as shown in Figure 5-12, then click on **Next**.

💠 Create Peripheral - User S/W Register
User S/W Register Configure the software accessible registers in your peripheral.
The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference. User logic software registers may take full advantage of the slave IPIF
Bus2IP_RdReq Bus2IP_WReq Bus2IP_RdCE Bus2IP_Data Bus2IP_Data Bus2IP_Data
P2Bus_Data P2Bus_RdAdk P2Bus_WrAck P2Bus_Error User Logic
More Info Cancel

Figure 5-12: Create/IP Wizard User S/W Register

12. In the Create Peripheral - IP Interconnect (IPIC) window shown in Figure 5-13, note the standard IPIC signals connected to the user logic that have been automatically selected by the Wizard, then click on **Next**.

💠 Create Peripheral - IP Interconnect ((IPIC)			
IP Interconnect (IPIC) Select the interface between the logic to be in	nplemented 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.				
Peripheral PLBv46 Slave Blocks PLBv46 Blocks PLBv46 Master Bayes Satto to J J L User Logic	♥ Bus2IP_Clk ♥ Bus2IP_Reset ▶us2IP_Addr ▶us2IP_CS ▶us2IP_Data ♥ bus2IP_BE ♥ bus2IP_RdCE ♥ bus2IP_Burst ♥ bus2IP_WrReq ♥ IP2Bus_Data ♥ IP2Bus_WrAck ♥ IP2Bus_WrAck ♥ IP2Bus_Error Restore Defaults	Port description		
More Info		< <u>Back</u> <u>Next</u> > Cancel		

Figure 5-13: Create/IP Wizard IP Interconnect (IPIC)

🗇 Create Peripheral - (OPTIONAL) Peripheral Si	mulation Support	
(OPTIONAL) Peripheral Simulation Support Generate optional files for simulation using Bus Functiona	I Models (BFM).	
The EDK provides a BFM simulation platform to help you simulat	Iate your peripheral. Indicate if you want this tool to generate the appropriate HD Generate BFM simulation platform for ModelSim-SE or ModelSim-PE This feature requires that you have accepted the associated IBM license and installed the BFM package. The link below shows how: BFM Package Installation Instructions	L and Bus agreement
More Info	< Back Next >	Cancel

UG443_4_14_082007

XILINX[®]

Figure 5-14: Create/IP Wizard Peripheral Simulation Support
14. In the Create Peripheral - (OPTIONAL) Peripheral Implementation Support shown in Figure 5-15, check Generate ISE and XST project files to help you implement the peripheral using XST flow and Generate template driver files to help you implement software interface, then click on Next.

🔶 Create Peripheral - (OPTIONAL) F	Peripheral Implementation Support
(OPTIONAL) Peripheral Implementation Generate optional files for hardware/softw	n Support vare implementation
Upon completion, this tool will create synthesi will need to complete the implementation of th synthesizable templates, so that you can hook	zable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You is module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the cup the generated peripheral to a processor system.
Peripheral (VHDL)	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.
IPIF (VHDL)	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
User Logic (VHDL)	Generate template <u>d</u> river files to help you implement software interface
More Info	< Back Next > Cancel

Figure 5-15: Create/IP Wizard (OPTIONAL) Peripheral Implementation Support



15. In the Create Peripheral - Finish window shown in Figure 5-16, click on **Finish** to generate the core.



Figure 5-16: Create/IP Wizard Finish

Parameters for PLBV46 Slave Cores

Certain parameters for the PLBV46 Slave interface are necessary for the tools as shown in Table 5-1.

Note: Some parameters in Table 5-1 are overwritten inside EDK to the system values depending on the system.

Table 5-1: PLBV46 Slave Burst and PLBV46 Slave Single Required Parameters

Parameter	Description	Allowable Values
C_SPLB_AWIDTH	Width of the PLB Address Bus.	32
C_SPLB_DWIDTH	Width of the PLB Data Bus.	32,64,128
C_SPLB_NATIVE_DWIDTH	Width of the slave IPIC data bus.	32,64,128
C_SPLB_NUM_MASTERS	Number of PLB Masters.	1 to 16
C_SPLB_MID_WIDTH	PLB Master ID Bus Width.	1 to 4
C_SPLB_SUPPORTS_BURST	Slave supports burst.	0,1
C_SPLB_P2P	Selects point-to-point or shared PLB topology.	0,1
C_SPLB_SMALLEST_MASTER	Data width of smallest master that will access the IPIF.	32,64,128

Note: Usually the PLBV34 bus has a DWIDTH of 64 bits, while the OPB has a DWIDTH of 32 bits.

