

Introduction

MPMC is a fully parameterizable memory controller that supports SDRAM/DDR/DDR2 memory. MPMC provides access to memory for one to eight ports, where each port can be chosen from a set of Personality Interface Modules (PIMs) that permit connectivity into PowerPC® 405 processors and MicroBlaze™ processors using CoreConnect® PLBv4.6 and the MPMC Native Port Interface (NPI) structures, as well as a Memory Interface Block (MIB) PIM (PPC440MC) for the PowerPC 440 Processor. MPMC supports the Soft Direct Memory Access (SDMA) controller that provides full-duplex, high-bandwidth, LocalLink interfaces into memory. A Video Frame Buffer Controller (VFBC) PIM is also available. Additionally, MPMC supports optional Error Correcting Code (ECC), Performance Monitoring (PM), and Debug Registers.

Features

- Soft Direct Memory Access (SDMA) support
- Double Data Rate (DDR/DDR2) and Single Data Rate (SDR) SDRAM memory support.
- DIMM support (registered and unbuffered).
- Error Correcting Code (ECC) Performance Monitoring (PM), and Debug Register support.
- Parameterizable:
 - number of ports (1 to 8)
 - number of data bits to memory (8, 16, 32, 64)
 - configuration of data path FIFOs
- Memory Interface Generator (MIG)-based PHY v.3.0 support.
- Static Physical (PHY) interface alternative to the MIG-based PHY.
- User configuration of arbitration algorithms.
- Customizable Interfaces: Xilinx CacheLink (XCL), LocalLink (using SDMA), Processor Local Bus (PLB v4.6) with Xilinx simplifications, Native Port Interface (NPI), MIB/PPC440MC, and VFBC.

LogiCORE™ Facts	
Core Specifics	
Supported Device Family ⁽¹⁾ and Design Tool Requirements	Device Family and Design Tool List
Resources Used	See "Performance, Timing, and Resource Utilization," page 172.
Provided with Core	
Documentation	DS643 Multi-Port Memory Controller (MPMC) (v5.00.a) Data Sheet (this document)
Design File Formats	Verilog/VHDL
Reference Designs Application Notes	See "Reference Documents," page 193.
Support	
Provided by www.xilinx.com	
1. MPMC treats Automotive Spartan-3 devices (aspartan3) as an equivalent non-automotive Spartan-3. The "qvirtex4" and "qvirtex4" families are also treated as an equivalent "virtex4" family device. Note: The "aspartan3", "qvirtex4", and "qvirtex4" family MPMC and MIG designs have not been tested or characterized in hardware.	

Note: This document uses the terms "PLB v4.6" and "PLB" to refer to PLB v4.6 with Xilinx simplifications. MPMC does not support earlier versions of PLB (such as PLB v3.4). Also note that MPMC does not directly connect to OPB peripherals; such connection would require a PLB v4.6 to OPB or an OPB to PLB v4.6 bridge. See "Reference Documents," page 193 for more information about this bus standard and about the migration of designs to use this standard.

Design Parameters

The following tables provide the design parameters, allowable values, and descriptions for the MPMC system, associated memory, and Personality Interface Modules (PIMs). Parameter values that are strings or that contain alpha-numeric characters must be upper-case.

© Copyright 2002 – 2009 Xilinx, Inc. All Rights Reserved.
 XILINX, the Xilinx logo, the Brand Window and other designated brands included herein are trademarks of Xilinx, Inc.
 The PowerPC name and logo are registered trademarks of IBM Corp., and used under license. All other trademarks are the property of their respective owners.

Xilinx is providing this design, code, or information (collectively, the "Information") to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information 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.

MPMC System Parameters

Table 1: MPMC System Parameters

Parameter Name	Default Value	Allowable Values	Description
C_ALL_PIMS_SHARE_ADDRESSES ⁽¹⁾	1	0,1	Specifies whether MPMC ports use the C_MPMC_BASEADDR and C_MPMC_HIGHADDR for address decoding or if ports have independent address range decoding. Also specifies whether SDMA control register interfaces use the C_SDMA_CTRL_BASEADDR and C_SDMA_CTRL_HIGHADDR for address decoding or MPMC ports and SDMA control register ports have independent address range decoding. 1 = MPMC ports use C_MPMC_BASEADDR and C_MPMC_HIGHADDR for address decoding; SDMA control registers use C_SDMA_CTRL_BASEADDR and C_SDMA_CTRL_HIGHADDR. 0 = MPMC ports and SDMA control registers have independent address range decoding.
C_ARB_PIPELINE	1	0,1	Enables or disables the Arbiter Pipeline: 0 = Disable Arbiter Pipeline. 1 = Enable Arbiter Pipeline. (performance)
C_ARB_USE_DEFAULT	0	0	Default Arbitration Algorithm to use (<i>unimplemented</i>).
C_ARB0_ALGO	ROUND_ROBIN	ROUND_ROBIN, FIXED, CUSTOM	String that specifies the arbitration scheme to use for Algorithm 0 (Custom will consume a block RAM). Only valid if C_NUM_PORTS > 1. When set to FIXED, the priority order is from Port 0 to Port 7 and cannot be changed irrespective of C_ARB0_SLOTx settings.
C_ARB0_NUM_SLOTS	1	1-16	Number of time slots to use for Custom Algorithm. Only valid if C_ARB0_ALGO = CUSTOM.
C_ALL_PIMS_SHARE_ADDRESSES ⁽¹⁾	1	0,1	Specifies whether MPMC ports use the C_MPMC_BASEADDR and C_MPMC_HIGHADDR for address decoding or if ports have independent address range decoding. Also specifies whether SDMA control register interfaces use the C_SDMA_CTRL_BASEADDR and C_SDMA_CTRL_HIGHADDR for address decoding or MPMC ports and SDMA control register ports have independent address range decoding. 1 = MPMC ports use C_MPMC_BASEADDR and C_MPMC_HIGHADDR for address decoding; SDMA control registers use C_SDMA_CTRL_BASEADDR and C_SDMA_CTRL_HIGHADDR. 0 = MPMC ports and SDMA control registers have independent address range decoding.

Note:

- When C_ALL_PIMS_SHARE_ADDRESSES is set to 1, C_MPMC_BASEADDR is used for all ports for memory access addressing and

Table 1: MPMC System Parameters (Continued)

Parameter Name	Default Value	Allowable Values	Description
C_ARB0_SLOT0 . . . C_ARB0_SLOT15	NONE	String of Numbers, Example: "01234567"	Arbitration Priority for Time Slot <i>n</i> where <i>n</i> is 0-15, and the number of valid Time Slots is from 0 to (C_ARB0_NUM_SLOTS-1). Left to right, highest to lowest priority. Every valid port must be specified once only. Only valid if C_ARB0_ALGO = CUSTOM.
C_DEBUG_REG_ENABLE	0	0,1	Reserved.
C_FAMILY	virtex5	STRING	See C_FAMILY parameter values .
C_SUBFAMILY	"fx"	STRING	See C_SUBFAMILY parameter values . This parameter cannot be set by the user, and is only used for internal Design Rule Checks (DRC). Value is set to "" for all other families.
C_IDELAYCTRL_LOC ⁽⁵⁾	NOT_SET	STRING	IDELAYCTRL constraint locations (Hyphen separated).
C_MAX_REQ_ALLOWED	1	1	Number of requests MPMC can queue per port.
C_MPMC_BASEADDR ⁽¹⁾	0xFFFFFFFF	Valid Address	MPMC PIMs Shared Base Address.
C_MPMC_HIGHADDR ⁽¹⁾	0x00000000	Valid Address	MPMC PIMs Shared High Address. MPMC supports a maximum of 2 gigabytes of memory
C_MPMC_CTRL_BASEADDR ⁽⁷⁾	0xFFFFFFFF	Valid Address	MPMC CTRL PLB v4.6 Base Address. Must be 64K aligned.
C_MPMC_CTRL_HIGHADDR ⁽⁷⁾	0x00000000	Valid Address	MPMC CTRL PLB v4.6 High Address.
C_MPMC_CTRL_AWIDTH ⁽⁷⁾	32	32	PLB v4.6 Address Width.
C_MPMC_CTRL_DWIDTH ⁽⁷⁾	64	32, 64, 128	PLB v4.6 Data Width.
C_MPMC_CTRL_NATIVE_DWIDTH ⁽⁷⁾	32	32	PLB v4.6 Native Data Width.
C_MPMC_CTRL_PLB_NUM_MASTERS ⁽⁷⁾	1	0-16	PLB v4.6 Number of masters on the Bus.
C_MPMC_CTRL_PLB_MID_WIDTH ⁽⁷⁾	1	0-4	PLB v4.6 Master ID Width.
C_MPMC_CTRL_P2P ⁽⁷⁾	1	0,1	PLB v4.6 Point-To-Point (P2P) support.
C_MPMC_CTRL_SUPPORT_BURSTS ⁽⁷⁾	0	0,1	PLB v4.6 PIM Burst support.
C_MPMC_CTRL_SMALLEST_MASTER ⁽⁷⁾	32	32, 64, 128	PLB v4.6 Smallest Master on Bus.

Notes:

- When C_ALL_PIMS_SHARE_ADDRESSES is set to 1, C_MPMC_BASEADDR is used for all ports for memory access addressing and C_SDMA_CTRL_BASEADDR is used for all SDMA PIMs (if applicable).
If set to 0, the C_PIMx_BASEADDR, and C_SDMA_CTRLx_BASEADDR parameters are used.
- Valid if C_PM_ENABLE is set to 1.
- Valid when using Static PHY (C_USE_STATIC_PHY = 1).
- C_PIM<Port_Num>_BASETYPE must be set to 0 for unused ports that are outside the number active ports specified by C_NUM_PORTS.
For example, if C_NUM_PORTS = 4, then C_PIM4_BASETYPE, C_PIM5_BASETYPE, C_PIM6_BASETYPE, and C_PIM7_BASETYPE must all be 0.
- Valid when using MIG-based Virtex-4/Virtex-5 DDR/DDR2 PHY.
- Memory calibration simulation times vary based on C_MEM_WIDTH and C_FAMILY:
Virtex4 DDR = 90us
Virtex4 DDR2 = 50us
Virtex5 DDR = 1400us
Virtex5 DDR2 = 100us
- Only valid if the Performance Monitors (PM), Error Correction Code (ECC), Debug Registers, or Static PHY feature is enabled.

Table 1: MPMC System Parameters (Continued)

Parameter Name	Default Value	Allowable Values	Description
C_NUM_IDELAYCTRL ⁽⁵⁾	1	0-16	Number of IDELAYCTRL elements to instantiate.
C_NUM_PORTS	1	1-8	Number of Interface Ports. MPMC GUI automatically sets the value and places the correct parameter in the Microprocessor Hardware Specification (MHS) file.
C_PM_ENABLE	0	0,1	Performance Monitor (PM) enable or disable: 1 = Enable 0 = Disable
C_PM_DC_WIDTH ⁽²⁾	48	1- 64	Sets the width of the PM dead cycle counters
C_PM_GC_CNTR ⁽²⁾	1	0,1	Global Clock Counter enable or disable: 1 = Enable 0 = Disable
C_PM_GC_WIDTH ⁽²⁾	48	1- 64	Sets the width of the PM Global Cycle counter.
C_PM_SHIFT_CNT_BY ⁽²⁾	1	0-3	Specifies the size of the histogram bins used by the Performance Monitors.
C_RD_DATAPATH_TML_MAX_FANOUT	0	0,1,2,4,8	Read Database Timing Management Logic Maximum Register Fanout. Controls the fanout of the PHY layer to the read FIFO data path: 0 = no register is instantiated. 1 = the read data is forwarded from the PHY to eight sets of registers, then forwarded on to each of the read FIFOs. 2 = the read data is forwarded from the PHY to four sets of registers. The outputs of the registers are the forwarded on to a maximum of two read FIFOs ¹ . 4 = the read data is forwarded from the PHY to two sets of registers. The outputs of the registers are the forwarded on to a maximum of four read FIFOs ¹ . 8 = the read data is forwarded from the PHY to a single register. The output of the register is forwarded on to a maximum of eight read FIFOs. Note: 1. Values of 3, 5, 6, 7 are invalid.
C_SKIP_SIM_INIT_DELAY ⁽⁶⁾	0	0,1	For simulation only, allows a shorter initialization sequence.
C_STATIC_PHY_RDDATA_CLK_SEL ⁽³⁾	0	0,1	Sets power-on or reset value of RDDATA_CLK_SEL register.
C_STATIC_PHY_RDDATA_SWAP_RISE ⁽³⁾	0	0,1	Sets power-on or reset value of RDDATA_SWAP_RISE register.
C_STATIC_PHY_RDEN_DELAY ⁽³⁾	5	0-15	Sets power-on or reset value of RDENDELAY register.
C_SPECIAL_BOARD	NONE	S3E_STKIT, S3E_1600E, S3A_STKIT, NONE	Xilinx® special physical layer for Spartan-3x boards.

Table 1: MPMC System Parameters (Continued)

Parameter Name	Default Value	Allowable Values	Description
C_USE_MIG_FLOW	0	0,1	Enables and disables the use of the integrated MIG GUI flow from the MPMC IP Configuration GUI. 0 = Normal MPMC flow without integrated MIG GUI flow. 1 = Enables the use of the integrated MIG GUI Flow for running the MIG GUI from the MPMC IP Configuration GUI. This setting also automatically links the area, timing, and I/O placement constraints from the MIG GUI with the MPMC EDK project. See "Integrated MIG GUI Flow," page 60 for more information.
C_USE_STATIC_PHY	0	0,1	Enables or disables a software controlled interface for the physical layer calibration (Static PHY): 0 = Static PHY Disabled. 1 = Static PHY Enabled. Static PHY is automatically enabled when C_MEM_TYPE = SDRAM.
C_WR_DATAPATH_TML_PIPELINE	1	0,1	Enables or disables the Write Data Path Timing Management: 0 = Write Data Path Timing Management Logic Pipeline Disabled. 1 = Write Data Path Timing Management Logic Pipeline Enabled.
C_WR_TRAINING_PORT ⁽⁵⁾	0	0-7	Specifies the port where the Write FIFO will be used for memory initialization. This value is an automatically calculated parameter that can be over-written. If the parameter is set by the user, it is not calculated.

Notes:

- When C_ALL_PIMS_SHARE_ADDRESSES is set to 1, C_MPMC_BASEADDR is used for all ports for memory access addressing and C_SDMA_CTRL_BASEADDR is used for all SDMA PIMs (if applicable).
If set to 0, the C_PIMx_BASEADDR, and C_SDMA_CTRLx_BASEADDR parameters are used.
- Valid if C_PM_ENABLE is set to 1.
- Valid when using Static PHY (C_USE_STATIC_PHY = 1).
- C_PIM<Port_Num>_BASETYPE must be set to 0 for unused ports that are outside the number active ports specified by C_NUM_PORTS.
For example, if C_NUM_PORTS = 4, then C_PIM4_BASETYPE, C_PIM5_BASETYPE, C_PIM6_BASETYPE, and C_PIM7_BASETYPE must all be 0.
- Valid when using MIG-based Virtex-4/Virtex-5 DDR/DDR2 PHY.
- Memory calibration simulation times vary based on C_MEM_WIDTH and C_FAMILY:
Virtex4 DDR = 90us
Virtex4 DDR2 = 50us
Virtex5 DDR = 1400us
Virtex5 DDR2 = 100us
- Only valid if the Performance Monitors (PM), Error Correction Code (ECC), Debug Registers, or Static PHY feature is enabled.

MPMC Per-port Parameters

The parameters in the following table are inside the MPMC core and can be set on a per-port basis.

Table 2: MPMC Per-port Parameters

Parameter Name	Default Value	Allowable Values	Description
C_PIM<Port_Num>_BASETYPE ⁽⁴⁾	2 (Port 0) 0 (Ports 1-7)	0 - 6	0 = INACTIVE 1 = XCL 2 = PLB v4.6 3 = SDMA 4 = NPI 5 = PPC440MC 6 = VFBC
C_PIM<Port_Num>_SUBTYPE	x	DXCL, DXCL2, IXCL, IXCL2, XCL, IPLB, DPLB, PLB, SDMA, NPI, PPC440MC, VFBC, INACTIVE	Specific Port Interface Type. MPMC GUI sets the value automatically and places the correct parameter in the MHS file. This value is an automatically calculated parameter that can be overwritten; if set by user, it is not auto-calculated.
C_PIM<Port_Num>_B_SUBTYPE ⁽⁹⁾	x	DXCL, DXCL2, IXCL, IXCL2, XCL	Specific Port Interface Type. MPMC GUI sets the value automatically and places the correct parameter in the MHS file. This value is an automatically calculated parameter that can be overwritten; if set by user, it is not auto-calculated.
C_PIM<Port_Num>_BASEADDR ^(1,7)	0xFFFFFFFF	Valid Address	PIM Base Address.
C_PIM<Port_Num>_HIGHADDR ^(1,8)	0x00000000	Valid Address	PIM High Address. MPMC supports a maximum of 2 gigabytes of memory.
C_PIM<Port_Num>_OFFSET ⁽¹⁾	0x00000000	Valid Address	PIM Offset Address.
C_PI<Port_Num>_RD_FIFO_TYPE	BRAM	BRAM, SRL, DISABLED	Read Data Path FIFO type.
C_PI<Port_Num>_WR_FIFO_TYPE ⁽⁶⁾	BRAM	BRAM, SRL, DISABLED	Write Data Path FIFO type.
C_PI<Port_Num>_ADDRACK_PIPELINE ⁽³⁾	1	0,1	AddrAck Pipeline enable.
C_PI<Port_Num>_RD_FIFO_APP_PIPELINE	1	0,1	Read FIFO Port Side pipeline.
C_PI<Port_Num>_RD_FIFO_MEM_PIPELINE ⁽⁴⁾	1	0,1	Read FIFO Memory Side pipeline.
C_PI<Port_Num>_WR_FIFO_APP_PIPELINE	1	0,1	Write FIFO Port Side pipeline.
C_PI<Port_Num>_WR_FIFO_MEM_PIPELINE ⁽⁵⁾	1	0,1	Write FIFO Memory Side pipeline.

Table 2: MPMC Per-port Parameters (Cont'd)

Parameter Name	Default Value	Allowable Values	Description
C_PI<Port_Num>_PM_USED (2,3)	1	0,1	Enable Performance Monitor.
C_PI<Port_Num>_PM_DC_CNTR (2)	1	0,1	Enable Dead Cycle Counter.