Table 5-2 shows the parameters specific to the PLBV46 Slave Single IPIF entity in top level template.

Table 5-2: PLBV46 Slave Single Parameters

Parameter	Description	Allowable Values
C_SIPIF_DWIDTH	Same as C_SPLB_NATIVE_DWIDTH.	32
C_BUS2CORE_CLK_RATIO	Selects the ratio of bus clock to core clock for use in dual clock systems.	1,2

The user has an option of running the core at one half of the bus speed by setting the C_BUS2CORE_CLK_RATIO parameter to 2 (2:1 ratio). The user must provide a slower clock that is edge synchronized with the bus clock and not use the Bus2IP_Clk from the IPIF. The Create IP Wizard does not have this option when generating a slave core.

Table 5-3 shows the parameters specific to the PLBV46 Slave Burst IPIF entity in the top level template.

Table 5-3: PLBV46 Slave Burst Parameters

Parameter	Description	Allowable Values
C_SIPIF_DWIDTH	Same as C_SPLB_NATIVE_DWIDTH.	32,64,128
C_WR_BUFFER_DEPTH	Size of the write buffer.	0,16,32,64
C_CACHELINE_ADDR_MODE	Cache Link addressing mode.	0,1

The C_WR_BUFFER_DEPTH parameter determines the size of the write buffer. If the parameter is set to **0**, the buffer is removed. When a buffer exists, (only with the PLBV46 Slave Burst), special consideration is needed when dealing with IPIC errors.

Modifying the Existing User Logic

Modifications are needed to the existing OPB and PLBV34 User Logic source VHDL files to match the top template generated by the Create IP Wizard. A block diagram of this process is shown in Figure 5-17.



Figure 5-17: Modifying User Logic

Modifying Existing OPB User Logic

The entity of the existing user logic is modified to match the top level template declaration of User Logic.

Modifying User Logic Entity Generics

The following parameters are globally replaced in the existing User Logic.

C_DWIDTH -> C_SLV_DWIDTH

C_NUM_CE->C_NUM_REG

C_IP_INTR_NUM->C_NUM_INTR

Remove the parameters, C_RDFIFO_DWIDTH and C_WRFIFO_DWIDTH, from the entity. The other instances in the User Logic are replaced with C_SLV_DWIDTH in both cases.

8);

Modifying User Logic Entity Ports and Connecting Logic

The following ports are added to the entity:

Bus2IP_BurstLength	:	in	<pre>std_logic_vector(0 to</pre>
Bus2IP_RdReq	:	in	<pre>std_logic;</pre>
Bus2IP_WrReq	:	in	<pre>std_logic;</pre>
IP2Bus_RdAck	:	out	<pre>std_logic;</pre>
IP2Bus_WrAck	:	out	<pre>std_logic;</pre>

The following port is removed because there are separate acknowledgement signals for read and write.

IP2Bus Ack : out std logic;

The following connecting logic is removed:

```
IP2Bus_AddrAck <= slv_write_ack or slv_read_ack;</pre>
```

The removed line is replaced with:

IP2Bus_WrAck <= slv_write_ack;</pre> IP2Bus RdAck <= slv read ack;</pre>

Modifying Existing PLBV34 User Logic

The entirety of the existing user logic is modified to match the top level template declaration of User Logic.

Modifying User Logic Entity Generics

The following parameters are globally replaced inside the existing User Logic

C_DWIDTH -> C_SLV_DWIDTH

C_NUM_CE->C_NUM_REG

C_IP_INTR_NUM->C_NUM_INTR

Remove the following parameters from the entity. C_RDFIFO_DWIDTH and C_WRFIFO_DWIDTH. The other instances inside the User Logic are replaced with C_SLV_DWIDTH in both cases.

Modifying User Logic Entity Ports and Connecting Logic

The following ports are added to the entity:

Bus2IP_BurstLength : in std_logic_vector(0 to 8);

The follow ports are removed from the entity:

IP2Bus_ToutSu	? :	out	std_	_logic;
IP2Bus_Busy	:	out	std	logic;
IP2Bus_Retry	:	out	std_	_logic;

The following connecting logic is removed:

IP2Bus_	_Busy	<=	'0';
IP2Bus_	_Retry	<=	'0';
IP2Bus	ToutSup	<=	'0';





Chapter 6

Migration of User IP Master/Slave Cores

Introduction

This chapter describes the migration process to go from an existing OPB or PLBv 3.4 master/slave core to a PLB v4.6 master/slave core.

The following migration steps will be discussed:

- 1. Review the Create IP Wizard master/slave services available for OPB/PLB v3.4 cores
- 2. Review the Create IP Wizard master/slave services available for PLB v4.6 cores
- 3. Describe services and functionality of the original OPB/PLB v3.4 master/slave cores created by Create IP Wizard
- 4. Use Create IP Wizard, within the EDK tools, to create an equivalent PLB v4.6 core for both PLB v3.4 or OPB master/slave cores
 - a. Set up services to match the functionality of the original OPB/PLB v3.4 master/slave core
 - b. Describe parameters dealing with the PLB v4.6 master/slave pcores

Overview of Create IP Wizard for OPB/PLBV34 Master/Slaves

Create IP Wizard Master Services

Create IP Wizard allows for the user to select DMA and User logic master support for both OPB and PLB v3.4. The wizard then creates an example logic inside the User Logic to demonstrate Master transactions.

For OPB, the master and slave interfaces has a data width of 32-bits. For PLB v3.4, the master and slave interfaces has a data width of 64-bits.

OPB/PLBV34 IPIFs

The OPB/PLBV34 IPIFs includes slave services and master services which are enabled/disabled through parameters inside the IPIF instance. Create IP Wizard enabled/disabled these parameter based upon user sections inside the Wizard.

Overview of Create IP Wizard for PLB v4.6 Master/Slaves

Create IP Wizard Master Services

Create IP Wizard offers the User logic master support service. The user has a option of adding Burst and cache-line support for the master interface. Burst and native data width of the Master are discussed later in this chapter.

The wizard then creates an example inside the User Logic logic showing the IPIC master interface receiving and transmitting data via the LocalLink interface.

Note: No DMA services are available through Create IP Wizard.

PLBV46 Master/Slave IPIFs

With PLBV46 IPIFs, master/slave cores require 2 separate PLBV46 Slave and PLBV46 Master IPIFs, for the master and slave interfaces for the core. For example, PLBV46 Slave Single can be used for slave registers to setup master transactions and the PLBV46 Master Burst can be used for the Master attachment.

Unlike the OPB/PLBV34 IPIFs, the master read/write transactions no longer involve the slave attachment. In turn, the master no longer supplies an IP2IP address.

With PLBV46 Master Burst, the User IP reads and writes from the PLB Master via the Xilinx LocalLink Interface protocol which is a different protocol than PLBV34/OPB IPIFs. The PLBV46 Master Single uses a simplified LocalLink Interface protocol for single reads and writes.

Indeterminate length bursts are not supported with PLBV46 Master Burst. The User Logic must only generate fixed length bursts of 2 to 16 data beats. The length is specified in bytes and a multiple of the Native Dwidth/8. In addition, transactions cannot be aborted. With unaligned addresses, the User IP has to issue single data beat requests until address alignment is established. Access to critical slave registers from the processor or another master that control Master operations must be considered since there is no IP2Bus_Busy signal from the slave.

The selection of the PLBV46 Master IPIF depends on the selection of Burst and cache-line support inside the Wizard. If Burst and cache-line support is not enabled inside the wizard, the PLBV46 Master Single IPIF is used for the master interface. If Burst and cache-line support is enabled, the PLBV46 Master Burst IPIF is used for the master interface.

No master services, like DMA, are included with the PLBV46 Master IPIFs. The user has the option of adding LocalLink interface(s) to the User Logic which can be connected to the Soft DMA(SDMA). DMA migration is described later inside the *Migration of DMA Solutions* chapter.