Notes:

1. Only valid if C_PIM_BASETYPE is not 4 (NPI) and C_ALL_PIMS_USE_SHARED_ADDRESSES is 0.
2. Only valid if C_PM_ENABLE = 1.
3. If C_PM<Port_Num>_PM_USED is set to 1, then C_PI<Port_Num>_ADDRACK_PIPELINE must be set to 1 to monitor correctly.
4. C_PI<Port_Num>_RD_FIFO_MEM_PIPELINE settings must all be the same from port 0 to port <C_NUM_PORTS-1>. For example, on a four port design ports 0 to 3 must have the same C_PI<Port_Num>_RD_FIFO_MEM_PIPELINE settings.
5. C_PI<Port_Num>_WR_FIFO_MEM_PIPELINE settings must all be the same from port 0 to port <C_NUM_PORTS-1>. For example, on a four port design ports 0 to 3 must have the same C_PI<Port_Num>_WR_FIFO_MEM_PIPELINE settings.
6. Write FIFOs are automatically disabled in an MPMC port that is an IXCL or IPLB subtype. There is no need to manually disable write FIFOs in an IXCL or IPLB configured port.
7. C_PI<Port_Num>_BASEADDR+C_PIM<Port_Num>_OFFSET represents the base physical memory address that the corresponding port is allowed to access. For example, if C_PIM<Port_Num>_OFFSET is 0x00000000, C_PIM<Port_Num>_BASEADDR will represent the physical address of memory. If your total memory size is 0x03FFFFFF, a C_PIM<Port_Num>_BASEADDR value of 0x00000000 will go to physical address 0x00000000. A value of 0x01000000 will go to physical address 0x01000000. A value of 0x04000000 will go to physical address 0x00000000. If you increase the C_PIM<Port_Num>_OFFSET to 0x02000000, a C_PIM<Port_Num>_BASEADDR value of 0x00000000 will go to physical address 0x02000000. A value of 0x01000000 will go to physical address 0x03000000. A value of 0x04000000 will go to physical address 0x02000000.
8. C_PIM<Port_Num>_HIGHADDR+C_PIM<Port_Num>_OFFSET represents the high physical memory address that the corresponding port is allowed to access.
9. Used only for XCL<Port_Num>_B port when C_XCL<Port_Num>_B_IN_USE is set to 1

Personality Interface Module (PIM) Parameters

XCL Design Parameters

Table 3: XCL Design Parameters

Parameter Name	Default Value	Allowable Values	Description
C_XCL<Port_Num>_LINESIZE (1,2)	4	1,4,8,16	Number of words per transaction.
C_XCL<Port_Num>_WRITEXFER (1)	1	0, 1, 2	XCL write transfer type: 0 = No write transfers. 1 = Single write transfers only. 2 = Cacheline transfer only.
C_XCL<Port_Num>_B_LINESIZE (1,2)	4	1,4,8,16	Number of words per transaction.
C_XCL<Port_Num>_B_WRITEXFER (1,2)	1	0, 1, 2	XCL write transfer type: 0 = No write transfers. 1 = Single write transfers only. 2 = Cacheline transfer only.
C_XCL<Port_Num>_B_IN_USE	0	0, 1	XCL B Port Enable. This parameter enables another XCL BUS on the same MPMC Port. 1 = Enable XCL B Port. 0 = Disable XCL B Port.

1. Valid when C_PIM<Port_Num>_BASETYPE = 1 (XCL) only
2. Valid when C_XCL<Port_Num>_B_IN_USE = 1 only

PLB v4.6 PIM Design Parameters

Table 4: PLB v4.6 Design Parameters

Parameter Name	Default Value	Allowable Values	Description
C_SPLB<Port_Num>_AWIDTH ^(2,3)	32	32	PLB Least Significant Address Bus Width.
C_SPLB<Port_Num>_DWIDTH ^(2,3)	64	32,64,128	Width of the PLB Data Bus.
C_SPLB<Port_Num>_NATIVE_DWIDTH ⁽²⁾	64	32,64	Width of the PIM Internal Data Bus.
C_SPLB<Port_Num>_PLB_NUM_MASTERS ^(2,3)	1	1-16	Number of masters that can be connected the PIM.
C_SPLB<Port_Num>_PLB_MID_WIDTH ^(1,2,3)	1	0- 4	PLB Master ID Width. PLB Master ID Bus Width. The value is \log_2 (C_SPLB<Port_Num>_PLB_NUM_MASTERS) with a minimum value of 1.
C_SPLB<Port_Num>_P2P ^(2,3)	1	0,1	Selects Shared Bus or Point-to-Point (P2P) configuration for the PLB slave port: 0 = PLB Shared Bus Connection. 1 = PLB P2P Connection. Must be set to 1 when C_PIM<Port_Num>_SUBTYPE is set to IPLB or DPLB.
C_SPLB<Port_Num>_SUPPORT_BURSTS ^(2,3)	0	0,1	PLB PIM Burst Support: 0 = Single Word transactions. 1 = Single, cacheline, and burst transactions.
C_SPLB<Port_Num>_SMALLEST_MASTER ^(2,3)	32	32,64,128	Width of the smallest Master Data Bus.

1. \log_2 represents a logarithm function of base 2. For example, $\log_2(1)=0$, $\log_2(2)=1$, $\log_2(4)=2$, $\log_2(8)=3$, $\log_2(16)=4$, etc.
2. Valid if C_PIM<Port_Num>_BASETYPE = 2 (SPLB)
3. These parameters are normally calculated by the XPS based on what devices are connected to the PLB bus.

SDMA Design Parameters

Table 5: SDMA Design Parameters

Parameter Name	Default Value	Allowable Values	Description
C_SDMA_CTRL_BASEADDR ^(1,2)	0xFFFFFFFF	Valid Address	SDMA CTRL Shared PLB v4.6 Base Address.
C_SDMA_CTRL_HIGHADDR ^(1,2)	0x00000000	Valid Address	SDMA CTRL Shared PLB v4.6 High Address.
C_SDMA_CTRL<Port_Num>_BASEADDR ^(1,2)	0xFFFFFFFF	Valid Address	SDMA CTRL PLB Base Address.
C_SDMA_CTRL<Port_Num>_HIGHADDR ^(1,2)	0x00000000	Valid Address	SDMA CTRL PLB High Address.
C_SDMA_CTRL<Port_Num>_AWIDTH ^(1,3)	32	32	PLB Address Width.
C_SDMA_CTRL<Port_Num>_DWIDTH ^(1,3)	64	32,64,128	PLB Data Width.
C_SDMA_CTRL<Port_Num>_NATIVE_DWIDTH ^(1,3)	32	32	PLB Native Data Width.
C_SDMA_CTRL<Port_Num>_PLB_NUM_MASTERS ^(1,3)	1	0-16	PLB Number of masters on the Bus.
C_SDMA_CTRL<Port_Num>_PLB_MID_WIDTH ^(1,3)	1	0-4	PLB Master ID Width.
C_SDMA_CTRL<Port_Num>_P2P ^(1,3)	1	0,1	PLB Point-to-Point (P2P) support: 0 = Not Supported. 1 = Supported.

Table 5: SDMA Design Parameters (Cont'd)

Parameter Name	Default Value	Allowable Values	Description
C_SDMA_CTRL<Port_Num>_SUPPORT_BURSTS ^(1,3)	0	0	PLB PIM Burst support: 0 = Not Supported. 1 = Supported.
C_SDMA_CTRL<Port_Num>_SMALLEST_MASTER ^(1,3)	32	32,64,128	PLB Smallest Master on Bus.
C_SDMA<Port_Num>_PRESCALAR ⁽¹⁾	100	0-1023	Interrupt Delay Timer Scale Factor.
C_SDMA<Port_Num>_PI2LL_CLK_RATIO ⁽¹⁾	1	1,2	NPI to LocalLink Clock ratio.
C_SDMA<Port_Num>_COMPLETED_ERR_TX ⁽¹⁾	1	0,1	Transmit complete with error checking. 0 = Disable complete bit error checking. 1 = Enable complete bit error checking.
C_SDMA<Port_Num>_COMPLETED_ERR_RX ⁽¹⁾	1	0,1	Receive complete with error checking. 0 = Disable complete bit error checking. 1 = Enable complete bit error checking.

Notes:

- Valid if C_PIM<Port_Num>_BASETYPE = 3 (SDMA)
- If C_ALL_PIMS_USED_SHARED_ADDRESS is 1, there is one common BASEADDR/HIGHADDR for all SDMAs (C_SDMA_CTRL_BASEADDR); otherwise, each SDMA Port has a unique BASE/HIGHADDR (C_SDMA_CTRL<Port_Num>_BASEADDR).
- These parameters are normally calculated by the XPS based on what devices are connected to the PLB bus.

NPI Design Parameters

Table 6: NPI Design Parameters

Parameter Name	Default Value	Allowable Values	Description
C_PIM<Port_Num>_DATA_WIDTH	64	32,64	PIM Native Data Width.

MIB/PPC440MC Design Parameters

Table 7: MIB/PPC440MC Design Parameters

Parameter Name	Default Value	Allowable Values	Description
C_PPC440MC<Port_Num>_BURST_LENGTH	4	2,4,8	Length of allowable bursts.
C_PPC440MC<Port_Num>_PIPE_STAGES	1	0-2	Number of pipeline stages to insert.

VFBC PIM Design Parameters

Table 8: VFBC Design Parameters

Parameter Name	Default Value	Allowable Values	Description
C_VFBC<Port_Num>_CMD_FIFO_DEPTH	32	1 - x ⁽¹⁾	Depth of the command FIFO in 32-bit words.
C_VFBC<Port_Num>_CMD_AFULL_COUNT	3	0 - C_VFBC<Port_Num>_ CMD_FIFO_DEPTH	Command FIFO almost full threshold.
C_VFBC<Port_Num>_RDWD_FIFO_DEPTH	1024	1 - x ⁽¹⁾	Read/Write FIFO depth in the number of data words (word size is defined by the RDWD_DATA_WIDTH parameter).
C_VFBC<Port_Num>_RDWD_DATA_WIDTH	32	8,16,32,64	Data width in number of bits.
C_VFBC<Port_Num>_ RD_AEMPTY_WD_AFULL_COUNT	3	0 - C_VFBC<Port_Num>_ RDWD_FIFO_DEPTH	Write FIFO Almost Full Threshold and Read FIFO Almost Empty Threshold.

1. As the FIFO depth for each FIFO is increased, the FIFO consumes more block RAMs. The upper limit is constrained by the number of block RAMs available on the FPGA device and the number of block RAMS used.

Memory and Memory Part Parameters

Table 9: Memory and Memory Part Parameters

Parameter Name	Default Value	Allowable Values	Description
C_MEM_TYPE	DDR2	DDR, DDR2, SDRAM	Memory architecture type.
C_MEM_PARTNO ⁽¹⁾	NONE	Database Part Number, Example: "mt4htf3264h-53e", CUSTOM	Specifies the memory part number from database or CUSTOM.
C_MEM_PART_DATA_DEPTH ^(1,8)	16	1, 2, 4, 8, 16, 32, 128, 256, 512, 1024	Discrete memory part data depth in megabits. This parameter currently not used and is reserved for future use.
C_MEM_PART_DATA_WIDTH ⁽¹⁾	16	8,16, 32, 64	Discrete memory part data width.
C_MEM_PART_NUM_BANK_BITS ⁽¹⁾	2	1–4	Number of bank bits on memory part.
C_MEM_PART_NUM_ROW_BITS ⁽¹⁾	13	1–20	Number of row bits on memory part.
C_MEM_PART_NUM_COL_BITS ⁽¹⁾	9	1–20	Number of column bits on memory part.
C_MEM_PART_TRAS ⁽¹⁾	x	Any Integer	tRAS - Minimum ACTIVE-to-PRECHARGE command (ps).
C_MEM_PART_TRASMAX ⁽¹⁾	x	Any Integer	tRAS - Maximum ACTIVE-to-PRECHARGE command (ps).
C_MEM_PART_TRC ⁽¹⁾	x	Any Integer	tRC - Minimum ACTIVE-to-ACTIVE (same bank) command (ps).
C_MEM_PART_TRCD ⁽¹⁾	x	Any Integer	tRCD - Minimum ACTIVE-to-READ or WRITE delay (ps).
C_MEM_PART_TDQSS ^(1,2)	1	1	tDQSS - Positive DQS latching edge to associated clock edge (tCK). This value should be (maximum value - minimum value) rounded up to the nearest integer.
C_MEM_PART_TWR ⁽¹⁾	x	Any Integer	tWR - Minimum write recover time (ps).
C_MEM_PART_TRP ⁽¹⁾	x	Any Integer	tRP - Minimum PRECHARGE command period (ps).
C_MEM_PART_TMRD ⁽¹⁾	x	Any Integer	tMRD - Minimum LOAD MODE command cycle time (tCK).
C_MEM_PART_TRRD ⁽¹⁾	x	Any Integer	tRRD - Minimum ACTIVE bank a to ACTIVE bank b command (ps).
C_MEM_PART_TRFC ⁽¹⁾	x	Any Integer	tRFC - Minimum REFRESH to ACTIVE or REFRESH to REFRESH command interval (ps).
C_MEM_PART_TREFI ⁽¹⁾	x	Any Integer	tREFI - Maximum average periodic REFRESH interval (ps).
C_MEM_PART_TAL ^(1,3)	0	0	tAL - Additive Latency desired (tCK).

Table 9: Memory and Memory Part Parameters (Cont'd)

Parameter Name	Default Value	Allowable Values	Description
C_MEM_PART_TCCD ^(1,3)	x	Any Integer	tCCD - Minimum CAS# to CAS# command delay (tCK).
C_MEM_PART_TWTR ^(1,3)	x	Any Integer	tWTR - Minimum internal WRITE-to-READ command delay (ps).
C_MEM_PART_TRTP ^(1,3)	7500	7500	tRTP - Minimum internal READ to PRECHARGE command delay (ps).
C_MEM_PART_CAS_A_FMAX ^(1,4)	x	Any Integer	Maximum Frequency for the lowest CAS latency.
C_MEM_PART_CAS_A ^(1,4)	x	Any Integer	Lowest CAS latency for this memory part.
C_MEM_PART_CAS_B_FMAX ^(1,4)	x	Any Integer	Maximum Frequency for the next lowest CAS latency (if applicable).
C_MEM_PART_CAS_B ^(1,4)	x	Any Integer	Next Lowest CAS latency for this memory part (if applicable).
C_MEM_PART_CAS_C_FMAX ^(1,4)	x	Any Integer	Maximum Frequency for the next lowest CAS latency (if applicable).
C_MEM_PART_CAS_C ^(1,4)	x	Any Integer	Next Lowest CAS latency for this memory part (if applicable).
C_MEM_PART_CAS_D_FMAX ^(1,4)	x	Any Integer	Maximum Frequency for the next lowest CAS latency (if applicable).
C_MEM_PART_CAS_D ^(1,4)	x	Any Integer	Next Lowest CAS latency for this memory part (if applicable).
C_MEM_DQS_IO_COL	x	any 18-bit value	Used with older MIG versions. A DRC error occurs if this parameter is set by the user. See " Memory Interface Generator (MIG) Based PHY Interface ," page 55 for more information if migrating from an older MPMC/MIG version.
C_MEM_DQ_IO_MS	0x000000 00000000 0000	any 72-bit value	Used with older MIG versions. A DRC error occurs if this parameter is set by the user. See " Memory Interface Generator (MIG) Based PHY Interface ," page 55 for more information if migrating from an older MPMC/MIG version.
C_MEM_CAS_LATENCY ⁽⁵⁾	0	0–9	Auto-calculated Memory CAS latency based on Clock speed.
C_MEM_ODT_TYPE ⁽³⁾	0	0, 1, 2, 3	On-Die Termination Setting (DDR2 only): 0 = Disabled 1 = 75 Ohms 2 = 150 Ohms 3 = Reserved/50 Ohms
C_MEM_REDUCED_DRV	0	0,1	Reduced drive output enable. (DDR/DDR2 only)
C_MEM_REG_DIMM	0	0,1	DIMM is registered.

Table 9: Memory and Part Parameters (Continued)

Parameter Name	Default Value	Allowable Values	Description
C_MPMC_CLK0_PERIOD_PS	10000	1-1000000	MPMC_CLK0 Period (ps). This value is automatically calculated based on what is connected to Port MPMC_CLK0 in XPS (for example a clock_generator output or a signal/port with MHS tag CLK_FREQ = xxxx). The value can be overwritten; if set by the user, it is not calculated.
C_MEM_CLK_WIDTH	1	1-16	Number of external clock pins. This value is an automatically calculated parameter that can be overwritten; if set by the user, it is not calculated.
C_MEM_ODT_WIDTH ⁽³⁾	1	1-16	Number of external ODT Pins. This value is an automatically calculated parameter that can be overwritten; if set by the user, it is not calculated. This must be set to an integer multiple of C_NUM_RANKS * C_NUM_DIMMS.
C_MEM_CE_WIDTH	1	1-16	Number of external chip enable pins. This value is an automatically calculated parameter that can be overwritten; if set by the user, it is not calculated.
C_MEM_CS_N_WIDTH	1	1-16	Number of external chip select pins. This value is an automatically calculated parameter that can be overwritten; if set by the user, it is not calculated. This must be set to an integer multiple of C_NUM_RANKS * C_NUM_DIMMS.
C_MEM_ADDR_WIDTH	13	1-20	Number of external address pins.
C_MEM_BANKADDR_WIDTH	2	1-4	Number of external bank address pins.
C_MEM_DATA_WIDTH ⁽⁸⁾	64	8,16,32,64	Number of external data pins.
C_MEM_BITS_DATA_PER_DQS	8	8	Number of data bits per DQS bit.
C_MEM_DM_WIDTH	8	1,2,4,8	Number of external data mask pins.
C_MEM_DQS_WIDTH ^(2,3)	8	1,2,4,8	Number of external DQS pins.
C_MEM_NUM_DIMMS	1	1	Number of DIMMs. Set to 1 if not using a DIMM. Multiple DIMMS are not supported.
C_MEM_NUM_RANKS ⁽⁷⁾	1	1-2	Number of ranks per DIMM. A value of 2 is not recommended.

Notes:

- These values are auto-updated from the IP Configurator database if C_MEM_PARTNO is set to a part number from the database. If set to CUSTOM, the values must be filled in according to the memory parameters provided by the manufacturer. Unlisted parts can be requested for future versions by opening a Xilinx support WebCase and attaching the new memory datasheet. The database is a Comma Separated Value (CSV) file located at `<MPMC_pcore_location>/data/mpmc_memory_database.csv`.
- DDR Parameter.
- DDR2 Parameter.
- CAS latencies/FMAX pairs should be arranged from Lowest CAS Latency and Slowest Frequency to Highest CAS Latency and Fastest frequency for pairs A-D.
- Non-user, auto-calculated value.
- Valid if C_INCLUDE_ECC_SUPPORT is enabled.
- The use of Multi-Rank designs is strongly discouraged. See the "Additional Information," page 67 for more information.
- SDMA supports the following configurations only:
 - 32- and 64-bit for DDR
 - 64-bit for SDRAM

Table 10: Additional Memory Part Parameters

Parameter Name	Default Value	Allowable Values	Description
Additional Part Parameters			
C_DDR2_DQSN_ENABLE ⁽³⁾	1	0,1	Enables differential DQS (DDR2 Only). Must be set to 0 when C_FAMILY = "spartan3". (Can be set to 1 for other Spartan3x families such as spartan3a, spartan3an, spartan3adsp, spartan3e) Must be set to 1 when using MIG-based Virtex-5 DDR2 PHY.
C_INCLUDE_ECC_SUPPORT	0	0,1	Enables ECC logic. ECC control registers are accessible from MPMC_CTRL interface when enabled.
C_ECC_DEFAULT_ON ⁽⁶⁾	1	0,1	Enables ECC enable register at RST:
C_INCLUDE_ECC_TEST ⁽⁶⁾	0	0,1	Enable or disable ECC Test Functionality and registers: 1 = Enable ECC test functionality/registers. 0 = No ECC test functionality (saves area).
C_ECC_SEC_THRESHOLD ⁽⁶⁾	1	0-4095	Single-bit data error interrupt threshold counter value.
C_ECC_DEC_THRESHOLD ⁽⁶⁾	1	0-4095	Double-bit data error interrupt threshold counter value.
C_ECC_PEC_THRESHOLD ⁽⁶⁾	1	0-4095	Specifies the parity-bit data error interrupt threshold counter value.
C_ECC_DATA_WIDTH ^(5,6)	0	0, 3-8	ECC Data Width (in Bits).
C_ECC_DM_WIDTH ^(5,6)	0	0,1	ECC DM Width.
C_ECC_DQS_WIDTH ^(5,6)	0	0,1	ECC DQS width.

Notes:

- These values are auto-updated from the IP Configurator database if C_MEM_PARTNO is set to a part number from the database. If set to CUSTOM, the values must be filled in according to the memory parameters provided by the manufacturer. Unlisted parts can be requested for future versions by opening a Xilinx support WebCase and attaching the new memory datasheet. The database is a Comma Separated Value (CSV) file located at *<MPMC_pcore_location>/data/mpmc_memory_database.csv*.
- DDR Parameter.
- DDR2 Parameter.
- CAS latencies/FMAX pairs should be arranged from Lowest CAS Latency and Slowest Frequency to Highest CAS Latency and Fastest frequency for pairs A-D.
- Non-user, auto-calculated value.
- Valid if C_INCLUDE_ECC_SUPPORT is enabled.

MPMC I/O Signals

The following tables provide the I/O signals for the MPMC system, memory, and PIMs.

System I/O Signals

Table 11: System I/O Signals

Signal Name	Direction	Init Status	Description
MPMC_Clk0	Input	x	System clock input.
MPMC_Clk90	Input	x	System clock input, phase shifted by 90 degrees. Not used with SDRAM.
MPMC_Clk0_DIV2	Input	x	MPMC_Clk0, divided by 2, clock input. Only valid when using MIG-based Virtex-5 DDR2 PHY.
MPMC_Clk_200MHz ⁽¹⁾	Input	x	200 MHz clock. Connects to IDELAY elements and does not have to be phase or frequency related to MPMC_Clk0. Valid only when using MIG-based Virtex-4/Virtex-5 DDR/DDR2 PHY.
MPMC_Rst	Input	x	System reset input.
MPMC_Clk_Mem ⁽²⁾	Input	x	Memory read data capture clock used by Static PHY; otherwise should be left unconnected
MPMC_Idelayctrl_Rdy_I ⁽¹⁾	Input	Automatically set to 1 if unconnected	This active high input is combined with internal IDELAYCTRL_RDY signals to indicate that the memory initialization can begin.
MPMC_Idelayctrl_Rdy_0 ⁽¹⁾	Output	0	This active high output signals that the internal IDELAYCTRL RDY signals and the MPMC_Idelayctrl_Rdy_I are all high.
MPMC_InitDone	Output	0	This active high signal, when asserted, indicates that the memory initialization has completed successfully. When LOW, the memory is currently being calibrated and configured.
MPMC_ECC_Intr	Output	0	ECC Interrupt: (level sensitive) 1 = Interrupt asserted. 0 = No Interrupt.
MPMC_DCM_PSEN ⁽²⁾	Output	x	Connects to PSEN pin of DCM to allow MPMC Static PHY to change DCM phase.
MPMC_DCM_PSINCDEC ⁽²⁾	Output	x	Connects to PSINCDEC pin of DCM to allow MPMC Static PHY to change DCM phase.
MPMC_DCM_PSDONE ⁽²⁾	Input	x	Connects to PSDONE pin of DCM to allow MPMC Static PHY to change DCM phase.

1. Signals are applicable MIG-based Virtex-4/Virtex-5 DDR/DDR2 PHY only.
2. Signals are applicable when using Static PHY only. This includes the SDRAM PHY.

Memory Signals

SDRAM PHY I/O Signals

Table 12: SDRAM PHY I/O Signals

Signal	Direction	Init Status	Description
SDRAM_Clk	Output	0	Clock to memory.
SDRAM_CE	Output	0	Clock enable. (memory CKE signal)
SDRAM_CS_n	Output	1	Chip select, active-low.
SDRAM_RAS_n	Output	1	Command input.
SDRAM_CAS_n	Output	1	Command input.
SDRAM_WE_n	Output	1	Command input.
SDRAM_BankAddr	Output	x	Bank address.
SDRAM_Addr	Output	x	Row/Column address.
SDRAM_DQ ⁽¹⁾	In/Out	z	Data bits.
SDRAM_DM	Output	0	Data masks.

1. The MHS signal connecting this port and the MHS external port must have the same name. See <http://www.xilinx.com/support/answers/14264.htm>. The "Reference Documents," page 193 has a link to this topic.

Double Data Rate (DDR) and Double Data Rate 2 (DDR2) I/O Signals

DDR I/O Signals

Table 13: DDR I/O Signals

Signal Name ⁽¹⁾	Direction	Init Status	Description
DDR_Clk	Output	0	Clock to memory.
DDR_Clk_n	Output	1	Inverted clock to memory.
DDR_CE	Output	0	1 = Clock enabled. (memory CKE signal)
DDR_CS_n	Output	1	0 = Chip select enabled.
DDR_RAS_n	Output	1	Command input.
DDR_CAS_n	Output	1	Command input.
DDR_WE_n	Output	1	Command input.
DDR_BankAddr	Output	x	Bank address.
DDR_Addr	Output	x	Row/Column address.
DDR_DQ ⁽³⁾	In/Out	x	Data.
DDR_DM	Output	x	Data mask outputs.
DDR_DQS ⁽³⁾	In/Out	x	Data strobe.
DDR_DQS_DIV_O ⁽²⁾	Output	x	Timing loop signal.
DDR_DQS_DIV_I ⁽²⁾	Input	x	Timing loop signal.

1. For detailed signal descriptions, refer to device-specific data sheets.
 2. Required when using MIG-based Spartan-3 (3/3A/3AN/3ADSP/3E) PHY.
 3. The MHS signal connecting this port and the MHS external port must have the same name. See <http://www.xilinx.com/support/answers/14264.htm>. The "Reference Documents," page 193 has a link to this topic.

DDR2 I/O Signals

Table 14: DDR2 I/O Signals

Signal Name ⁽¹⁾	Direction	Init Status	Description
DDR2_Clk	Output	0	Clock to memory.
DDR2_Clk_n	Output	1	Inverted clock to memory.
DDR2_CE	Output	0	1 = Clock enabled.
DDR2_CS_n	Output	1	0 = Chip select enabled.
DDR2_RAS_n	Output	1	Command input.
DDR2_CAS_n	Output	1	Command input.
DDR2_WE_n	Output	1	Command input.
DDR2_BankAddr	Output	x	Bank address.
DDR2_Addr	Output	x	Row/Column address.
DDR2_DQ ⁽³⁾	In/Out	x	Data.
DDR2_DM	Output	x	Data mask outputs.
DDR2_DQS ⁽³⁾	In/Out	x	Data Strobe.
DDR2_DQS_n ⁽⁴⁾	In/Out	x	Inverted Data Strobe.
DDR2_DQS_DIV_O ⁽²⁾	Output	x	Timing loop signal.
DDR2_DQS_DIV_I ⁽²⁾	Input	x	Timing loop signal.
DDR2_ODT	Output	0	On-Die-Termination signal. Care must be taken when connecting these pins to your memory when you have more than one rank; there is a direct relationship to the DDR2_CS_n pins.

1. For detailed signal descriptions, refer to device-specific data sheets.
2. Required when using MIG-based Spartan-3/3A/3AN/3ADSP/3E DDR/DDR2 PHY.
3. The MHS signal connecting this port and the MHS external port must have the same name. See <http://www.xilinx.com/support/answers/14264.htm>. The "Reference Documents," page 193 has a link to this topic.
4. Required when differential DQS is enabled (C_DDR2_DQSN_ENABLE = 1).

PIM I/O Signals

XCL PIM I/O Signals

Table 15: XCL PIM I/O Signals⁽¹⁾

Signal Name	Direction	Init Status	Description
FSL<Port_Num>_M_Clk	Input	x	Clock
FSL<Port_Num>_M_Write	Input	x	Write enable signal indicating that data is being written to the output FSL.
FSL<Port_Num>_M_Data	Input	x	Data value written to the output FSL.
FSL<Port_Num>_M_Control	Input	x	Control bit value written to the output FSL.
FSL<Port_Num>_M_Full	Output	0	Full Bit indicating output FSL FIFO is full when set.
FSL<Port_Num>_S_Clk	Input	x	Clock
FSL<Port_Num>_S_Read	Input	x	Read acknowledge signal indicating that data has been read from the input FSL.
FSL<Port_Num>_S_Data	Output	x	Data value currently available at the top of the input FSL.
FSL<Port_Num>_S_Control	Output	0	Control Bit value currently available at the top of the input FSL.
FSL<Port_Num>_S_Exists	Output	0	Flag indicating that data exists in the input FSL.

- When C_XCL<Port_Num>_B in use is enabled the B port will have the same signals names with an _B appended after the <Port_Num>.

PLB v4.6, SDMA_CTRL, and MPMC_CTRL PIM I/O Signals

MPMC contains Slave PLB ports for the PLB PIM, SDMA Control Registers (SDMA_CTRL), and MPMC Control Register (MPMC_CTRL) interfaces. Each of these slave PLB interfaces have the same set of signal names with different prefixes on the Port Bus Names. The <Bus_Name> prefixes are as follows:

- SDMA Control Registers (SDMA_CTRL) for Ports 0 to 7: SDMA_CTRL<Port_Num>_
 - SDMA_CTRL is valid if C_PIM<Port_Num>_BASETYPE = 3
- MPMC Control Registers (MPMC_CTRL): MPMC_CTRL
 - MPMC_CTRL is valid if PM, ECC, Debug Registers, or Static PHY is enabled
- MPMC Slave PLB v4.6 PIM: SPLB<Port_Num>
 - SPLB<Port_Num> is valid if C_PIM<Port_Num>_BASETYPE = 2

The following table lists the available signals for SDMA_CTRL, MPMC_CTRL, and PLB v4.6 PIM (SPLB). Replace <Bus_Name> with the appropriate bus prefix.

Table 16: SDMA_CTRL, MPMC_CTRL, and PLB v4.6 (SPLB) PIM I/O Signals

Signal Name	Direction	Init Status	Description
<Bus_Name>_Clk	Input	x	Bus Clock.
<Bus_Name>_Rst	Input	x	PLB reset, active high.
<Bus_Name>_PLB_ABus	Input	x	PLB address bus.
<Bus_Name>_PLB_PAVValid	Input	x	PLB primary address valid.
<Bus_Name>_PLB_SAVValid	Input	x	PLB secondary address valid.
<Bus_Name>>_PLB_masterID	Input	x	PLB current master identifier.
<Bus_Name>_PLB_RNW	Input	x	PLB read not write.

Table 16: SDMA_CTRL, MPMC_CTRL, and PLB v4.6 (SPLB) PIM I/O Signals (Cont'd)

Signal Name	Direction	Init Status	Description
<Bus_Name>_PLB_BE	Input	x	PLB byte enables.
<Bus_Name>_PLB_UABus	Input	x	PLB size of requested transfer.
<Bus_Name>_PLB_rdPrim	Input	x	PLB secondary to primary read request indicator.
<Bus_Name>_PLB_wrPrim	Input	x	PLB secondary to primary write request indicator.
<Bus_Name>_PLB_abort	Input	x	PLB abort bus request.
<Bus_Name>_PLB_busLock	Input	x	PLB bus lock.
<Bus_Name>_PLB_MSize	Input	x	PLB data bus width indicator.
<Bus_Name>_PLB_size	Input	x	PLB size of requested transfer.
<Bus_Name>_PLB_type	Input	x	PLB transfer type.
<Bus_Name>_PLB_lockErr	Input	x	PLB lock error indicator.
<Bus_Name>_PLB_wrPendReq	Input	x	PLB pending write bus request indicator.
<Bus_Name>_PLB_wrPendPri	Input	x	PLB pending write request priority.
<Bus_Name>_PLB_rdPendReq	Input	x	PLB read bus request indicator.
<Bus_Name>_PLB_rdPendPri	Input	x	PLB read bus request priority.
<Bus_Name>_PLB_reqPri	Input	x	PLB current request priority.
<Bus_Name>_PLB_TAttribute	Input	x	PLB transfer attribute bus.
<Bus_Name>_PLB_rdBurst	Input	x	PLB read burst transfer indicator.
<Bus_Name>_PLB_wrBurst	Input	x	PLB burst write transfer indicator.
<Bus_Name>_PLB_wrDBus	Input	x	PLB write data bus.
<Bus_Name>_SI_addrAck	Output	0	Slave address acknowledge.
<Bus_Name>_SI_SSize	Output	0	Slave data bus size.
<Bus_Name>_SI_wait	Output	0	Slave wait indicator.
<Bus_Name>_SI_rearbitrate	Output	0	Slave rearbitrate bus indicator.
<Bus_Name>_SI_wrDAck	Output	0	Slave write data acknowledge.
<Bus_Name>_SI_wrComp	Output	0	Slave write transfer complete indicator.
<Bus_Name>_SI_wrBTerm	Output	0	Slave terminate write burst transfer.
<Bus_Name>_SI_rdDBus	Output	0	Slave read data bus.
<Bus_Name>_SI_rdWdAddr	Output	0	Slave read word address.
<Bus_Name>_SI_rdDAck	Output	0	Slave read data acknowledge.
<Bus_Name>_SI_rdComp	Output	0	Slave read transfer complete indicator.
<Bus_Name>_SI_rdBTerm	Output	0	Slave terminate read burst transfer.
<Bus_Name>_SI_MBusy	Output	0	Slave busy indicator.
<Bus_Name>_SI_MRdErr	Output	0	Slave read error indicator.
<Bus_Name>_SI_MWrErr	Output	0	Slave write error indicator.
<Bus_Name>_SI_MIRQ	Output	0	Slave interrupt indicator.

SDMA LocalLink I/O Signals

Table 17: SDMA LocalLink Interface Signals

Signal Name	Direction	Init Status	Description
LocalLink System Interface			
SDMA<Port_Num>_Clk	Input	x	LLink_Clk
Transmit LocalLink Interface			
SDMA<Port_Num>_TX_D(0:31)	Output	0	Transmit LocalLink Data Bus.
SDMA<Port_Num>_TX_Rem(0:3)	Output	1	Transmit LocalLink Remainder Bus.
SDMA<Port_Num>_TX_SOF	Output	1	Transmit LocalLink Start of Frame.
SDMA<Port_Num>_TX_EOF	Output	1	Transmit LocalLink End of Frame.
SDMA<Port_Num>_TX_SOP	Output	1	Transmit LocalLink Start of Payload.
SDMA<Port_Num>_TX_EOP	Output	1	Transmit LocalLink End of Payload.
SDMA<Port_Num>_TX_Src_Rdy	Output	1	Transmit LocalLink Source Ready.
SDMA<Port_Num>_TX_Dst_Rdy	Input	x	Transmit LocalLink Destination Ready.
Receive LocalLink Interface			
SDMA<Port_Num>_RX_D(0:31)	Input	x	Receive LocalLink Data Bus.
SDMA<Port_Num>_RX_Rem(0:3)	Input	x	Receive LocalLink Remainder Bus.
SDMA<Port_Num>_RX_SOF	Input	x	Receive LocalLink Start of Frame.
SDMA<Port_Num>_RX_EOF	Input	x	Receive LocalLink End of Frame.
SDMA<Port_Num>_RX_SOP	Input	x	Receive LocalLink Start of Payload.
SDMA<Port_Num>_RX_EOP	Input	x	Receive LocalLink End of Payload.
SDMA<Port_Num>_RX_Src_Rdy	Input	x	Receive LocalLink Source Ready.
SDMA<Port_Num>_RX_Dst_Rdy	Output	1	Receive LocalLink Destination Ready.
SDMA System Interface			
SDMA<Port_Num>_Rx_IntOut	Output	0	Receive interrupt output.
SDMA<Port_Num>_Tx_IntOut	Output	0	Transmit interrupt output.
SDMA<Port_Num>_RstOut	Output	0	Soft Reset Acknowledge.

NPI PIM I/O Signals

The NPI PIM runs only at a 1:1 clock ratio to the MPMC memory clock (PORT_MPMC_Clk0); consequently there is no clock input for this interface.

Table 18: NPI PIM I/O Signals

Signal Name	Direction	Init Status	Description
Address Phase Related Input Ports			
PIM<Port_Num>_Addr	Input	x	Indicates the starting address of a particular request. Only valid when PIM<Port_Num>_AddrReq is valid. See the "Address Path," page 48 for address alignment requirements.
PIM<Port_Num>_AddrReq	Input	x	This active high signal indicates that NPI is ready for MPMC to arbitrate an address request. This request cannot be aborted. Must be asserted until PIM<Port_Num>_AddrAck is asserted. See "NPI Design Restrictions and Recommendations," page 155 for additional restrictions.
PIM<Port_Num>_RNW	Input	x	Read/Not Write: 0 = Request is a Write request. 1 = Request is a Read request. Only valid when PIM<Port_Num>_AddrReq is valid.
PIM<Port_Num>_Size	Input	x	Indicates the transfer type of the request: <ul style="list-style-type: none"> • 0x0 = Word transfers (32-bit NPI only) • 0x0 = Double-word transfers (64-bit NPI only) • 0x1 = 4-word cache-line transfer • 0x2 = 8-word cache-line transfers • 0x3 = 16-word burst transfers • 0x4 = 32-word burst transfers • 0x5 = 64-word burst transfers (Not available in all configurations. Available configurations are described in "Restrictions on 64-Word Burst Transfers," page 156.) • Only valid when PIM<Port_Num>_AddrReq is valid.
PIM<Port_Num>_RdModWr	Input	x	This active high signal indicates that if the request is a write, MPMC should do a read/modify/write. Only valid when PIM<Port_Num>_AddrReq is valid. Only valid when C_INCLUDE_ECC_SUPPORT is set to 1. This is required to be set to 1, if: <ul style="list-style-type: none"> • The total transfer size specified by PIM<Port_Num>_Size * 32(bits/word) is less than C_MEM_DATA_WIDTH * 4 (beats/burst), to satisfy the constant memory burst length of 4. or the PIM<Port_Num>_WrFIFO_BE bits for the transfer are not guaranteed to be 1, because MPMC ECC does not use data mask (DM) signals.

Table 18: NPI PIM I/O Signals (Cont'd)

Signal Name	Direction	Init Status	Description
Other Outputs			
PIM<Port_Num>_InitDone	Output	0	1 indicates that initialization is complete and that FIFOs are available for use. Do not assert PIM<Port_Num>_WrFIFO_Push or PIM<Port_Num>_RdFIFO_Pop until PIM<Port_Num>_InitDone is equal to 1.
Address Phase Related Output Ports			
PIM<Port_Num>_AddrAck	Output	0	This active high signal indicates that MPMC has begun arbitration for address request. Valid for one cycle of MPMC_Clk0. PIM<Port_Num>_AddrReq must be deasserted on the next cycle of MPMC_Clk0 unless NPI is requesting a new transfer.
Write Data Phase Related Input Ports			
PIM<Port_Num>_WrFIFO_Data	Input	x	Data to be pushed into MPMC write FIFOs. Only valid with PIM<Port_Num>_WrFIFO_Push. Data is little-endian as shown in Figure 8, page 78 .
PIM<Port_Num>_WrFIFO_BE	Input	x	Indicates which bytes of PIM<Port_Num>_WrFIFO_Data to write. Only valid with PIM<Port_Num>_WrFIFO_Push.
PIM<Port_Num>_WrFIFO_Push	Input	x	This active high signal indicates push PIM<Port_Num>_WrFIFO_Data into write FIFOs. Must be asserted for one cycle of MPMC_Clk0. Cannot be asserted while PIM<Port_Num>_InitDone is 0. Cannot be asserted while PIM<Port_Num>_WrFIFO_AlmostFull is asserted. Can be asserted before, after, or during the address phase unless MPMC is configured in one of several special cases. See the "NPI Design Restrictions and Recommendations," page 155 .
PIM<Port_Num>_WrFIFO_Flush	Input	x	Reserved. Drive with 0.
Write Data Phase Related Output Ports			
PIM<Port_Num>_WrFIFO_Empty	Output	1	This active high signal indicates that there are less than C_MEM_DATA_WIDTH bits of data in the write FIFO.
PIM<Port_Num>_WrFIFO_AlmostFull	Output	0	This active high signal indicates that PIM<Port_Num>_WrFIFO_Push cannot be asserted on the next cycle of MPMC_Clk0. This signal is only asserted when using SRL FIFOs. If BRAM FIFOs are used, the PIM cannot allow more than 1024 bytes of data to be pushed into the FIFOs.