Overview of OPB/PLBV34 Master/Slave Cores Created By Create IP Wizard

The OPB and PLBV34 Master/Slave Example pcores were created with Create IP Wizard, within the EDK tools. Both example cores use DMA in Simple Mode and User logic master support while no slave services are selected. Even though the master support is selected, the OPB IPIF or the PLBV34 IPIF is set to include both a master and slave. The slave is used to access registers that control master transactions. In this configuration, the master will contain sixteen 8-bit registers for control and status.

Creating the PLBV46 Master/Slave Cores Using Create IP Wizard

Generating the PLBV46 Master/Slave Cores

Follow the steps in the following section to generate both the migrated OPB and PLBV34 master/slave cores. Notice the migration considerations for both OPB/PLBV34 in the following steps.

1. Invoke the Create IP Wizard and click on **Next** as shown in Figure 6-1.



Figure 6-1: Create/IP Wizard Welcome



2. In the Create and Import Peripheral Wizard - Peripheral Flow window, under **Select** flow, select **Create templates for a new peripheral** as shown in Figure 6-2, then click on **Next**.

💠 Create and Import Peripheral Wizard - F	Peripheral Flow	
Peripheral How Indicate if you want to create a new peripheral or in	mport an existing peripheral.	
This tool will help you create templates for a new EDK the interface files and directory structures required by the interface files and directory structures required	compliant peripheral, or help you import an existing peripheral into an XPS project or E EDK will be generated. Select flow Create templates for a new peripheral Import existing peripheral Plow description This tool will create HDL templates that have the EDK compliant port/parameter in will need to implement the body of the peripheral.	DK repository.
More Info	< <u>B</u> ack <u>N</u> ext >	Cancel

UG443_5_2_082007

Figure 6-2: Create/IP Wizard Peripheral Flow

3. In the Create Peripheral - Repository or Project window, select a location for the repository where the cores will be generated as shown in Figure 6-3. then click on **Next**.

🗢 Create Peripheral - Repository or Project	
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 an EDK repository, the peripheral can be accesse XPS projects.	ed by multiple
To an EDK user repository (Any directory outside of your EDK installation path)	
Repository: C:\edk_user_repository	Bro <u>w</u> se
○ To an XPS project Project: ▼	Browse
C:\edk_user_repository\MyProcessorIPLib\pcores	
More Info	Cancel

UG443_5_3_082007

Figure 6-3: Create/IP Wizard Repository



4. In the Create Peripheral - Name and Version window, enter the name of the master/slave migrated core that will be created. The OPB migration example is shown in Figure 6-4. Click on **Next**.

💠 Create Peripheral - Name and Version	
Name and Version Indicate the name and version of your peripheral.	X
Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.	
Name: xps_opb_master_example	
Version: 1.00.a	
Major revision: Minor revision: <u>H</u> ardware/Software compatibility revision:	
1 🐼 00 😴 a 💭	
Description:	
Logical library name: xps_opb_master_example_v1_00_a All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK	rame
repositories indicated in the XPS project settings.	
More Info	Cancel

UG443_5_4_082007

Figure 6-4: Create/IP Wizard Name and Version for OPB MIgration

In the Create Peripheral - Name and Version window, enter the name of the master/slave migrated core that will be created. The PLB migration example is shown in Figure 6-5. Click on **Next**.

💠 Create Peripheral - Name and Version	
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: xps_plb_master_example	
Version: 1.00.a	
Major revision: Minor revision: <u>H</u> ardware/Software compatibility revision:	
1 🔿 00 😴 a 💭	
Description:	
Lasiallitary area una alterrativa avanula y 1.00 a	
All HDL files (either created by you or generated by this tool) that are used to implement this perioheral must be commiled into the logical library	vname
above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.	:
More Info	Cancel

UG443_5_5_082007

Figure 6-5: Create/IP Wizard Name and Version for PLB Migration



XILINX[®]