Table 18: NPI PIM I/O Signals (Cont'd)

Signal Name	Direction	Init Status	Description
Read Data Phase Related Input Ports			
PIM<Port_Num>_RdFIFO_Pop	Input	x	This active high signal indicates that read FIFO fetch the next value of PIM<Port_Num>_RdFIFO_Data. Must be asserted for one cycle of MPMC_Clk0. Cannot be asserted while PIM<Port_Num>_InitDone is 0. Cannot be asserted while PIM<Port_Num>_RdFIFO_Empty is asserted. See information in PIM<Port_Num>_RdFIFO_RdFIFO_Latency to know when PIM<Port_Num>_RdFIFO_Data is valid.
PIM<Port_Num>_RdFIFO_Flush	Input	x	This active high signal indicates that the read FIFO flags should be reset. Must be asserted for one cycle of MPMC_Clk0. Caution! Only assert this signal when PIM<Port_Num>_RdFIFO_Empty is 1 and no reads are outstanding to prevent problems with the read FIFO address counters. If it is Flush asserted when multiple read address requests are acknowledged, but where the data phases corresponding to the address phases have not completed, there is a possibility that MPMC is, or will be, in the process of pushing read data from the second address phase into the FIFOs. If the FIFO flags are reset during this time, the FIFO address counters could obtain an unexpected value, putting MPMC in an unstable state; risking either memory errors or the PIM going into a state of deadlock.
Read Data Phase Related Output Ports			
PIM<Port_Num>_RdFIFO_Data	Output	0	Data to be popped out of MPMC read FIFOs. Only valid a certain number of cycles after PIM<Port_Num>_RdFIFO_Push is asserted, and/or PIM<Port_Num>_RdFIFO_Empty is deasserted, as specified by PIM<Port_Num>_RdFIFO_Latency. Data is little-endian as shown in Figure 8, page 78 .
PIM<Port_Num>_RdFIFO_RdWdAddr	Output	0	Indicates the word of a cacheline transfer to which PIM<Port_Num>_RdFIFO_Data corresponds. Only valid a certain number of cycles after PIM<Port_Num>_RdFIFO_Push is asserted, as specified by PIM<Port_Num>_RdFIFO_Latency. Counts by 1 with a 32-bit NPI; counts by 2 with a 64-bit NPI.
PIM<Port_Num>_RdFIFO_Empty	Output	1	When this active high signal, is de-asserted, (0), it indicates that enough data is in the read FIFOs to assert PIM<Port_Num>_RdFIFO_Pop.

Table 18: NPI PIM I/O Signals (Cont'd)

Signal Name	Direction	Init Status	Description
PIM<Port_Num>_RdFIFO_Latency	Output	0, 1, 2	<p>Indicates the number of cycles from the time PIM<Port_Num>_RdFIFO_Pop is asserted and/or PIM<Port_Num>_RdFIFO_Empty is deasserted until PIM<Port_Num>_RdFIFO_Data and PIM<Port_Num>_RdFIFO_RdWdAddr are valid</p> <ul style="list-style-type: none"> 0 = PIM<Port_Num>_RdFIFO_Data and PIM<Port_Num>_RdFIFO_RdWdAddr are valid in the same cycle as the assertion of PIM<Port_Num>_RdFIFO_Pop. 1 = PIM<Port_Num>_RdFIFO_Data and PIM<Port_Num>_RdFIFO_RdWdAddr are valid in the cycle following the assertion of PIM<Port_Num>_RdFIFO_Pop. 2 = PIM<Port_Num>_RdFIFO_Data and PIM<Port_Num>_RdFIFO_RdWdAddr are valid two cycles following the assertion of PIM<Port_Num>_RdFIFO_Pop. <p>This is a constant value for a particular MPMC configuration. Because it is not possible to pass a parameter from one processor core to another, this value is provided as a port. This is a static signal.</p>

PPC440MC PIM I/O Signals

Table 19: PPC440MC PIM I/O Signal Description

Signal Name	Direction	Initial Status	Description
PPC440MC<Port_Num>_MIMCReadNotWrite	Input	x	PPC440MC read not write signal.
PPC440MC<Port_Num>_MIMCAddress[0:35] ⁽¹⁾	Input	x	PPC440MC address bus.
PPC440MC<Port_Num>_MIMCAddressValid	Input	x	PPC440MC address valid identifier.
PPC440MC<Port_Num>_MIMCWriteData[0:127]	Input	x	PPC440MC write data bus.
PPC440MC<Port_Num>_MIMCWriteDataValid	Input	x	PPC440MC write data valid identifier.
PPC440MC<Port_Num>_MIMCByteEnable[0:15]	Input	x	PPC440MC byte enables.
PPC440MC<Port_Num>_MCMIReadData[0:127]	Output	0	PIM read data bus.
PPC440MC<Port_Num>_MCMIReadDataValid	Output	0	PIM read data valid.
PPC440MC<Port_Num>_MCMIAAddrReadytoAccept	Output	0	PIM ready to accept address indicator.

1. MPMC only supports 32 bits [4:35] of address.

VFBC PIM I/O Signals

Table 20: VFBC PIM I/O Signals

Port Name	Direction	Init Status	Description
VFBC Command Interface			
VFBC<Port_Num>_Cmd_Clk	Input	x	Command Clock. Can be asynchronous from the MPMC_Clk0.
VFBC<Port_Num>_Cmd_Reset ^(1, 2)	Input	x	Command Reset (Active High).
VFBC<Port_Num>_Cmd_Data[31:0]	Input	x	Command Data (See Table 73, page 147 for more information on the Command Packet Data Structure.)
VFBC<Port_Num>_Cmd_Write	Input	x	Command Write. The command words are pushed onto the command FIFO when this signal is high.
VFBC<Port_Num>_Cmd_End	Input	x	Command End. When high, the command word currently being written is the last command word in the command. Used to terminate a command early for non-2D transfers. Command word 1 is the only valid command word to provide the End signal. This signal is usually tied low.
VFBC<Port_Num>_Cmd_Full	Output	1	Command Fifo Full. Active high only when the Command FIFO is full.
VFBC<Port_Num>_Cmd_Almost_Full	Output	1	Command Fifo Almost Full. High only when the Command FIFO is almost full. Controlled by the CMD0_AFULL_CNT parameter.
VFBC Write Data Interface			
VFBC<Port_Num>_Wd_Clk	Input	x	Write Data FIFO Clock. Can be asynchronous from the MPMC_Clk0.
VFBC<Port_Num>_Wd_Reset ^(1, 2)	Input	x	Write Data FIFO Reset. (Active High) When asserted, this command: <ul style="list-style-type: none"> Flushes the Write Data FIFO. Clears the current Write Command from the Command FIFO. Resetting the Write Data FIFO returns the internal read/write FIFO pointers to zero. The current write command is also removed from the command FIFO even if the command has not completed.
VFBC<Port_Num>_Wd_Flush ^(1, 2)	Input	x	Write Data FIFO Flush. (Active High) When asserted this command returns the internal read/write FIFO pointers to zero. Unlike a FIFO reset, the current write command is kept active in the command FIFO.
VFBC<Port_Num>_Wd_Write	Input	x	Write Data FIFO Push (Active High)
VFBC<Port_Num>_Wd_Data [C_VFBC<Port_Num>_RDWD_DATA_WIDTH-1:0]	Input	x	Write Data FIFO Data. Must be valid when VFBC<Port_Num>_Wd_Write is High.
VFBC<Port_Num>_Wd_DataByteEn [C_VFBC<Port_Num>_WRDWD_DATA_WIDTH/8-1:0]	Input	x	Reserved for Write Data FIFO Byte Enables. This input is currently not used but included for compatibility with future VFBC PIM versions.

Table 20: VFBC PIM I/O Signals (Cont'd)

Port Name	Direction	Init Status	Description
VFBC<Port_Num>_Wd_End_Burst	Input	x	Burst End. Used only when the transfer is not a multiple of the burst size. If the transfer ends on a non 32-word boundary, this signal must be asserted high during the last word transferred. This signal is usually tied low for aligned transfers.
VFBC Read Data Interface			
VFBC<Port_Num>_Rd_Clk	Input	x	Read Data FIFO Clock: Can be asynchronous from the MPMC_Clk0 Clock.
VFBC<Port_Num>_Rd_Reset ^(1, 2)	Input	x	Read Data FIFO Reset (Active High). When asserted, this command: <ul style="list-style-type: none"> • Flushes the Read Data FIFO. • Clears the current Read command from the command FIFO. Resetting the Read Data FIFO returns the internal read/write FIFO pointers to zero. The current write command is also removed from the command FIFO even if the command has not completed.
VFBC<Port_Num>_Rd_Flush ^(1, 3)	Input	x	Read Data FIFO Flush (Active High). Asserting this command returns the internal read/write FIFO pointers to zero. Unlike a FIFO reset, the current read command is kept active in the command FIFO.
VFBC<Port_Num>_Rd_Read	Input	x	Read Data FIFO Pop (Active High).
VFBC<Port_Num>_Rd_End_Burst	Input	x	Burst End. Used only when the transfer is not a multiple of the burst size. If the transfer ends on a non 32-word boundary, this signal must be asserted high during the last word transferred. This signal is usually tied low.
VFBC<Port_Num>_Rd_Data [C_VFBC<Port_Num>_RDWD_DATA_WIDTH-1:0]	Output	x	Read Data FIFO Data. The data is valid one clock cycle after when the VFBC<Port_Num>_Rd_Read is High.
VFBC<Port_Num>_Rd_Empty	Output	1	Read Data Fifo Empty. Active high only when the read data FIFO is empty.
VFBC<Port_Num>_Rd_Almost_Empty	Output	1	Read Data Fifo Almost Empty. Active high only when the read data FIFO is almost empty. Controlled by the C_VFBC<Port_Num>_RD_AEMPTY_WD_AFULL_COUNT parameter.

1. The VFBC Reset and Flush inputs must be held high for at least two MPMC_Clk0 cycles. Because these inputs could be controlled from a different clock domain than the MPMC_Clk0, the relative frequency of the reset/flush clock domain must be taken into account when determining the number of clock cycles to assert the reset or flush and to wait after reset or flush. The following equation is used to determine the number of clock cycles to hold the reset or flush input high: $2 * (VFBC_Clk_Freq / MPMC_Clk0_Freq)$
2. After reset, there should not be any accesses to the VFBC interfaces for 6 MPMC_Clk cycles.
3. After flush, there should not be any accesses to the VFBC interfaces for 6 MPMC_Clk cycles.

MPMC Parameter and Port Dependencies

Table 21: MPMC Dependencies

Parameter Name	Affects Signal	Relationship Description
C_MEM_ADDR_WIDTH	DDR2_Addr DDR_Addr SDRAM_Addr	Width of address to memory.
C_MEM_BANKADDR_WIDTH	DDR2_BankAddr DDR_BankAddr SDRAM_BankAddr	Width of bank address to memory.
C_MEM_CE_WIDTH	DDR2_CE DDR_CE SDRAM_CE	Number of clock enable outputs.
C_MEM_CLK_WIDTH	DDR2_Clk DDR2_Clk_n DDR_Clk DDR_Clk_n SDRAM_Clk	Number of clock/inverted/clock pair outputs.
C_MEM_CS_N_WIDTH	DDR2_CS_n DDR_CS_n SDRAM_CS_n	Number of chip select output.
C_MEM_DATA_WIDTH C_ECC_DATA_WIDTH	DDR2_DQ DDR_DQ SDRAM_DQ	Width of data at memory interface.
C_MEM_DM_WIDTH C_ECC_DM_WIDTH	DDR2_DM DDR_DM SDRAM_DM	Width of data mask bits at memory interface.
C_MEM_DQS_WIDTH C_ECC_DQS_WIDTH	DDR2_DQS DDR2_DQS_n DDR_DQS	Width of data strobe bits at memory interface.
C_MEM_ODT_WIDTH	DDR2_ODT	Width of ODT bits to memory.
C_DDR2_DQSN_ENABLE	DDR2_DQS_n	Controls visibility of the differential DQS_n signal.
C_NUM_PORTS	PIM<Port_Num>_*	Determines number of ports attached to MPMC.
C_INCLUDE_ECC_SUPPORT	MPMC_ECC_Intr	Interrupt output available only if C_INCLUDE_ECC_SUPPORT is 1.
C_FAMILY C_USE_STATIC_PHY	MPMC_Clk_Mem MPMC_DCM_PSEN MPMC_DCM_PSINC_DEC MPMC_DCM_PS_DONE	Signals are available only if C_USE_STATIC_PHY is 1.
C_FAMILY C_USE_STATIC_PHY C_MEM_TYPE	MPMC_Clk_200MHz MPMC_Idelayctrl_Rdy_I MPMC_Idelayctrl_Rdy_O	These signals are only available if C_FAMILY is virtex4 or virtex5, and C_USE_STATIC_PHY is 0 and C_MEM_TYPE is DDR or DDR2.
C_MEM_TYPE	SDRAM_* DDR_* DDR2_*	Only one set of these ports are available depending on C_MEM_TYPE setting of SDRAM, DDR, or DDR2.
C_FAMILY C_USE_STATIC_PHY C_MEM_TYPE	MPMC_Clk0_DIV2	This signal is only available when C_FAMILY is virtex5, C_USE_STATIC_PHY is 0, and C_MEM_TYPE is DDR2.

PLB v4.6 Bus Parameter and Port Dependencies

The following table lists the parameter and port dependencies for the slave PLB PIM, as well as for the slave PLB control interfaces on the SDMA and MPMC. The slave PLB bus names on the SDMA and MPMC are `SDMA_CTRL` and `MPMC_CTRL`, respectively. See the "MPMC I/O Signals," page 15 for parameter prefix options.

Table 22: PLB v4.6 PIM Dependencies

Parameter	Affects	Relationship Description
<code>C_SPLB<Port_Num>_SUPPORT_BURSTS</code>	<code>C_PIM<Port_Num>_SUBTYPE</code>	<code>C_PIM<Port_Num>_SUBTYPE</code> must be set to PLB when <code>C_SPLB<Port_Num>_SUPPORT_BURSTS = 1</code> if the desired PIM must support single, cacheline, and burst transactions.
		<code>C_PIM<Port_Num>_SUBTYPE</code> must be set to PLB when <code>C_SPLB<Port_Num>_SUPPORT_BURSTS = 0</code> if the desired PIM must support single transactions only.
<code>C_SPLB<Port_Num>_SMALLEST_MASTER</code>	<code>C_SPLB<Port_Num>_NATIVE_DWIDTH</code>	See Table 69, page 139.
<code>C_PIM<Port_Num>_SUBTYPE</code>	<code>C_SPLB<Port_Num>_SUPPORT_BURSTS</code>	<code>C_SPLB<Port_Num>_SUPPORT_BURSTS</code> must be set to 1 when <code>C_PIM<Port_Num>_SUBTYPE = PLB</code> for the PIM to support single, cacheline, and burst transactions.
		<code>C_SPLB<Port_Num>_SUPPORT_BURSTS</code> must be set to 0 when <code>C_PIM<Port_Num>_SUBTYPE = PLB</code> for the PIM to support single transactions only.
	<code>C_SPLB<Port_Num>_NATIVE_DWIDTH</code>	When <code>C_PIM<Port_Num>_SUBTYPE = PLB</code> , <code>C_SPLB<Port_Num>_NATIVE_DWIDTH</code> can be 32 or 64. When <code>C_PIM<Port_Num>_SUBTYPE = DPLB</code> or <code>IPLB</code> , <code>C_SPLB<Port_Num>_NATIVE_DWIDTH</code> must be set to 64.
<code>C_PIM<Port_Num>_OFFSET</code>	<code>SPLB<Bus_Name>_PLB_ABus</code>	Access to memory will be at address the of <code>SPLB<Port_Num>_ABus</code> plus <code>C_PIM<Port_Num>_OFFSET</code> .
Dependencies Applying to all Slave Ports		
<code><Bus_Name>_AWIDTH</code>	<code><Bus_Name>_PLB_ABus</code>	Width of the PLB Address Bus.
<code>C_<Bus_Name>_DWIDTH</code>	<code><Bus_Name>_PLB_wrDBus</code>	Width of the PLB Write Data Bus.
	<code><Bus_Name>_SI_rdDBus</code>	Width of the PLB Read Data Bus.
	<code><Bus_Name>_PLB_BE</code>	<code><Bus_Name>_PLB_BE = C_<Bus_Name>_DWIDTH/8</code> .
<code>C_<Bus_Name>_MID_WIDTH</code>	<code><Bus_Name>_PLB_masterID</code>	Width of the PLB Master ID Bus.

NPI Parameter and Port Dependencies

Table 23: NPI Parameter and Port Dependencies

Parameter Name	Affects Signals	Relationship Description
C_PIM<Port_Num>_DATA_WIDTH	PIM<Port_Num>_WrFIFO_Data PIM<Port_Num>_WrFIFO_BE PIM<Port_Num>_RdFIFO_Data	Width of the data at each port and corresponding byte enable.

Control and Status Registers

The MPMC pcore might contain various status and control registers depending on the configuration options. These are all controlled through a single PLB v.46 interface designated MPMC_CTRL.

Additionally, each SDMA PIM also contains its own control and status register PLB v.46 interface to control the operation of DMA. The MPMC_CTRL interface is composed of the following sections:

- [ECC Register Summary](#)
- [Static PHY Register Summary](#)
- [MIG PHY Debug Registers](#)
- [MPMC Status Register](#)
- [Performance Monitor Register Summary](#)
- [SDMA Register Summary](#)

ECC Register Summary

The following registers are available only when ECC is enabled. Refer to "ECC Registers," page 88 for detailed ECC register information.

Table 24: ECC Register Descriptions

MPMC_CTRL Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
ECC Core				
C_MPMC_CTRL_BASEADDR + 0x0	ECCC ⁽¹⁾	R/W ⁽⁶⁾	00000000 ⁽³⁾	ECC Control register.
C_MPMC_CTRL_BASEADDR + 0x4	ECCS ⁽¹⁾	R/ROW ⁽²⁾	00000000	ECC Status register.
C_MPMC_CTRL_BASEADDR + 0x8	ECCSEC ⁽¹⁾	R/ROW ⁽²⁾	00000000	ECC Single Bit Error Count register.
C_MPMC_CTRL_BASEADDR + 0xC	ECCDEC ⁽¹⁾	R/ROW ⁽²⁾	00000000	ECC Double Bit Error Count register.
C_MPMC_CTRL_BASEADDR + 0x10	ECCPEC ⁽¹⁾	R/ROW ⁽²⁾	00000000	ECC Parity Field Single Bit Error Count register.
C_MPMC_CTRL_BASEADDR + 0x14	ECCADDR ⁽¹⁾	RO ⁽⁵⁾	N/A	ECC Error Address register.
ECC ISC				
C_MPMC_CTRL_BASEADDR + 0x1C	DGIE ⁽¹⁾	R/W	00000000	ECC Device Global Interrupt Enable register.
C_MPMC_CTRL_BASEADDR + 0x20	IPIS ⁽¹⁾	R/TOW ⁽⁴⁾	00000000	ECC IP Interrupt Status register.
C_MPMC_CTRL_BASEADDR + 0x24	IPIE ⁽¹⁾	R/W	00000000	ECC IP Interrupt Enable register.

1. Used when C_INCLUDE_ECC_SUPPORT = 1 only
2. ROW = Reset On Write. A write operation will reset the register
3. Reset condition of ECCC depends on the value of parameter C_ECC_DEFAULT_ON
4. TOW = Toggle On Write. Writing 1 to a bit position within the register causes the corresponding bit position in the register to toggle
5. RO = Read Only
6. R/W = Read/Write

Static PHY Register Summary

Note: This register is available only when the Static PHY is enabled. Please refer to "Static PHY Interface," page 70 for more information regarding Static PHY.

Table 25: Static PHY Register Summary

Summary Grouping	MPMC_CTRL Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
Static PHY	C_MPMC_CTRL_BASEADDR + 0x1000	SPIR	R/W	Based on parameter settings	Static PHY Control Register.

MIG PHY Debug Registers

The MIG PHY debug registers allow for software-based access into MIG PHY calibration logic, which lets you read or change the PHY calibration settings. This feature is useful for board bring-up, debug of a memory interface, and margin analysis.