Bus Interface Indicate the bus interface supported by your peripheral. To which bus will this peripheral be attached? • [Processor Local Bus (PLB v4.6)] • [ast Simplex Link (FSL) • [FSL) ATTENTION Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) buses (including PLB v4.6 interconnect) and the FSL interface.
To which bus will this peripheral be attached?
ATTENTION Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) buses (including PLB v4.6 interconnect and OPB/PLB v3.4 interconnect) and the FSL interface.
NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface. <u>CoreConnect Specification</u> <u>PLB (v4.6) Slave IPIF Specification for single data beat transfer</u> <u>PLB (v4.6) Slave IPIF Specification for burst data transfer</u> <u>PLB (v4.6) Master IPIF Specification for single data beat transfer</u> <u>PLB (v4.6) Master IPIF Specification for burst data transfer</u> <u>PLB (v4.6) Master IPIF Specification for burst data transfer</u>
Note Xilinx recommends using the new PLB v4.6 bus standard, however, the wizard still supports the OPB and PLB v3.4 bus interfaces. Enable OPB and PLB v3.4 bus interfaces
More Info < Back

Figure 6-6: Create/IP Wizard Bus Interface

6. In the Create Peripheral - IPIF (IP Interface) Services window, check **User logic master** as shown in Figure 6-7. No slave services are selected. Click on **Next**.

🗇 Create Peripheral - IPIF (IP Interface) Services	
IPIF (IP Interface) Services Indicate the IPIF services required by your peripheral.	
Your peripheral will be connected to the PLB (v4.6) interconnect through to implement the interface between the PLB interconnect and the user log IPIF module, the wizard tool also offers other commonly used services and Processor Local Bus (version 4.6) PLBv46 PLBv46 PLBv46 Slave PLBv46 PLBv46 Slave PLBv46 Master Write FIFO Write FIFO Write FIFO Master Cnthr	corresponding PLB IP Interface (IPIF) modules, which provide you with a quick way gic. Besides the standard functions like address decoding provided by the slave d configuration 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 EIFO User logic memory gpace Interrupt control Master service and configuration Typically required by complex peripherals like Ethemet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected). Image: User logic master User logic master
More Info	< <u>B</u> ack <u>N</u> ext > Cancel
	UG443_5_7_08200

Figure 6-7: Create/IP Wizard IPIF Services



7. In the Create Peripheral - Slave Interface window, confirm that **Burst and cache-line support** is not selected as shown in Figure 6-8, then click on **Next**.

🗢 Create Peripheral - Slave Interface	
Slave Interface Configure the slave interface of your peripheral	*
The IPIF slave library provides a quick way to implement a slave interface between the user logic and the PLB v4.6 interconnect. It provides a decoding over various ranges as configured by the user and implements the protocol and timing translation between the PLB v4.6 interconnec (IP InterConnect . Interface between user logic and IPIF).	ddress t and the IPIC
Slave peripherals support single beat read/write data transfers by default. If performance is key to the slave peripheral (i.e. memory controller have the burst transfer support turned on - this feature provides higher data transfer rates for the PLB Cacheline access and enables the transfer rotocol for PLB Fixed Length Burst operations.	s), you can Isfer
Data width — The native bit width of the internal data bus may be less than or equal to the PLB slave interface data bus width. To conserve FPGA resource value to be the same as the smallest PLB master in the system that may interact with your peripheral. Native data width: 32 v bit	ces, set the
More Info	Cancel

UG443_5_8_082007

Figure 6-8: Create/IP Wizard Slave Interface

8. In the Create Peripheral - Master Interface window, several design considerations are needed.

Burst support should be checked if bursting support is needed or the native dwidth of the master needs to be **64** or **128**.

There are design considerations for the Native Dwidth if Burst support is enabled. If the Smallest Slave Native Dwidth < Master Native Dwidth, conversion cycles are required which takes additional resources. The native dwidth of the master should be equal to the smallest native dwidth of the slave inside the desired system to save resources.

For the OPB example, **Burst support** is **enabled** and the **Native data width:** is set to **32** as shown in Figure 6-9. Click on **Next**.