These MIG debug registers provide access points into the MIG PHY calibration logic. For more information about the MIG calibration registers and calibration algorithm, refer to the *MIG User Guide* and the Application Notes XAPP701, *Virtex-4 DDR1/DDR2*, XAPP768c, *Spartan-3 DDR1*, XAPP454, *Spartan-3 DDR2*, XAPP851 *Virtex-5 DDR1*, or XAPP858, *Virtex-5 DDR2*. The "Reference Documents," page 193 contains links to these documents.

To use the MPMC debug registers and to understand the function of these registers, you need to be familiar with these MIG documents. Also, refer to the software applications located at `<EDK Install>/sw/XilinxProcessorIPLib/drivers/mpmc_v3_00_a/examples/mpmc_debug*.c` for an example.

The following tables list the debug registers that are common to all devices and debug registers that are device-specific.

Common MIG PHY Debug Registers

Table 26: Common MIG PHY Debug Registers

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description
CALIB_RST_CTRL 0x000	0:30	unused			
	31	REG_DEFAULT_ON_RST	R/W	1	0 = MPMC reset does not change control registers. 1 = Upon MPMC reset, set all calibration control registers back to default values (except this register).
The following registers are only valid if <code>C_INCLUDE_ECC_SUPPORT = 1</code>					
ECC_DEBUG 0x010	0	ECC_BYTE_ACCESS_EN	R/W	0	0 = ECC byte data is controlled by the normal ECC logic. 1 = Enable debug access to the ECC byte lane to read/write the ECC byte data directly.
	1:31	unused			
ECC_READ_DATA 0x014	0:7	ECC Read Data0	R	0	Data read from ECC byte lane on the first byte of the data in the 4 beat memory burst.
	8:15	ECC Read Data1	R	0	Data read from ECC byte lane on the second byte of the data in the 4 beat memory burst.
	16:23	ECC Read Data2	R	0	Data read from ECC byte lane on the third byte of the data in the four beat memory burst.
	24:31	ECC Read Data3	R	0	Data read from ECC byte lane on the fourth byte of the data in the four beat memory burst.
ECC_WRITE_DATA 0x018	0:7	ECC Write Data0	R/W	0	First byte of ECC data in the four beat memory burst to be written out.
	8:15	ECC Write Data1	R/W	0	Second byte of ECC data in the four beat memory burst to be written out.
	16:23	ECC Write Data2	R/W	0	Third byte of ECC data in the four beat memory burst to be written out.
	24:31	ECC Write Data3	R/W	0	Fourth byte of ECC data in the four beat memory burst to be written out.

Spartan®-3 MIG PHY Debug Registers

Table 27: Spartan-3 MIG PHY Debug Registers

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description
S3_CALIB_REG 0x040	0:6	unused			
	7	VIO_OUT_DQS_EN	R/W	0	Enable signal for strobe tap selection. 0 = not enabled 1 = enabled
	8:10	unused			
	11:15	VIO_OUT_DQS	R/W	01111	Used to change the tap values for strobes.
	16:22	unused			
	23	VIO_OUT_RST_DQS_DIV_EN	R/W	0	Enable signal for RST_DQS_DIV tap selection. 0 = not enabled 1 = enabled
	24:26	unused			
	27:31	VIO_OUT_RST_DQS_DIV	R/W	01111	Used to change the tap values for RST_DQS_DIV.
S3_CALIB_STATUS 0x044	0:2	unused			
	3:7	DBG_DELAY_SEL	R		Tap value from the calibration logic used to delay the strobe and RST_DQS_DIV.
	8:10	unused			
	11:15	DBG_PHASE_CNT	R		Phase count gives the number of LUTs in the clock phase.
	16:17	unused			
	18:23	DBG_CNT	R		Counter used in the calibration logic.
	24	unused			
	25	DBG_TRANS_ONEDTCT	R		Asserted when the first transition is detected.
	26	DBG_TRANS_TWODTCT	R		Asserted when the second transition is detected.
	27	DBG_ENB_TRANS_TWO_DTCT	R		Enable signal for DBG_TRANS_TWODTCT. Related MIG PHY signal = PHY_INIT_DONE.
28:31	unused				

Virtex-4 MIG PHY Debug Registers

Table 28: Virtex-4 MPMC MIG PHY Debug Registers

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description
V4_CALIB_REG 0x100	0:5	unused			
	6	IDELAYCTRL_RDY_O	R	0	Status of MPMC_Idelayctrl_Rdy_ O. 0 = not ready 1 = ready
	7	IDELAYCTRL_RDY_I	R	0	Status of MPMC_Idelayctrl_Rdy_ I. 0 = not ready 1 = ready
	8:12	unused			
	13	FORCE_INITDONE	R/W	0	1 = Force MPMC INIT_DONE signal to be equal to FORCE_INITDONEVAL 0 = Allow hardware calibration engine to drive MPMC INIT_DONE signal
	14	FORCE_INITDONE_VAL	R/W	0	Value to set MPMC INIT_DONE when FORCE_INITDONE = 1. 0 = not done 1 = done
	15	MIG_INIT_DONE	R	0	0 = incomplete. 1 = MIG_HW_CALIBRATION initialization is complete. <i>Note:</i> HW calibration could be complete but the INIT_DONE signal from the PIM might be masked by the FORCE_INITDONE signal. Related MIG PHY signal: PHY_INIT_DONE
	16:30	unused			
31	HW_CALIB_ON_RESET	R/W	1	Calibrate HW upon reset. 0 = Do not run hardware calibration engine upon MPMC reset. 1 = Start memory hardware calibration engine upon MPMC reset.	

Table 28: Virtex-4 MPMC MIG PHY Debug Registers (Cont'd)

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description	
V4_CALIB_STATUS 0x104	0:5	unused				
	6	SEL_DONE	R	0	Indicates calibration process of center-aligning DQS with respect to clock is complete. 0 = not done 1 = done Related MIG PHY signal: SEL_DONE	
	7:15	DONE_STATUS	R	0	Tap control and pattern compare calibration completion status, 1 bit plus 1 bit per DQS bit. 0 = calibration not complete 1 = calibration complete Related MIG PHY signals: COMP_DONE COMP_ERROR	
	16:22	unused				
	23:31	ERR_STATUS	R	0	4-bit calibration error status. 0 = error 1 = no error Related MIG PHY signal: COMP_ERR	
V4_CALIB_DQS_GROUP0 . . . V4_CALIB_DQS_GROUP8	0x140 ... 0x160	0:7	DQ_IN_BYTE_ALIGN<n>	R/W	0	Calibration bit alignment of 8 bits within the byte. 0 = bits aligned or align bits 1 = no bit alignment Related MIG PHY signal: DELAY_ENABLE
		11:15	RDEN_DLY<n>	R/W	0	Number of cycles after read command until read data is valid for DQS group<n>. Related MIG PHY signal: CLK_COUNT
		16:29	unused			
		30	DELAY_RD_FALL<n>	R/W		Indicates relative alignment of bytes for DQS group<n>. Related MIG PHY signal: DELAY_RD_FALL
		31	RD_SEL<n>	R/W	0	Final read capture MUX set for positive or negative edge capture for DQS group<n>. Related MIG PHY signal: FIRST_RISING

Table 28: Virtex-4 MPMC MIG PHY Debug Registers (Cont'd)

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description
V4_CALIB_DQS_TAP_GROUP0 0x180 . . . V4_CALIB_DQS_TAP_GROUP8 0x1a0	0:6	unused			
	7	DQS_TAP_CNT_INC<n>	Wr Only	N/A	DQS<n>IDELAY tap count increment, 1 tap increment per write.
	8:14	unused			
	15	DQS_TAP_CNT_DEC<n>	Wr Only	N/A	DQS<n>IDELAY tap count decrement, 1 tap decrement per write.
	16:25	unused			
	26:31	DQS_TAP_CNT<n>	R	0	DQS<n>IDELAY tap count. 0 = Pass 1 = Fail
V4_CALIB_DQ_TAP_COUNT<n> 0x200 . . . 0x31c	0:22	unused			
	23	DQ_DELAY_EN<n>	W	0	DQ<n> alignment of bits within a byte lane
	24:31	unused			

Virtex-5 MIG PHY Debug Registers

Table 29: Virtex-5 MIG PHY Debug Registers

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description
V5_CALIB_REG 0x400	0:5	unused			
	6	IDELAYCTRL_RDY_O	R	0	Status of MPMC_Idelayctrl_Rdy_O. 0 = not ready 1 = ready
	7	IDELAYCTRL_RDY_I	R	0	Status of MPMC_Idelayctrl_Rdy_I. 0 = not ready 1 = ready
	8:12	unused			
	13	FORCE_INITDONE	R/W	0	0 = Allow hardware calibration engine to drive MPMC INIT_DONE signal. 1 = Force MPMC INIT_DONE signal to be equal to FORCE_INITDONEVAL.
	14	FORCE_INITDONE_VAL	R/W	0	Value to set MPMC INIT_DONE when FORCE_INITDONE = 1.

Table 29: Virtex-5 MIG PHY Debug Registers (Cont'd)

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description
V5_CALIB_REG 0x400	15	MIG_INIT_DONE	R	0	0 = incomplete. 1 = MIG_HW_CALIBRATION initialization is complete; <i>Note:</i> HW calibration could be complete but the INIT_DONE signal from the PIM might be masked by the FORCE_INITDONE signal.
	16:30	unused			
	31	HW_CALIB_ON_RESET	R/W	1	Calibrate HW upon reset. 1 = Start memory hardware calibration engine upon MPMC reset 0 = Do not run hardware calibration engine upon MPMC reset
V5_CALIB_STATUS 0x404	0	unused			
	1:7	BIT_ERR_INDEX	R	0	When a calibration error is reported, this field indicates which bit DQ is failing.
	8:11	unused			
	12:15	DONE_STATUS	R	0	4-bit calibration completion status: 0 = Incomplete 1 = Complete
	16:27	unused			
	28:31	ERR_STATUS	R	0	4-bit calibration error status. 0 = no error 1 = error
V5_CALIB_DQS_GROUP0 0x440 . . . V5_CALIB_DQS_GROUP8 0x460	0:10	unused			
	11:15	RDEN_DLY<n>	R/W	0	Number of cycles after read command until read data is valid for DQS group<n>.
	16:18	unused			
	19:23	GATE_DLY<n>	R/W	0	Number of cycles after read command until clock enable for DQ byte group is deasserted to prevent postamble glitch for DQS group <n>.
	24:30	unused			
	31	RD_SEL<n>	R/W	0	Final read capture MUX set for positive or negative edge capture for DQS group<n>: 0 = Pass 1 = Fail

Table 29: Virtex-5 MIG PHY Debug Registers (Cont'd)

Register Name Base Address + Offset (hex)	Bits 0:31	Field Name	Access Type	Default Value	Description	
V5_CALIB_DQS_TAP_CNT0 . . . V5_CALIB_DQS_TAP_CNT8	0x4a0 . . . 0x4c0	0:6	unused			
		7	DQS_TAP_CNT_INC<n>	Wr Only	N/A	DQS<n> IDELAY tap count increment, 1 tap increment per write.
		8:14	unused			
		15	DQS_TAP_CNT_DEC<n>	Wr Only	N/A	DQS<n> IDELAY tap count decrement, 1 tap decrement per write.
		16:25	unused			
		26:31	DQS_TAP_CNT<n>	R	0	DQS<n> IDELAY tap count. 0 = Pass 1 = Fail
V5_CALIB_GATE_TAP_CNT0 . . . V5_CALIB_GATE_TAP_CNT8	0x4c0 . . . 0x4e0	0:6	unused			
		7	GATE_TAP_CNT_INC<n>	Wr Only	N/A	GATE<n> IDELAY tap count increment, 1 tap increment per write.
		8:14	unused			
		15	GATE_TAP_CNT_DEC<n>	Wr Only	N/A	GATE<n> IDELAY tap count decrement, 1 tap decrement per write.
		16:25	unused			
		26:31	GATE_TAP_CNT<n>	R	0	GATE<n> IDELAY tap count.
V5_CALIB_DQ_TAPCNT0 . . . V5_CALIB_DQ_TAP_CNT71	0x600 . . . 0x71c	0:6	unused			
		7	DQ_TAP_CNT_INC<n>	Wr Only	N/A	DQ<n> IDELAY tap count increment, 1 tap increment per write.
		15	DQ_TAP_CNT_DEC<n>	Wr Only	N/A	DQ<n> IDELAY tap count decrement, 1 tap decrement per write.
		16:25	unused			
		26:31	DQ_TAP_CNT<n>	R	0	DQ<n> IDELAY tap count.

MPMC Status Register

The MPMC Status register is available only when one of set of registers is enabled. It consists of a single Read-only register that displays static information about MPMC that is useful for software to read to obtain hardware configuration information. The following tables provide the MPMC status register and bit definitions.

Table 30: MPMC Status Register Description

MPMC_CTRL Base Address + Offset (hex)	Register Name	Access Type	Default value (hex)	Description
C_MPMC_CTRL_BASEADDR + 0x3000	MCSR0	RO	N/A	MPMC Ctrl Status Register 0

Table 31: MCSR0 Bit Definitions

Bit(s)	Field Name	Core Access	Default Value	Description
0	ECC CTRL Interface Present	R	x	1 = MPMC has been synthesized with <code>C_INCLUDE_ECC_SUPPORT = 1</code> and the error correcting code registers are present.
				0 = MPMC has been synthesized with <code>C_INCLUDE_ECC_SUPPORT = 0</code> and the error correcting code registers are not present.
1	Static PHY CTRL Interface Present	R	x	1 = MPMC has been synthesized with <code>C_USE_STATIC_PHY = 1</code> and the Static PHY registers are present.
				0 = MPMC has been synthesized with <code>C_USE_STATIC_PHY = 0</code> and the Static PHY registers are not present.
2	Debug Registers CTRL Interface Present	R	x	1 = MPMC has been synthesized with <code>C_DEBUG_ENABLE = 1</code> and the Debug registers are present.
				0 = MPMC has been synthesized with <code>C_DEBUG_ENABLE = 0</code> and the Static PHY registers are not present.
3	MPMC_CTRL Status Interface Present	R	1	1 = MPMC has been synthesized with <code>C_INCLUDE_ECC_SUPPORT = 1</code> and/or <code>C_USE_STATIC_PHY = 1</code> and/or <code>C_DEBUG_ENABLE = 1</code> and/or <code>C_PM_ENABLE = 1</code>
				0 = MPMC has not been synthesized with the MPMC_CTRL interface (not possible)
4:6	Reserved	R	x	Reserved
7	PM CTRL Interface Present	R	x	1 = MPMC has been synthesized with <code>C_PM_ENABLE = 1</code> and the Performance Monitor registers are present.
				0 = MPMC has been synthesized with <code>C_PM_ENABLE = 0</code> and the Performance Monitor registers are not present.
8:15	Reserved	R	x	Reserved
16:19	Memory Type	R	x	External Memory Type: 0x0 = SDRAM 0x1 = DDR 0x2 = DDR2
20:23	Memory Width (not including ECC Data Bits)	R	x	External Memory Interface Width: 0x3 = 8 bits 0x4 = 16 bits 0x5 = 32 bits 0x6 = 64 bits

Table 31: MCSRO Bit Definitions (Cont'd)

Bit(s)	Field Name	Core Access	Default Value	Description
24	Reserved	R	x	Reserved
25:27	Number of Ports	R	x	Number of Ports: 0x0 = 1 port 0x1 = 2 port 0x2 = 3 ports 0x3, = 4 ports 0x4 = 5 ports 0x5 = 6 ports 0x6 = 7 Ports 0x7 = 8 ports
28:31	Device Family	R	x	Device Family: 0x0 = Spartan®-3 0x1 = Virtex®-4 0x2 = Virtex-5 0xF = Unknown Family

Performance Monitor Register Summary

Note: These registers are available only when a Performance Monitor (PM) is enabled. Refer to "Performance Monitor Registers," page 95 for detailed Performance Monitor register information.

Table 32: Performance Monitor Register Summary

MPMC_CTRL Base Address + Offset (hex)	Register Name	Access Type	Default value (hex)	Description
PM CONTROL				
C_MPMC_CTRL_BASEADDR + 0x7000	PMCTRL	R/W	00000000	PM Control Register.
C_MPMC_CTRL_BASEADDR + 0x7004	PMCLR	W	00000000	PM Clear Register.
C_MPMC_CTRL_BASEADDR + 0x7008	PMSTATUS	R/TOW	00000000	PM Status Register.
PM DATA				
C_MPMC_CTRL_BASEADDR + 0x7010	PMGCC	R	0000000000000000 ⁽¹⁾	PM Global Cycle Counter.
C_MPMC_CTRL_BASEADDR + 0x7020	PM0_DCC	R	0000000000000000 ⁽²⁾	PM Dead Cycle Counter Port 0.
C_MPMC_CTRL_BASEADDR + 0x7028	PM1_DCC	R	0000000000000000 ⁽²⁾	PM Dead Cycle Counter Port 1.
C_MPMC_CTRL_BASEADDR + 0x7030 – C_MPMC_CTRL_BASEADDR + 0x7050	PM2_DCC – PM6_DCC	R	0000000000000000 ⁽²⁾	PM Dead Cycle Counter Port 2-6.
C_MPMC_CTRL_BASEADDR + 0x7058	PM7_DCC	R	0000000000000000 ⁽²⁾	PM Dead Cycle Counter Port 7.
C_MPMC_CTRL_BASEADDR + 0x8000	PM0_DATABIN0	R	0000000000000000 ⁽³⁾	PM Port 0, Data Bin 0.
C_MPMC_CTRL_BASEADDR + 0x8008	PM0_DATABIN1	R	0000000000000000 ⁽³⁾	PM Port 0, Data Bin 1,
C_MPMC_CTRL_BASEADDR + 0x8010 - C_MPMC_CTRL_BASEADDR + 0x8FF8	PM0_DATABIN2 – PM0_DATABIN511	R	0000000000000000 ⁽³⁾	PM Port 0, Data Bin 2 – PM Port 0, Data Bin 511,
C_MPMC_CTRL_BASEADDR + 0x9000 - C_MPMC_CTRL_BASEADDR + 0xFFFF8	PM1_DATABIN0 – PM7_DATABIN511	R	0000000000000000 ⁽³⁾	PM Port 1, Data Bin 0. PM Port 7, Data Bin 511.

1. The size of this register is 64 bits and is determined by the C_PM_GC_WIDTH parameter. If this parameter is less than 64 bits, the MSB is padded with 0s.
2. The sizes of these registers are 64 bits and are determined by the C_PM_DC_WIDTH parameter. If this parameter is less than 64 bits, the MSB is padded with 0s.
3. The size of this register is 64 bits, the data bins only hold 36 bins, therefore the upper MSBs are padded with 0s.

SDMA Register Summary

The Service Base Address varies based on the setting of parameter `C_ALL_PIMS_USE_SHARED_ADDRESSES`:

- If set to 0, the Service Base Address is located at `C_SDMA_CTRL<Port_Num>_BASEADDR`.
- If set to 1, each port will share the same base address of `C_SDMA_CTRL_BASEADDR` (with no `<Port_Num>` specified), and the Service Base Address for each Port is defined as follows:
 - Port 0: `C_SDMA_CTRL_BASEADDR + 0x0`
 - Port 1: `C_SDMA_CTRL_BASEADDR + 0x80`
 - Port 2: `C_SDMA_CTRL_BASEADDR + 0x100`
 - Port 3: `C_SDMA_CTRL_BASEADDR + 0x180`
 - Port 4: `C_SDMA_CTRL_BASEADDR + 0x200`
 - Port 5: `C_SDMA_CTRL_BASEADDR + 0x280`
 - Port 6: `C_SDMA_CTRL_BASEADDR + 0x300`
 - Port 7: `C_SDMA_CTRL_BASEADDR + 0x380`

The following table shows the SDMA registers and the PLB address offset from the Service Base Address assignment with the allowed access to that register. These registers are available only when an SDMA interface is enabled. Refer to "[SDMA Registers](#)," page 130 for SDMA register information.