💠 Create Peripheral - Master Interface	
Master Interface Configure the master interface of your peripheral	*
The IPIF master library provides a quick way to implement a mastering interface between the user logic and the PLB v4.6 interconnect. Tran protocol between the user logic and the PLB is through the IPIC master interface, and the user logic receives data from and transmits data t Xilinx LocalLink interface protocol. An example logic showing operation on the IPIC master interface to receive/transmit data via the LocalLi be included in the generated user logic module for your reference.	nsfer request to the PLB via the ink interface will
Master performance Master performance Master peripherals support single beat read/write data transfers by default. If performance is key to the master peripheral (i.e. Ethemet), yo the burst transfer support turned on - this feature enables the transfer protocol for PLB Fixed Length burst operations of 2 to 16 data beats U	ou can have 8.
Data width The native bit width of the internal data bus may be less than or equal to the PLB master interface data bus width. To conserve FPGA resolute to be the same as the smallest PLB slave in the system that may interact with your peripheral. Native data width: 32 v bit	sources, set the
More Info	Cancel

Figure 6-9: Create/IP Wizard Master Interface for OPB MIgration

UG443 5 9 082007



For the PLB example, check **Burst support** and set the **Native data width** to **64** as shown in Figure 6-10. Click on **Next**.

🗢 Create Peripheral - Master Interface	
Master Interface Configure the master interface of your peripheral	
The IPIF master library provides a quick way to implement a mastering interface between the user logic and the PLB v4.6 interconnect. Transfer protocol between the user logic and the PLB is through the IPIC master interface, and the user logic receives data from and transmits data to the Xlinx LocalLink interface protocol. An example logic showing operation on the IPIC master interface to receive/transmit data via the LocalLink be included in the generated user logic module for your reference.	er request he PLB via the interface will
Master performance Master performance Master peripherals support single beat read/write data transfers by default. If performance is key to the master peripheral (i.e. Ethemet), you the burst transfer support turned on - this feature enables the transfer protocol for PLB Fixed Length burst operations of 2 to 16 data beats.	can have
Data width The native bit width of the internal data bus may be less than or equal to the PLB master interface data bus width. To conserve FPGA resouvalue to be the same as the smallest PLB slave in the system that may interact with your peripheral. Native data width:	rces, set the
More Info	Cancel

UG443_5_10_082007

Figure 6-10: Create/IP Wizard Master Interface for PLB MIgration

9. In the Create Peripheral - IP Interconnect (IPIC) window, make the selections shown in Figure 6-11. These are the standard IPIC signals connected to the user logic. Click on **Next**.

🔶 Create Peripheral - IP Interconnect	(IPIC)		
IP Interconnect (IPIC) Select the interface between the logic to be in	mplemented in your peripheral and the IPIF.		\$ \$
Your peripheral will be connected to the PLB (v4. interfaces to the IPIF module(s) and other sub-blo present, some are pre-selected based on the IPIF needs.	6) interconnect through suitable IPIF master/ cks through a set of signals called the IP inte services you required, and you can choose Note: all IPIC ports are active high.	slave module(s). Your custom logic from the user-logic rconnect (IPIC) interface. Some of the ports are alway other optional ports to be included in the design based	module s I on your
Peripheral PLBv46 Slave Other PLBv46 Master Blocks PLBv46 Blocks PLBv46 Master Blocks PLBv46 Master Blocks PLBv46 Blocks PLBv46	Bus2IP_Clk Bus2IP_Reset Bus2IP_Addr Bus2IP_CS Bus2IP_Data Bus2IP_BE Bus2IP_RCE Bus2IP_WrCE IP2Bus_Data IP2Bus_WrAck IP2Bus_WrAck IP2Bus_MstRd_Req IP2Bus_MstRd_Req IP2Bus_Mst_Addr IP2Bus_Mst_Length Restore Defaults	Port description	
More Info		< Back Next > C	

Figure 6-11: Create/IP Wizard IP Interconnect (IPIC)



10. In the (OPTIONAL) Peripheral Simulation Support window, the user can click on **Next** to continue or has the option of creating a BFM simulation environment to test the core by selecting **Generate BFM simulation platform for ModelSim-SE or ModelSim-PE**.