Table 33: SDMA Registers and PLB Address Offsets from Service Base Address

PLB Address Offset from Service Base Address Assignment	Register Name	Access Type	Default Value (hex)	Description
Transmit Registers				
0x00	TX_NXTDESC_PTR	R	00000000	TX Next Descriptor Pointer.
0x04	TX_CURBUF_ADDR	R	00000000	TX Current Buffer Address.
0x08	TX_CURBUF_LENGTH	R	00000000	TX Current Buffer Length.
0x0C	TX_CURDESC_PTR	R/W	00000000	TX Current Descriptor Pointer.
0x10	TX_TAILDESC_PTR	R/W	00000000	TX Tail Descriptor Pointer.
0x14	TX_CHNL_CTRL	R/W	00000000	TX Channel Control.
0x18	TX_IRQ_REG	R/W	00FF0000	TX Interrupt Register.
0x1C	TX_CHNL_STS	R	00000000	TX Status Register.
Receive Registers				
0x20	RX_NXTDESC_PTR	R	00000000	RX Next Descriptor Pointer.
0x24	RX_CURBUF_ADDR	R	00000000	RX Current Buffer Address.
0x28	RX_CURBUF_LENGTH	R	00000000	RX Current Buffer Length.
0x2C	RX_CURDESC_PTR	R/W	00000000	RX Current Descriptor Pointer.
0x30	RX_TAILDESC_PTR	R/W	00000000	RX Tail Descriptor Pointer.
0x34	RX_CHNL_CTRL	R/W	00000000	RX Channel Control.
0x38	RX_IRQ_REG	R/W	00FF0000	RX Interrupt Register.
0x3C	RX_CHNL_STS	R	00000000	RX Status Register.

Table 33: SDMA Registers and PLB Address Offsets from Service Base Address (Cont'd)

PLB Address Offset from Service Base Address Assignment	Register Name	Access Type	Default Value (hex)	Description
Control Registers				
0x40	DMA_CONTROL_REG	R/W	000001C	DMA Control Register.

MPMC Use Cases

You can use MPMC to build different system use cases. Because of software variables, device bus transactions, memory latency, and speed; each system is capable of different size and data bandwidths. MPMC allows you to build systems quickly for different use cases, and then compare the size and the performance. There are many more use cases than can be shown; the use cases in this document can be used as an aide to understanding what types of trade-offs can be made when configuring MPMC. Each application is different, and no use case is ideal for all applications.

The following subsections provide examples of these available system use cases:

- [Standard PowerPC 405 CoreConnect Use Case](#)
- [Single MicroBlaze Processor Use Case](#)
- [Dual PowerPC Processor 405 Use Case](#)
- [Using MPMC in Standalone Systems](#)

Standard PowerPC 405 CoreConnect Use Case

An MPMC module fits easily into existing PowerPC 405 CoreConnect-based systems as a single-port memory controller as shown in the following figure. This is particularly useful as a starting point to port an existing design into an MPMC use case that allows for improved performance.

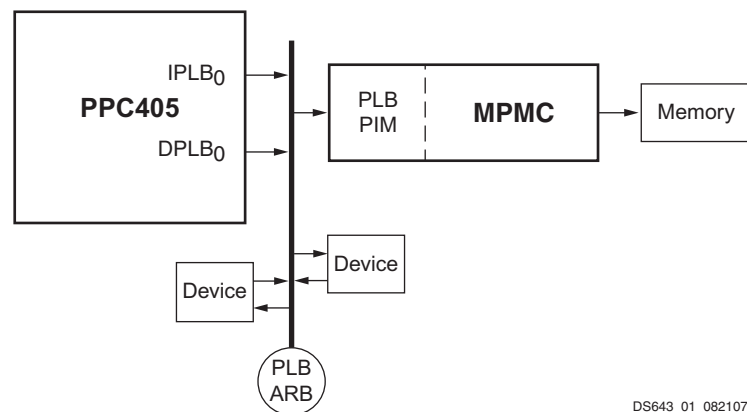


Figure 1: Standard PowerPC CoreConnect Use Case

Single MicroBlaze Processor Use Case

The following figure shows an example of a common MicroBlaze processor system layout. The MPMC module provides direct memory access to the processor IXCL and DXCL interfaces.

A standard PLB port is defined for use with PLB devices. The MicroBlaze processor can be connected directly to the PLB bus attached to the PLB PIM also.

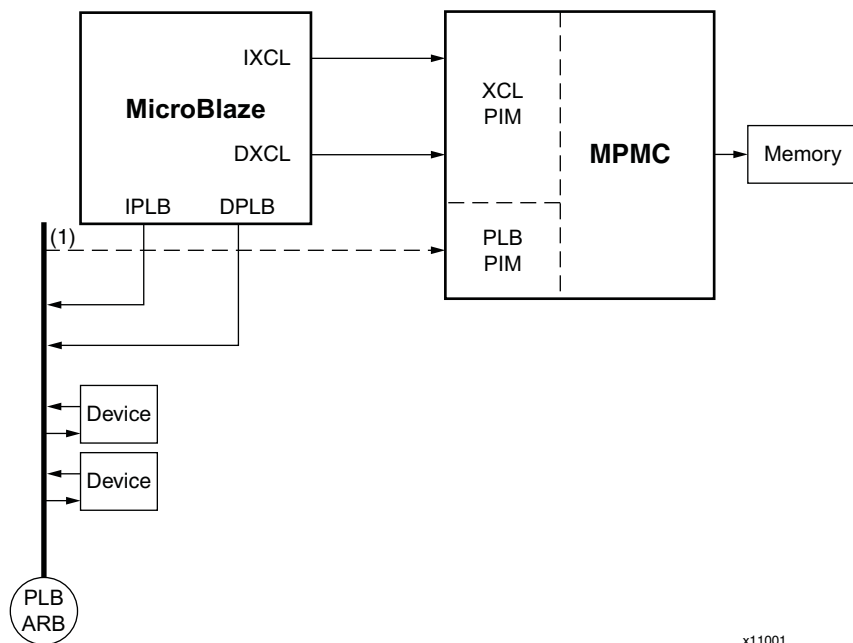


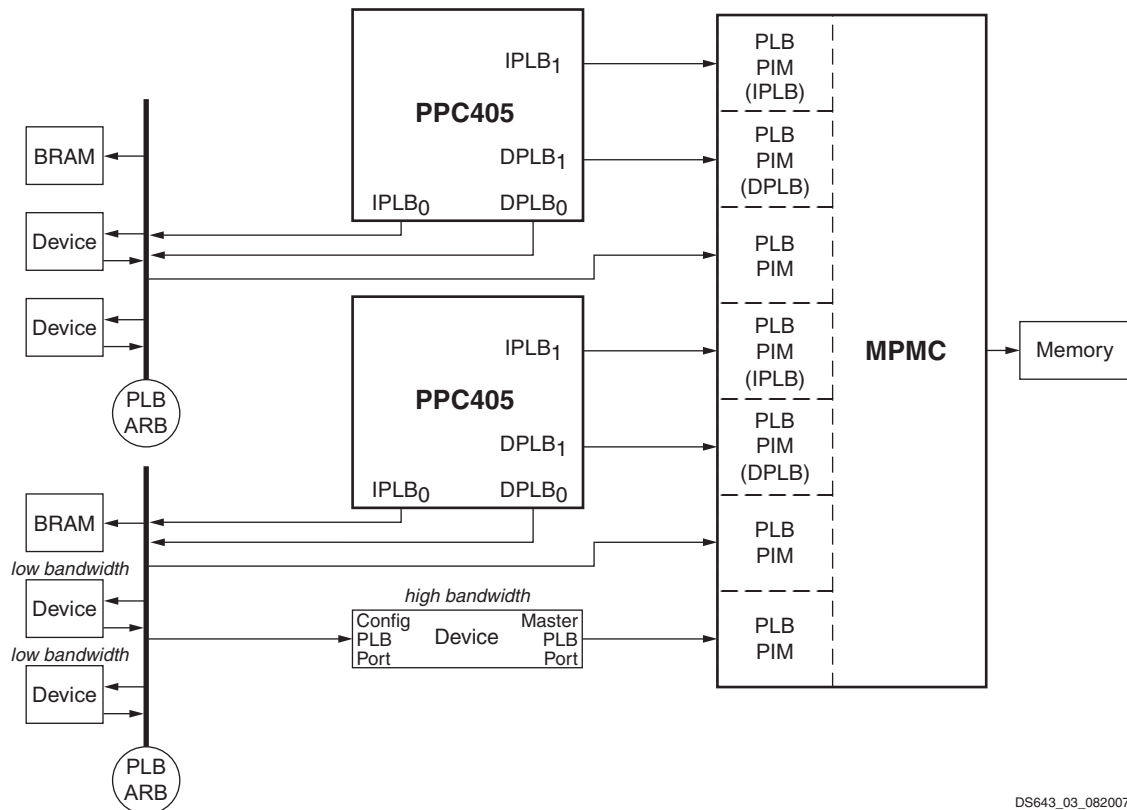
Figure 2: Single MicroBlaze Processor Use Case

Note: (1) This figure shows MicroBlaze processor version 7.20 or greater using a PLB connection to MPMC for uncached access to memory. This approach uses a second MPMC port for the uncached access over PLB. An alternate connection scheme is to remove the second MPMC port and use MicroBlaze processor parameters `C_ICACHE_ALWAYS_USED = 1` and `C_DCACHE_ALWAYS_USED = 1`. These parameters cause a MicroBlaze processor to make uncached memory accesses over XCL instead of PLB to reduce system size.

1. This diagram also shows a dual XCL PIM connection where a pair of MicroBlaze XCL interfaces connect to a single XCL PIM that consumes 1 MPMC port. The dual XCL feature is enabled using the `C_XCL<Port_Num>_B_IN_USE` parameter.

Dual PowerPC Processor 405 Use Case

This figure shows an example of two PowerPC 405 processors connected directly to an MPMC module.



DS643_03_082007

Figure 3: Dual Processor PowerPC Use Case

On the first PowerPC 405 processor:

- The IPLB1 and DPLB1 ports of the first PowerPC 405 processor are connected to the first two MPMC PLB PIMs. These are point to point connections for improved performance. Additionally, the PLB PIMs are designated as IPLB and DPLB PIMs to allow for performance optimizations inside of the PIM.
- The IPLB0 and DPLB0 ports of the first PowerPC 405 are connected to the PLB Bus attached to the third PLB PIM. Also attached to the PLB Bus are block RAM memory, which can be used to boot code and other PLB devices necessary to a particular application.

On the second PowerPC 405 processor:

- The IPLB1 and DPLB1 ports of the second PowerPC 405 processor are connected directly to the fourth and fifth PLB PIMs.
- The IPLB0 and DPLB0 ports of the second PowerPC 405 are connected to the PLB bus attached to the sixth PLB PIM. Also on this PLB bus are the block RAM memory, which can be used to boot code and other “low-bandwidth” PLB devices.
- On the seventh PLB PIM is a “high bandwidth” PLB device. This device has a direct connection to MPMC to improve performance. The configuration PLB port of this high bandwidth device is connected to the sixth PLB PIM to allow configuration from the second PowerPC 405 processor.

Using MPMC in Standalone Systems

You can use the MPMC core in a processor system that is not derived from the Embedded Development Toolkit (EDK) Xilinx Platform Studio (XPS). The general flow is to run EDK to configure and generate the MPMC core, then export MPMC out of EDK into Project Navigator to be used as a standalone core afterwards.

Note: EDK is required to configure or generate MPMC. MPMC cannot be configured or generated without EDK. Do not attempt to manually modify the parameters in a standalone MPMC core after it has been generated because many parameters have dependencies that are managed by the EDK tools.

Using EDK XPS to Manage the MPMC Core

To use the MPMC in a non-EDK processor system use EDK XPS to manage the MPMC core.

1. Create a design using the Base System Builder (BSB) wizard, choosing a development board that uses the same memory technology to be used in the actual design, with all other cores deselected except for the MPMC.
2. Double-click on the MPMC instance in the System Assembly view to configure MPMC using the MPMC GUI. Parameterize the MPMC as needed, including choosing the desired PIMs.
3. Remove all unwanted peripherals and processors. Delete any external I/O associated with these peripherals.
4. In the XPS Port tab, connect all relevant PIM signals as external I/O.

Choose **Filters >Default Connections** port filter to show all MPMC ports that are usually part of a bus connector.

Each PIM I/O signal has a port number and PIM type in the GUI that corresponds to the port order and PIM type shown in the MPMC GUI. For example, if PIM2 is configured as VFBC, the port signals would begin with "VFBC2_". NPI ports use the generic prefix "PIM", such as "PIM2_". You can save time by selecting the relevant ports then right-clicking the **Make External** option.

5. Run **Hardware >Generate Netlist** to check for any incorrect, missing, or extraneous connections; correct as needed.
6. Add the XPS file as a source in the ISE Project Navigator project and build. For an instantiation template, select the XPS project in the Sources tab and then run **View HDL Instantiation Template** from the Processes tab.

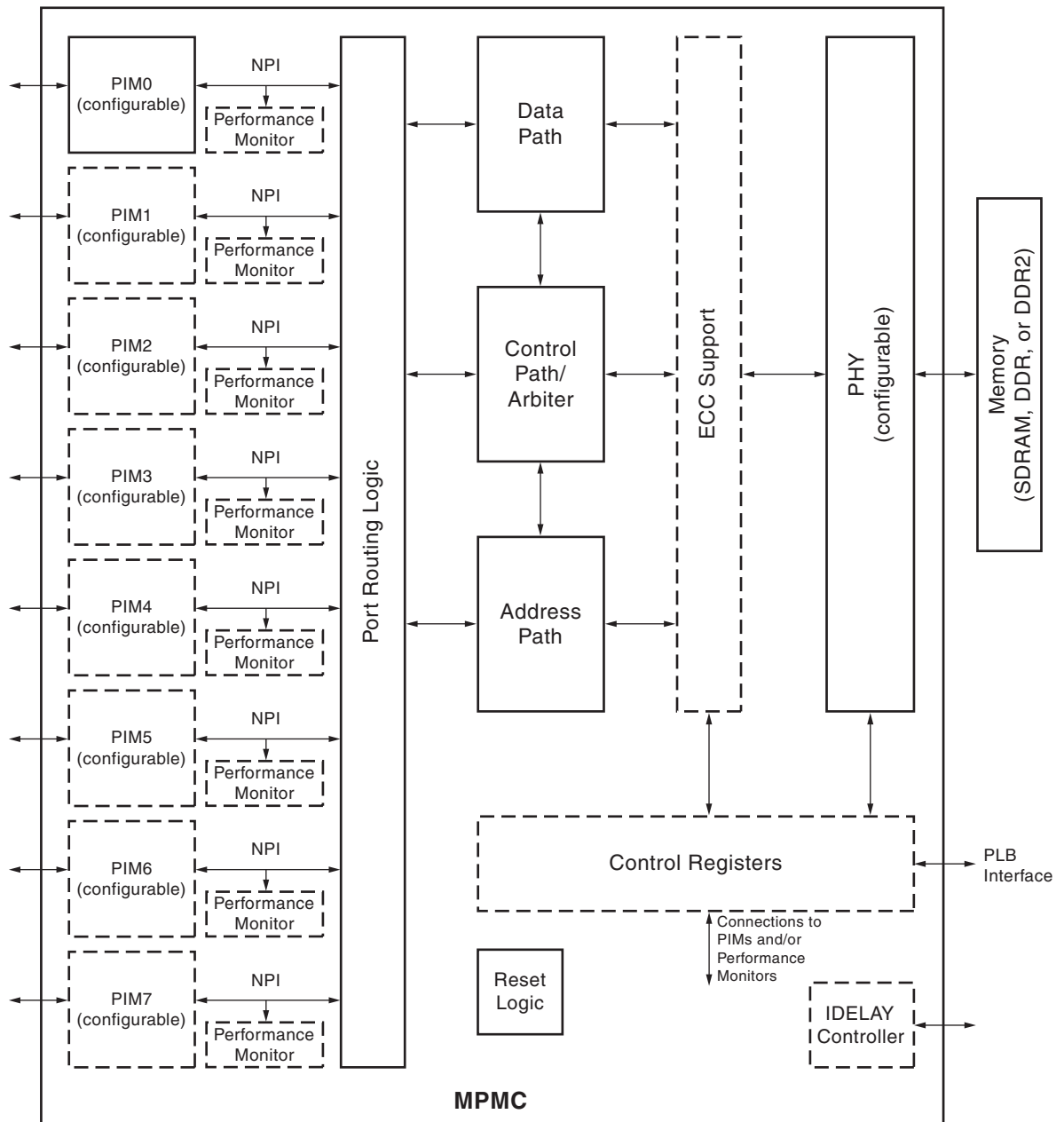
Memory Controller Architecture

The MPMC architecture comprises the following components:

- [Address Path](#)
- [Data Path](#)
- [Control Path / Arbiter](#)
- [Configurable Physical Interface](#)
- [Clock and Reset Logic](#)
- [Transaction Ordering and Memory Coherency](#)
- [Error Correction Code](#) (optional)
- [Performance Monitoring](#) (optional)
- Descriptions of the MPMC "[Personality Interface Modules](#)," [page 102](#) detail how the following PIMS allow MPMC to connect to various interfaces:
 - [Xilinx CacheLink PIM](#)
 - [Soft Direct Memory Access Controller for LocalLink Interfaces](#)
 - [Processor Local Bus Version 4.6 PIM](#)
 - [PowerPC 440 Memory Controller PIM](#)
 - [Video Frame Buffer Controller PIM](#)
 - [Native Port Interface PIM](#)

Throughout this document, the term “word” signifies a 32-bit word.

The following figure is a block diagram of the MPMC architecture. The following subsections describe the MPMC architectural features



DS643_04_082207

Figure 4: MPMC Architecture Block Diagram

Address Path

The address path allows each PIM to have independent access. Each PIM supports 32-bit addresses. This allows at least one 32-bit address from each PIM to be acknowledged simultaneously. Then the control path determines the order that the addresses go to memory. Refer to "[Transaction Ordering and Memory Coherency](#)," page 84 for more details.

The address path information includes the following topics:

- [Base/High/Offset Parameters](#)
- [Address Encoding](#)
- [Address Path Pipeline](#)
- [Address Alignment by Read and Write Request](#)

Base/High/Offset Parameters

Each PIM has PIM-specific Base/High/Offset address parameters. Refer to the documentation of the PIM you are using for correct address information. The MPMC parameters are listed in "[Design Parameters](#)," page 1.

The Base/High/Offset parameters are defined as a 32-bit value of `C_<PIM_Type>_[BASEADDR|HIGHADDR|OFFSET]`. The value set in the `C_ALL_PIMS_SHARE_ADDRESSES` parameter determines if all ports have a common base and high address or if each port has independently-configured memory address ranges. If you want to implement a shadow or aliased memory you need to double the amount of addressable memory. This can be done by increasing the `C_<PIM_Type>_HIGHADDR` by an amount that will double the address range. MPMC supports a maximum of 2 gigabytes total memory.

Address Encoding

MPMC does not perform address range validation, and MPMC will respond to all addresses, so it is the responsibility of the PIM to ensure that valid addresses are being passed across the NPI interface. The address path is responsible for translating the address into an encoded address that adheres to SDRAM, DDR, or DDR2 memory specifications.

The following memory parameters are used to encode the address:

- `C_MEM_DATA_WIDTH`
- `C_MEM_PART_NUM_COL_BITS`
- `C_MEM_PART_NUM_ROW_BITS`
- `C_MEM_PART_NUM_BANK_BITS`
- `C_MEM_NUM_RANKS`
- `C_MEM_NUM_DIMMS`

The memory address offsets are as follows:

- $\text{col_addr_startbit} = \log_2(\text{C_MEM_DATA_WIDTH}/8)$
- $\text{row_addr_startbit} = \text{col_addr_startbit} + \text{C_MEM_PART_NUM_COL_BITS}$
- $\text{bank_addr_startbit} = \text{row_addr_startbit} + \text{C_MEM_PART_NUM_ROW_BITS}$
- $\text{rank_addr_startbit} = \text{bank_addr_startbit} + \text{C_MEM_PART_NUM_BANK_BITS}$
- $\text{dimm_addr_startbit} = \text{rank_addr_startbit} + \text{C_MEM_NUM_RANKS}$
- $\text{total_addr_startbit} = \text{dimm_addr_startbit} + \text{C_MEM_NUM_DIMMS}$

The following table shows the address column, row, bank, rank, DIMM, and total memory address.

Table 34: Address Type and Corresponding Signal

Memory Address Type	Corresponds to:
Column	PIM<Port_Num>_Addr[row_addr_startbit: col_addr_startbit]
Row	PIM<Port_Num>_Addr[bank_addr_startbit-1:row_addr_startbit]
Bank	PIM<Port_Num>_Addr[rank_addr_startbit-1:bank_addr_startbit]
Rank	PIM<Port_Num>_Addr[dimm_addr_startbit-1:rank_addr_startbit]
DIMM	PIM<Port_Num>_Addr[total_addr_startbit-1:dimm_addr_startbit]
Total memory space	PIM<Port_Num>_Addr[total_addr_startbit-1:0]

You can set the base and high addresses of your custom PIM to cover this address space. Standard PIMs (such as PLB and XCL PIMs) have address range and offset parameters to handle how addresses are mapped from the bus interface to the physical memory.

Address Path Pipeline

The address path pipeline is controlled by C_PIM<Port_Num>_ADDRACK_PIPELINE, a per-port parameter which, when set in the IP Configuration interface, allows the NPI address acknowledge signal to be registered. Enable this parameter to achieve better timing, but expect one extra cycle of latency on the assertion of address acknowledge. Refer to "[IP Configuration Graphical User Interface](#)," page 189 for more information on setting per-port parameters.

Address Alignment by Read and Write Request

When requesting a transfer, the signal, PIM<Port_Num>_Addr, has alignment restrictions on Reads and Writes. The following subsections list the alignment restrictions by Read and Write request.

Read Requests

Addresses corresponding to a Read request must be aligned as follows:

- Word transfers (32-bit NPI only) are 4-byte aligned
- Double-word transfers (64-bit NPI only) are 8-byte aligned
- 4-word, cacheline transfers (32-bit NPI only) are 4-byte aligned
- 4-word, cacheline transfers (64-bit NPI only) are 8-byte aligned
- 8-word, cacheline transfers (32-bit NPI only) are 4-byte aligned
- 8-word, cacheline transfers (64-bit NPI only) are 8-byte aligned
- 16-word, burst transfers are 64-byte aligned
- 32-word, burst transfers are 128-byte aligned
- 64-word, burst transfers are 256-byte aligned (Not supported in all configurations. See "[Restrictions on 64-Word Burst Transfers](#)," page 156 for more information.)

Write Requests

Addresses corresponding to a Write request must be aligned to the size of the requested transfer as:

- Word transfers (32-bit NPI only) are 4-byte aligned
- Double-word transfers (64-bit NPI only) are 8-byte aligned
- 4-word cache-line transfers are 16-byte aligned
- 8-word cache-line transfers are 32-byte aligned
- 16-word burst transfers are 64-byte aligned
- 32-word burst transfers are 128-byte aligned
- 64-word burst transfers are 256-byte aligned (Not supported in all configurations. See ["Restrictions on 64-Word Burst Transfers," page 156](#) for more information.)

Data Path

The MPMC data path comprises the following:

- [Supported Data Widths](#)
- [FIFO Types](#)
- [Read Word Address](#)
- [Data Path Pipelines](#)

Supported Data Widths

MPMC supports NPI data widths of 32 and 64 bits. In the discussion of data widths, “32-bit NPI” refers to an NPI data width of 32-bits, “64-bit NPI” refers to an NPI data width of 64-bits, and “NPI” refers to either 32- or 64-bit data widths.

The data path also supports SDRAM, DDR, and DDR2 memories that have total physical data widths of 8, 16, 32, and 64 bits.

FIFO Types

In MPMC, you can select either block RAM (BRAM) or 16-bit Shift Register Lookup (SRL) table FIFOs. Generally, a block RAM FIFO gives the best performance because the timing is better and the FIFO depth is larger and does not create stalls in the data path due to a full FIFO condition. The write block RAM FIFO does not assert the `PIM<Port_Num>_WrFIFO_AlmostFull` which can also simplify the design of PIMs because the PIM does not need to dynamically monitor the almost full flag. The PIM is therefore required to ensure that a block RAM FIFO will never reach a write FIFO full state. The block RAM FIFO also provides better timing than an SRL FIFO which may allow for a higher FMAX. This is especially true for Spartan-3 where the timing on the SRL primitive is significantly worse than the timing on block RAM primitive timing. In some cases, however, an SRL FIFO can improve system timing versus a block RAM FIFO when the MPMC has a large number of ports and routing to the all the block RAMs results in excessive net delays.

Some configurations of MPMC might require more block RAMs than are available on a particular device, in which case the SRL FIFOs can be used. Additionally particular applications or system configurations can use a significant number of block RAMs, leaving few resources for MPMC.

The number of block RAM or SRL resources consumed by the FIFOs depends on:

- Number of ports and whether a particular port has read FIFOs, write FIFOs, or both
- NPI data width and memory data width

See the "[Resource \(block RAM\) Utilization](#)," page 183 for more information about block RAM utilization in MPMC. This section also explains how many LUTs are used when a SRL FIFO is selected.

Read Word Address

When Read data for a cacheline request is returned, it might not be returned target-word first. Read data is returned sequentially, but it could be returned where the requested target word appears later in the sequence than desired due to memory access optimizations and the allowance of back-to-back read requests. You must monitor the `PIM<Port_Num>_RdFIFO_RdWdAddr` output value to determine which word is being returned first.

Data Path Pipelines

The following table outlines the pipeline stages in the MPMC data path; which is set using the MPMC interface. See the "[IP Configuration Graphical User Interface](#)," page 189 for more information about setting the pipeline. Adding pipeline stages improves timing but also increases latency.

Table 35: MPMC Data Path Pipelines

Pipeline Stage	Description
Write Data Path input	Registers the write FIFO inputs (push, FIFO address, data, byte enables).
Write Data Path output	Registers the write FIFO outputs (data, byte enables). All ports must have the same setting.
Write Data Path timing management	There is a multiplexer (MUX) that selects which write FIFO the PHY is currently using. This pipeline stage adds a register after that MUX
Read Data Path input	Registers the read FIFO inputs (push, FIFO address, data). All ports must have the same setting.
Read Data Path output	Registers the read FIFO outputs (data, read word address).
Read Data Path fanout	<p>The read data going from the PHY to the data path is routed to the read FIFO for each port. If the FIFOs are spaced far apart (which is likely when using block RAM FIFOs), the routing delays can be large. Setting the fanout has the following effect:</p> <ul style="list-style-type: none"> 0 = No register is instantiated. 1 = Read data is forwarded from the PHY for up to eight sets of registers. The outputs of the registers are then forwarded on to a maximum of one read FIFO. 2 = Read data is forwarded from the PHY to up to four sets of registers. The outputs of the registers are the forwarded on to a maximum of two read FIFOs.¹ 4 = Read data is forwarded from the PHY to two sets of registers. The outputs of the registers are the forwarded on to a maximum of four read FIFOs.¹ 8 = Read data is forwarded from the PHY to a single register, then forwarded on to each of the read FIFOs. <p>Note: 1. Values of 3, 5, 6, 7 are invalid.</p>

Control Path / Arbiter

The MPMC control path is configured for optimal memory bandwidth given a particular memory. It can be configured to support different physical (PHY) interfaces also. The following subsections describe the Control Path/Arbiter:

- [Transfer Types](#)
- [Arbitration Algorithms](#)
- [Arbiter Pipeline](#)
- [Control Path and Arbiter block RAM Utilization](#)

Transfer Types

The control path supports the following transfer types:

- Word reads and writes (32-bit NPI only).
- Double-word reads and writes (64-bit NPI only).
- 4-word, cacheline reads and writes.
- 8-word, cacheline reads and writes.
- 16-word, burst reads and writes.
- 32-word, burst reads and writes.
- 64-word, burst reads and writes. (Not supported in all configurations. See "[Restrictions on 64-Word Burst Transfers](#)," page 156 for more information.)

Arbitration Algorithms

The following subsections describe the supported arbitration algorithms in the MPMC Arbiter.

Fixed

In the Fixed arbitration algorithm, the priority is fixed so that Port 0 has highest priority, Port 1 has next highest priority, etc. The last port in the design has the lowest priority. The ordering from Port 0 to Port 7 cannot be changed. This algorithm uses the lowest amount of FPGA resources to implement.

Round Robin (Default)

In the Round Robin arbitration algorithm (default), each MPMC port is given an equal overall priority. This is accomplished by rotating through the relative priority of each port each time an arbitration is performed. A given port will therefore have one turn as highest priority, one turn at second highest priority, one turn at lowest priority, etc.

Custom

In Custom arbitration, the user has the flexibility to configure the arbitration. The user can choose the number of arbitration time slots (`C_ARB0_NUM_SLOTS`) to a number between 1 and 16 slots. The arbiter will rotate through these arbitration time slots each time an arbitration is performed to initiate a memory transaction. For each arbitration time slot, the user can specify the relative priority of all the ports (`C_ARB0_SLOT<Port_Number>`).

Arbitration Examples

An example of Custom arbitration shows that in a 4 port system, the user can set the following:

```
C_ARB0_NUM_SLOTS = 6
C_ARB0_SLOT0 = "0123"
C_ARB0_SLOT1 = "0123"
C_ARB0_SLOT2 = "1230"
C_ARB0_SLOT3 = "1230"
C_ARB0_SLOT4 = "2301"
C_ARB0_SLOT5 = "3012"
```

Assuming all ports are making requests, Port 0 would be selected 1/3 of time, Port 1 would be selected 1/3 of time, Port 2 would be selected 1/6 of time, and Port 3 would be selected 1/6 of time.

The functionally equivalent setting for Round Robin arbitration is as follows:

```
C_ARB0_NUM_SLOTS = 4
C_ARB0_SLOT0 = "0123"
C_ARB0_SLOT1 = "1230"
C_ARB0_SLOT2 = "2301"
C_ARB0_SLOT3 = "3012"
```

The functionally equivalent setting for Fixed arbitration is as follows:

```
C_ARB0_NUM_SLOTS = 1
C_ARB0_SLOT0 = "0123"
```

For more information on how to configure the arbiter, see the "[Arbitration](#)" information in the "[IP Configuration Graphical User Interface](#)," page 189.

Arbiter Pipeline

An optional pipeline is allowed in the arbitration logic. To achieve best timing, enable this pipeline using the IP Configuration interface. Enabling the arbiter pipeline could also increase latency. See the "[Arbitration](#)" information in the "[IP Configuration Graphical User Interface](#)," page 189 for more information.

Control Path and Arbiter block RAM Utilization

The MPMC control logic is designed around a block RAM-based state machine; therefore, the control logic always consumes one block RAM.

The arbiter uses one block RAM when a custom arbitration algorithm is used. With Fixed or Round Robin arbitration, or when `C_NUM_PORT` is set to 1, no additional block RAM is needed.

The MPMC control logic does not support row or bank management. After each NPI transaction, the row and bank that was accessed is closed with a precharge.

Configurable Physical Interface

The MPMC Physical Interface (PHY) is the interface situated between memory and the MPMC address path, control path, and data path. The PHY interface can be configured to support SDRAM, DDR SDRAM, and DDR2 SDRAM memories across Virtex-4, Virtex-5, and Spartan-(3/3A/3E/3AN/3ADSP) platforms. Mobile DDR memories are not currently supported by any PHY interface.

The following subsections describe these topics:

- **Memory Interface Generator (MIG) Based PHY Interface:** (MIG)-based PHY interface is a DDR/DDR2 physical memory interface core/reference design technology used by MPMC. See "Reference Documents," page 193 for links to more information about MIG. This is the recommended PHY interface. Using MIG-based PHY requires that you follow the MIG FPGA pinout and board layout guidelines. You must use the MIG tool to generate pinout, placement, and timing constraints (UCF file constraints) for MPMC designs. Mobile DDR and Low Power DDR are not supported by MPMC.
- **Static PHY Interface:** You can use the Static PHY when MIG pinout and layout guidelines were not followed. The Static PHY is currently available for DDR and DDR2 memories only. MPMC does not support Mobile DDR and Low Power DDR.
- **SDRAM PHY Interface:** SDRAM PHY is the interface between the Single Data Rate Access Memory (SDRAM) and the MPMC control path, address path, and data path. The SDRAM PHY interface supports Virtex-4, Virtex-5, and Spartan-(3/3E/3A/3AN/3ADSP) platforms. MPMC can read and write to Mobile SDRAM devices, but MPMC does not make use of the low power features of Mobile SDRAM, such as deep power down or partial-array refresh.
- **Connecting Memory to the PHY Interface**

The following table summarizes the available PHY interfaces available for each FPGA device family and memory combination.

Table 36: PHY Layer by Xilinx FPGA Family and Memory Type

FPGA Family	Memory Type		
	DDR2	DDR	SDRAM
Virtex-5	MIG-Based Virtex-5 DDR2 PHY ⁽¹⁾ Static PHY	MIG-Based Virtex-5 DDR PHY ⁽¹⁾ Static PHY	SDRAM PHY ⁽¹⁾
Virtex-4	MIG-Based Virtex-4 DDR2 PHY ⁽¹⁾ Static PHY	MIG-Based Virtex-4 DDR PHY ⁽¹⁾ Static PHY	SDRAM PHY ⁽¹⁾
Spartan-3 Spartan-3E Spartan-3A Spartan-3AN Spartan-3ADSP	MIG-Based Spartan-3 DDR2 PHY ⁽¹⁾ Static PHY	MIG-Based Spartan-3 DDR PHY ⁽¹⁾ Static PHY	SDRAM PHY ⁽¹⁾

1. Note: Default selection.

Memory Interface Generator (MIG) Based PHY Interface

The MIG-based PHY interface contains:

- Input and output flip-flops
- Read data delay logic
- Data capture logic
- Memory initialization logic

The read data delay logic uses IDELAYs for Virtex-4 and Virtex-5 families, and Look-Up Table (LUT) delays for Spartan-3 platforms. The delay logic is used to align the middle of the valid read data to the `MPMC_Clk0` clock edge. This is necessary to accommodate for variations in trace delays on different boards. The delay elements change the time that the read data arrives at the FPGA to align it to the main clock.

In the Virtex-4 and Virtex-5 families, this alignment is done as part of the memory initialization logic. At the end of the initialization/configuration sequence, the PHY issues dummy write and read commands. The delay logic then determines the edges of the input data and shifts the input data to allow `MPMC_Clk0` to properly capture the data. To ensure a robust interface, you must match the trace lengths for the data signals to the corresponding data strobe signal and ensure that the proper FPGA I/O pin selections are made as described in the *MIG User Guide*. "[Reference Documents](#)," [page 193](#) has a link to the MIG web page that contains the user guide.

In DDR and DDR2 cases, the data capture logic then takes the DDR read data and turns it into Single Data Rate (SDR) data. In SDRAM cases, the data is already available as SDR data. The PHY then pushes the data into the data path FIFOs, making it available to the NPI.

For more information on the use of IDELAY in DDR and DDR2 applications see the *Memory Interfaces Data Capture Using Direct Clocking Technique* document. The "[Reference Documents](#)," [page 193](#) contains a link to this document.

For more information about the MIG physical interface design, register and download the MIG design. The "[Reference Documents](#)," [page 193](#) contains a link to the Xilinx Memory webpage where the MIG design is located.

The following subsections describe the MIG-based PHY interface:

- [MIG-Based PHY Design Considerations](#)
- [Important Notes on MIG Board Compatibility](#)
- [MIG/MPMC Tool Flow](#)
- [MIG-based PHY Design and Implementation Considerations by Device](#)
- [Additional Information](#)

MIG-Based PHY Design Considerations

The following are design considerations for using the MIG-based PHY interface:

- This version of MPMC is designed to be used with MIG v3.0. Other MIG versions (either older or newer) might not produce a User Constraint File (UCF) compatible with this version of MPMC. For migrating the UCF file from older MPMCv3 and MPMCv4 designs (based on MIGv1.73-MIGv2.3) to MPMC5, the following options are available:
 - The MIG v3.0 GUI will provide an **Update Design** option that will generate MIG v3.0 constraints using an older project files from MIG v.1.73, v2.0, 2.1, 2.2, or 2.3. If your design originally used an older MIG version, it is strongly recommended that you use this option to update the design to make sure the correct constraints are being used.
 - Scripts and instructions are provided to migrate MPMC designs manually:
 - For migration of MPMCv3 designs based on MIG v1.73 to MPMCv5. See "[Migrating an MPMCv3 Design to MPMCv5](#)," page 59 for more information.
 - For migration of MPMCv4 Virtex-5 DDR2 designs to MPMCv5, see "[Migrating an MPMCv4 Virtex-5 DDR2 Design to MPMCv5](#)," page 60 for more information.

However, it is strongly recommended that the **Update Design** option in the MIG GUI be used instead of these manual or scripted method. These manual or scripted methods are intended to be used to help debug the migration of older MPMC designs only.

- When running the MIG GUI in standalone mode, ensure that the MIG memory settings match the parts you intend to use on your custom board and also match your MPMC memory settings. In particular, ensure that memory parameters that affect the memory interface placement like data width, number of rows, columns, bank bits, and memory types are correct and consistent.
- Virtex-5 DDR2 boards must be designed to support differential DQS.
- It is required and extremely important to ensure that the layout and pinout of any other boards to be used with MPMC follow the MIG design requirements. Failure to follow MIG design rules and all applicable MIG UCF constraints could result in an inoperable MPMC. Review all information in this section before implementing the design.
- If using ECC, ensure that the board supports a full 8-bit wide data byte lane for the ECC check bits. In previous versions of MPMC, only 4 physical bits were used for ECC, but this and all future versions of MPMC require the full byte lane to be present to accommodate the PHY data calibration algorithm.
- During memory initialization (following reset), the Virtex-4 and Virtex-5 DDR and DDR2 MIG-based PHY writes to the top locations in memory to set up the write training pattern. This can affect memory address between {C_MPMC_HIGHADDR - 0xFF} and C_MPMC_HIGHADDR; therefore, after any reset, these locations in memory are overwritten. Software code should not be stored in the top 0x100 address locations of memory. If you are using shadow memory, review the "[Address Encoding](#)," page 48 to determine which address locations will be overwritten.
- In multi-rank systems, the MIG PHY will only calibrate its data capture timing to one of the ranks on one of the DIMMs. Differences in timing, process variation across memory devices, or bus loading affects across ranks will reduce timing margin and can affect the frequency range of operation.

Note: Multi-rank and multi-DIMM systems are not tested or characterized by MIG. You must ensure that the maximum skew and signal integrity is controlled across ranks. The use of multi-rank designs is strongly discouraged. The use of multi-DIMM designs is currently unsupported.

- Ensure that the map tool is run with the option `-pr b` set to so the tools will pack internal flip-flops into IOBs.

Important Notes on MIG Board Compatibility

Many existing evaluation boards were developed before the MIG tool was finalized, so it is often the case that Xilinx or third-party evaluation boards do not have exactly the same pinout that would be generated by supported MIG versions. However, these boards usually come with a predefined MPMC UCF file that is suitable for that board to function correctly.