🔶 Create Peripheral - (OPTIONAL) Peripheral Si	mulation Support	
(OPTIONAL) Peripheral Simulation Support Generate optional files for simulation using Bus Functional	Models (BFM).	
The EDK provides a BFM simulation platform to help you simul Functional Language (BFL) stimulus file for the target bus.	Iate your peripheral. Indicate if you want this tool to generate the appropriate HE Generate BFM simulation platform for ModelSim-SE or ModelSim-PE This feature requires that you have accepted the associated IBM license and installed the BFM package. The link below shows how: BFM Package Installation Instructions	L and Bus
More Info	< <u>B</u> ack <u>N</u> ext >	Cancel

UG443_5_12_082007

Figure 6-12: Create/IP Wizard Peripheral Simulation Support

- 11. In the Create Peripheral (OPTIONAL) Peripheral Implementation Support window, make the selections as shown in Figure 6-13:
 - a. Check Generate ISE and XST project files to help you implement the peripheral using XST flow
 - b. Check Generate template driver files to help you implement software interface
 - c. Click on Next.

Create Peripheral - (OPTIONAL) P (OPTIONAL) Peripheral Implementation Generate optional files for hardware/softw	eripheral Implementation Support a Support vare implementation	
Upon completion, this tool will create synthesis will need to complete the implementation of th synthesizable templates, so that you can hook Peripheral (VHDL)	table HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be is module using standard HDL design flows. The tool will also generate EDK interface files (mpd/ up the generated peripheral to a processor system. Note Should the peripheral interface (ports/parameters) or file list change, you will need to regen EDK interface files using the import functionality of this tool.	created. You pao) for the nerate the
IPIF (VHDL)	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]
User Logic (VHDL)		
More Info	< <u>B</u> ack <u>N</u> ext >	Cancel

Figure 6-13: Create/IP Wizard (OPTIONAL) Peripheral Implementation Support



12. In the Create Peripheral - Finish window, click on **Finish** to generate the core as shown in Figure 6-14.

💠 Create Peripheral - Finish	
	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: \edk_user_repository\MyProcessorIPLib/pcores/xps_opb_master_example_v1_00_ and peripheral software templates under C: \edk_user_repository\MyProcessorIPLib/drivers/xps_opb_master_example_v1_00_ respectively.
	Peripheral Summary:
	top name : xps_opb_master_example version : 1.00.a type : PLB (v4.6) master slave features : slave attachment master attachment user master model
	Address Block Summary:
More Info	< <u>B</u> ack <u>Finish</u> Cancel

UG443_5_14_082007

Figure 6-14: Create/IP Wizard Finish

Parameters for PLBV46 Master/Slave Cores

Certain parameters for the PLBV46 Master interface are necessary for the tools. Table 6-1 shows these parameters. The necessary parameters for the PLBV46 Slave interface is shown in the previous chapter.

Note: Some of the parameters may be overwritten inside EDK to system values depending on the system.

Table 6-1: PLBV46 Master Necessary Parameters

Parameter	Description	Allowable Values
C_MPLB_NATIVE_DWIDTH	Specifies the internal native data width of the master.	32,64,128
C_MPLB_AWIDTH	Width of the PLB Address Bus.	32
C_MPLB_DWIDTH	Width of the PLB Data Bus.	32,64,128
C_MPLB_SMALLEST_SLAVE	Indicated the smallest Native Dwidth of any Slave attached to the PLBV46 used by the master.	32,64,128

The PLBV46 Master Burst parameters are described in Table 6-2. The C_INHIBIT_CC_BLE_INCLUSION parameter is set to **0** to automatically include the logic for conversion cycles and burst length expansion.

Table 6-2: PLBV46 Master Burst Parameters

Parameter	Description	Allowable Values
C_MPLB_NATIVE_DWIDTH	Specifies the internal native data width of the master.	32,64,128
C_MPLB_AWIDTH	Width of the PLB Address Bus.	32
C_MPLB_DWIDTH	Width of the PLB Data Bus.	32,64,128
C_MPLB_SMALLEST_SLAVE	Indicated the smallest Native Dwidth of any Slave attached to the PLBV46 used by the master.	32,64,128
C_INHIBIT_CC_BLE_INCLUSION	Parameter is used to override the automatic inclusion of the Conversion Cycle and Burst length Expansion logic.	0,1

The PLBV46 Master Single parameters are described in Table 6-3.

Table 6-3: PLBV46 Master Single Parameters

Parameter	Description	Allowable Values
C_MPLB_NATIVE_DWIDTH	Specifies the internal native data width of the master.	32
C_MPLB_AWIDTH	Width of the PLB Address Bus.	32
C_MPLB_DWIDTH	Width of the PLB Data Bus.	32,64,128