Some Xilinx boards, including the Spartan-3E Starter Kit (XC3S500E), Spartan-3E 1600E MicroBlaze Development Kit, and the Spartan-3A/-3AN Starter Kit boards, have a pinout that significantly deviates from normal MIG pin-out assignment rules. For these boards, a parameter to MPMC called `C_SPECIAL_BOARD` is used to modify the internal PHY logic to compensate for the memory pinout on these boards. A predefined MPMC UCF file is needed for these boards. Systems created using the BSB have the necessary `C_SPECIAL_BOARD` and UCF settings. Having more severe MIG pinout violations means the Xilinx Spartan-3E Starter Kit (XC3S500E) and Spartan-3E 1600 MicroBlaze Development Kit should not have their memory clocks run above 100 MHz. Other boards, including the Virtex-5 FXT FPGA ML507 Evaluation Platform, might also have a sub-optimal pinout that limits the maximum operating frequency of the memory interface.

Do not copy the pinout of evaluation boards that do not match a supported MIG-generated pinout. All user boards must be designed exactly to the pinout provided by the prescribed version of MIG. Non-MIG pinouts might not be supported and might not be robust. User boards must also follow all layout recommendations specified in the *MIG User Guide*, including special trace delays for the `DQS_DIV_O` to `DQS_DIV_I` loopback signal (Spartan-3 PHY).

It is highly recommended that, if you are developing a new custom board, run the MIG design with hardware testbench logic or an MPMC test design through the full ISE tool flow to verify that proposed pinouts will implement properly through the ISE tools.

MIG/MPMC Tool Flow

This section outlines the MPMC to MIG tool flow. MIG is a tool that is delivered within the Xilinx CORE Generator™ software tool. There are three options in the MIG GUI for using MIG with MPMC:

- [Standalone MIG GUI Flow](#)
- [Integrated MIG GUI Flow](#)
- [Hybrid Flow](#)

The following subsections describe the options.

Standalone MIG GUI Flow

The Standalone MIG GUI flow offers the highest degree of flexibility because the full capability of the MIG GUI is made available. This standalone flow also provides greater visibility into the intermediate files and to the steps for generating an MPMC UCF file from the output of the MIG GUI.

This option is also useful if you use a standalone MIG-based memory controller on the same hardware board platform. This Standalone MIG GUI is the flow used in previous versions of MPMC. The following steps describe how to use the Standalone MIG GUI option:

1. Ensure that you have downloaded any available Core Generator technology IP updates.
2. Create a Core Generator project with the same FPGA device and package to match the EDK project settings. The MIG tool is available under the View By Function tab > **Memories and Storage Elements > Memory Interface Generators**.
3. Ensure that MIG is set to Verilog output mode (**Project > Project Options > Generation Tab > Design Entry > Verilog**).
4. Run the MIG tool version from Core Generator, set up memory part information, data widths, I/O banking locations, and so forth. (If necessary, click the User Guide button and review the board layout requirements for the memory interface.)
5. Click the Generate button to generate a MIG pinout.

Note: Clicking the Generate button also produces a MIG hardware testbench design at: `<coregen_project>/mem_interface_top_withtb`. This testbench design can be run as a standalone hardware design on the board to help test/debug the memory PHY interface.

6. After running the MIG tool, take the UCF file created by MIG located at:
`<coregen_project>/mem_interface_top_withouttb/par/mem_interface_top.ucf`.
7. Run the `convert_ucf.pl` script as outlined below in "[Converting MIG UCF to MPMCv.5 UCF](#)" to convert the MIG UCF file to an MPMC UCF file, then copy the UCF information into the `<EDK_Project>/data/system.ucf` file.
8. Update the UCF file to include the correct port names for your design.

For example, the `convert_ucf` script will produce port names like `fpga_0_DDR_SDRAM_DDR_DQ` and `fpga_0_DDR_SDRAM_DDR_Addr_pin` similar to how BSB names its ports. Those MPMC UCF port names should be updated to reflect the actual top-level port names in the `system.mhs` file.

9. Ensure that the MPMC Core is configured for the correct memory part and memory configuration in XPS using the MPMC GUI. The MPMC Core memory parameters must match the settings in the MIG GUI for the UCF constraints to match the logic that is generated.
10. Retain the MIG project file with the MPMC project files. The original MIG project file might be needed to modify the design, revise to a newer version of MPMC, or for debug and testing.

Note: The UCF file provided with BSB-generated designs might not match the format and values provided by the MIG UCF conversion script due to the use of Xilinx and third-party boards that do not have an exact MIG-generated pinout. In many cases these UCF files are manually generated and maintained.

Converting MIG UCF to MPMCV5 UCF

In the Standalone MIG GUI flow, you must run a script to modify the MIG-generated UCF file to be compatible with the MPMC design. This involves changing the instance and internal path names of the constraints to match MPMC. Note that this manual UCF conversion step is not required when the Integrated MIG GUI Flow is used.

A script is provided at:

```
<EDK_Install_Dir>/hw/XilinxProcessorIPLib/pcores/mpmc_<version>/data/convert_ucf.pl to assist with the process of converting a MIG UCF into an MPMC UCF.
```

After running the MIG tool to generate a Verilog MIG design, take the UCF file from the /mig_<version>/user_design/par directory and execute this script in a shell where ISE® and EDK tools are in the path environment:

```
xilperl <EDK InstallDir>
/hw/XilinxProcessorIPLib/pcores/mpmc_<version>/data/convert_ucf.pl
[--mhs <MHS File>] <MIG UCF> <OUTPUT UCF>
```

The optional `-mhs` flag with the entire system `<MHS File>` provided as an argument will use the MHS net names for the memory signals in `<OUTPUT_UCF>`. The `<OUTPUT_UCF>` file has UCF path names modified to match MPMC path names. You must add the contents of the `<OUTPUT_UCF>` file manually into the user EDK project UCF, such as `<EDK_Project_Dir>/data/system.ucf`.

This script is provided to assist with the process of translating the UCF but it does not necessarily generate a complete working UCF to use with MPMC. You must still verify the output of the script and you might need to adjust it to fit a particular design. Unless the `--mhs` option is used, the script does not completely translate the names of clock, reset, or memory I/O signals to match the top level port names in the EDK design. You must translate these signal names to match your design.

Migrating an MPMCV3 Design to MPMCV5

If using the Virtex-4 DDR or DDR2 MIG PHY or the Virtex-5 DDR PHY, no revision (revup) is required. A revup is required for Spartan-3 DDR/DDR2 and Virtex-5 DDR2 MIG PHY designs. You can run MIG and use the **MIG > Update Design** option to open the MIG v1.73 project file and generate a new constraint file. Using MIG to update the constraint file is the strongly recommended flow. For cases where the **MIG > Update Design** flow is not possible, the following script is provided to revup an MPMC v3.00.a or v3.00.b design that uses a MIG v1.73 PHY to an MPMC v5.00.a design that uses a MIG PHY.

- To revise a Spartan-(3/3A/3AN/3E/3ADSP) DDR or DDR2 design where the **MIG > Update Design** flow is not practical, execute the following commands in a shell where ISE tools are in the path:

```
cd <EDK_Install_Dir>/hw/XilinxProcessorIPLib/pcores/mpmc_<version>/data
xilperl revup_s3_ucf.pl <MPMC v3 UCF> <USER UCF>
```

where:

- `<EDK_Install_Dir>` is the directory in which the EDK is installed.
- `<MPMC v3 UCF>` is the name of your UCF file from your `mpmc_v3_00_a` or `mpmc_v3_00_b` project.
- `<USER UCF>` is the UCF file to be used in your MPMC project. Delete any constraints containing "RLOC_ORIGIN" statements because these are no longer applicable.

To revise a Virtex-5 DDR2 design from MPMCV3 to MPMCV5, a **MIG > Update Design** flow must be followed. The steps are:

1. A MIG project must be generated, if not already existing, using either the integrated or standalone MIG flow.

Note: Because a pinout is already available, all banks can be selected in the bank selection screen.

2. If the default MIG pinout generated is not the desired pinout, run the **MIG > Update Design** flow using the MIG project, and a UCF with the desired pinout in a MIG-formatted UCF.

Note: The input UCF parsing to the **MIG > Update Design** flow currently requires strict adherence to the formatting similar to MIG UCF output for complete Update Design UCF parsing.

Warnings in this step relating to missing pins in the UCF can be ignored if similar pins are not used by MPMC. For example the `clk200_p`, `clk200_n`, `sys_rst_n`, `phy_init_done`, and extra `ddr2_cs_n`, `ddr2_odt`, `ddr2_ck`, and `ddr_ck_n` ports can be ignored during this step.

3. If using the standalone MIG flow, convert the MIG UCF to MPMC and include into the system UCF by completing the steps in "**Converting MIG UCF to MPMCV5 UCF**," [page 59](#). This step can be skipped when using the integrated MIG flow as MPMC will convert and manage the final UCF.

More information on the MIG Update Design flow can be found in the MIG documentation. "[Reference Documents](#)," [page 193](#) contains a link to that documentation.

Migrating an MPMCV4 Virtex-5 DDR2 Design to MPMCV5

To migrate an existing Virtex-5 DDR2 design from MPMCV4 to MPMCV5, the following steps are necessary. All other MIG PHY types do not need special consideration to be revised.

Note: Tool errors will be reported if these steps are not performed.

1. In the MHS file, the `C_MEM_DQS_IO_COL` and `C_MEM_DQ_IO_MS` parameters are no longer used and must be removed.
2. In the UCF file, MPMC specific constraints containing `AREA_GROUP` and `RLOC_ORIGIN` must be removed.

Integrated MIG GUI Flow

The Integrated MIG GUI flow offers a simplified method for working with the MIG GUI to generate a memory interface pinout and constraint file. In this flow, the MIG GUI is invoked from within the MPMC GUI and the output of the MIG tool is imported automatically into EDK. This flow has fewer steps for running MIG to generate MPMC pinouts and constraints. The integrated flow seeds the Core Generator software project files and the MIG GUI project files automatically and eliminates some of the option screens from within the MIG GUI by inferring them from the MPMC settings.

The disadvantage of this flow is that it reduces visibility into the tool interactions which can make debugging, handling special cases, or workarounds more difficult to implement. This flow is recommended for new designs. For designs with existing MPMC and MIG projects built under the previous Standalone MIG GUI flow, it is recommended that the standalone flow continue to be used although you can use the integrated flow to update an older MIG project.

Note: BSB MPMC designs use the standalone flow for handling the MPMC UCF constraints.

The following steps describe the integrated flow. Refer to the *MIG User Guide* for more information on options in the MIG GUI.

1. Ensure that you have installed the Core Generator software and any available updates. Although the Core Generator software is not invoked in GUI mode, the tool is still executed at the command line by underlying scripts.
2. Using the MPMC IP configuration GUI, configure MPMC for your application. This process is described in "[IP Configuration Graphical User Interface](#)," page 189.
3. Configure MPMC as needed; ensure the Memory Interface tab has all the memory information.
4. Click the MIG Settings tab, and check the USE Integrated MIG GUI Flow box to enable the integrated MIG GUI flow. This will allow the **Launch MIG** button to be active; click this button to open MIG.
5. In the MIG **Output Options** window, Select **Create Design>Next**.
 - a. Note that you can use the **Verify UCF File** option to verify a user-modified UCF file when testing an alternate pinout that might be better for board layout.
 - b. The **Update Design** option allows an older MIG project file to be updated to the current version of MIG.
6. In the MIG **Pin Compatible FPGAs** window, check any other devices that the pinout must use to be compatible, then click **Next**.
7. In the MIG **FPGA Options** and **Extended FPGA Options** windows, make the appropriate selections, then click **Next**.
 - a. The selection of Single Ended or Differential System Clock does not affect the MPMC design, but it is useful to set correctly so that any standalone example MIG design generated by this project will match the board.
 - b. For Spartan designs, select the top/bottom clock input location to match the location of the Global Clock Buffer (BUFG) driving the memory clock (Port MPMC_Clk0) in the final system design.
8. In the MIG **Reserve Pins** window, select any pins that you want the MIG tool not to use in its pin assignments.

Note: The **Read UCF File** option only reads in a specially formatted file listing pins to prohibit. Consult the MIG documentation to make sure the correct file formatting is used to reserve pins.
9. In the MIG **Bank Selection** window, select the banks to which you want to allow MIG to assign pins. You must select enough banks to meet the required pin count totals.
10. In the MIG **Summary** window, review the options and click **Next**.
11. In the MIG **Memory Model License** window, mark the checkbox if memory models are desired, then click **Generate** to run MIG and create the constraint files. This returns you to the MPMC GUI.
12. The models created are raw simulation models for memory. To use these models in an MPMC simulation, you must manually create a testbench and properly connect it to the top-level EDK system.
13. After returning to the MPMC GUI, click **OK** to return to XPS. The MIG constraints are passed into a core-level constraint file that is merged into the ISE tools during ngdbuild. The core-level constraint file contains the memory interface pin location, timing, and area constraints needed for the design. The EDK `system.ucf` file is not modified.

Additional Important Information for the Integrated MIG GUI flow

1. The MIG project files that are generated in this flow are stored in the `<EDK_Project_File>/__xps/mig`. It is very important to backup and retain these files for future use. Make sure that these files are also kept for project archival or whenever the EDK project needs to be copied or moved.
 - a. The raw MIG pinout and constraints information are located in `<EDK_Project_Dir>/__xps/mig/gui/*/user_design/par/*.ucf`.
 - b. The MIG ucf file that is converted to have MPMC compatible instance names is located at `<EDK_Project_Dir>/__xps/mig/platgen/*/user_design/par/*.ucf`. This becomes the core level UCF constraints that are passed to the EDK flow.
 - c. The core level UCF used by the EDK/ISE flow during ngdbuild is in `<EDK_Project_Dir>/implementation/<MPMC name>/<MPMC name>.ucf`.
 - d. To completely clean away an old run of the MIG GUI to start the process over, remove the `<EDK_Project_Dir>/__xps/mig` directory. This directory contains all the state information for the integrated MIG flow.
2. In the integrated MIG flow, the constraints and some parameters are handled automatically:
 - a. The EDK `system.ucf` file should not contain any memory interface pinout, timing, or placement constraints as they are handled in the integrated flow using the core-level UCF files.
 - b. The parameters `C_NUM_IDELAYCTRL` and `C_IDELAYCTRL_LOC` are set automatically for Virtex-4 and Virtex-5. See "[IDELAY Controller](#)," page 68 for more information.
3. Verify that design correctly runs through all back end tools, is correctly placed and routed, meets timing, and shows no errors or significant warnings before committing the generated pinout to board layout.
 - a. Verify in the PAR report that all memory interface pins are "LOCed". Check the `.pad` file to view or debug the final memory interface pin locations and IO standards.
 - b. Review the information in this document and the *MIG User Guide* to ensure key aspects of the MIG PHY such as the template router (Spartan-3) or DIRT strings (Virtex-5) have been placed and routed correctly.
4. When the integrated MIG flow is enabled (`C_USE_MIG_FLOW = 1`), MPMC scripts are invoked during the Platgen EDK build phase that check for changes to MPMC configurations that would require MIG to be rerun. If the script flags an MPMC change such as a new memory width, the script will generate an error that asks the user to rerun the MIG GUI.
5. Perform a hardware clean in XPS before rebuilding a design where the MIG GUI settings have been changed.

Hybrid Flow

A hybrid flow is possible where the integrated GUI flow is run initially to generate constraints and then the user changes to the standalone flow for the EDK system builds.

In this case:

- The `<EDK_Project_Dir>/__xps/mig/platgen/*/user_design/par/*.ucf` file can be copied into the EDK `system.ucf` and the `C_USE_MIG_FLOW` parameter can be set back to 0. This will cause EDK to process MIG PHY constraints using the normal system level UCF file. For the conversion to work correctly:
- The UCF file might need to be edited for the correct top-level port names, and all instance or net names inside MPMC must be prefixed with the MPMC EDK instance names. For example:

Core Level UCF:

```
INST
  "DDR2_SDRAM/mpmc_core_0/gen_v5_ddr2_phy.mpmc_phy_if_0/u_phy_io_0/gen_dqs[0]
```

```
.u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y222";
```

System UCF (MPMC has INSTANCE parameter name of "DDR2_SDRAM"):

```
INST
```

```
"DDR2_SDRAM/DDR2_SDRAM/mpmc_core_0/gen_v5_ddr2_phy.mpmc_phy_if_0/u_phy_io_0/g  
en_dqs[0].u_iob_dqs/u_iddr_dq_ce" LOC = "ILOGIC_X0Y222";
```

- The C_NUM_IDELAYCTRL and C_IDELAYCTRL_LOC parameters (Virtex-4 and Virtex-5) must be set manually. The values of these parameters can be seen in `<EDK_Project_Dir>/hdl/<MPMC Instance Name>` after running a hardware build once when `C_USE_MIG_FLOW = 1`.

This flow can be useful for debug because the user explicitly controls the MPMC constraints and parameters related to the MIG PHY.

Perform a hardware clean in XPS before rebuilding a design converted from integrated to standalone mode.

MIG-based PHY Design and Implementation Considerations by Device

MIG Spartan-3 Design Considerations

MPMC uses the Spartan-3 PHY from MIG to implement the physical level interface for Spartan- (3/3A/3AN/3E/3ADSP) platforms with DDR1 or DDR2 memory. The MIG Spartan-3 PHYs use very specific UCF constraints to constrain the pinout and placement of internal elements in the fabric. These constraints are specific to an individual Spartan-3 device and are not necessarily portable across devices (even if the different devices are in the same package).

Spartan- (3/3A/3AN/3E/3ADSP) devices can have data width limitations depending on part and package size. Verify data width compatibility through the MIG GUI or through the "Supported Devices" section of the *MIG User Guide*, [ug086.pdf](#), in the "Spartan-(3/3A/3AN/3E/3ADSP) FPGA to Memory Interfaces" section. The "[Reference Documents](#)," [page 193](#) contains a link to this resource.

MIG also has guidelines for the board layout of the memory interface so that the PHY functions properly. It is extremely important that any user boards be designed in compliance with the MIG pinout constraints and layout guidelines for Spartan-3 and other MIG PHY families.

MIG provides a GUI in the Core Generator tool that guides you through the process of generating the pinout and UCF constraints for their design.

MIG Spartan-3 PHY Use of Top/Bottom I/O Banks for Data Signals

MPMC does not support the use of top/bottom memory interface banks for the Spartan-(3/3A/3AN/3E/3ADSP) FPGA families. Do not choose top or bottom banks for the DQ or DQS pins of the memory interface in the MIG tool. You must choose the left or right side banks for the DQ or DQS pins of the memory interface pinout.

MIG Spartan-3 Placement of Calibration Control Area Group and BUFG Driving Memory Clock Port MPMC_Clk0

For Spartan-(3/3A/3AN/3E/3ADSP) designs, ensure that the system clock input pin is correctly specified to the MIG tool. This will cause the MIG tool to place the tap delay circuit and calibration control ("cal_ctl" area group near the BUFG corresponding to the top or bottom location of the system clock input pin). MIG does not allow the use of side BUFGs or DCMs to be directly driving the memory clock. Ensure that in the final design the "cal_ctl" area group corresponds to the location near the BUFG that drives MPMC_Clk0.

For the correct placement of the calibration control circuit near the BUFG, the UCF file uses a device dependent AREA_GROUP directive with SLICE location calculated by the MIG tool. Verify in FPGA Editor that the "cal_ctl" area group contains slice locations near the BUFG driving the memory clocks.

In addition to the area group, the UCF contains an RLOC_ORIGIN directive to align a critical section of the tap delay circuit into a single column inside the "cal_ctl" AREA_GROUP. In MIGv2.3 or later, the AREA_GROUP and RLOC_ORIGIN values should be calculated automatically. If an older version of MIG was originally used or if clock circuits have changed, it might be necessary to re-run the original MIG project files on the current version of MIG. With the standalone mode, constraints missing from older versions of MIG might need to be corrected. It is also recommended that users verify the correct placement of BUFG, calibration control, and tap delay circuits as described in the following examples.

Spartan-3 Top Bank Clock Selection

The MIG output UCF includes an AREA_GROUP constraint on "cal_ctl". The first value in the RANGE of this constraint is the value in the RLOC_ORIGIN constraint. Here is an example AREA_GROUP from a MIG output UCF:

```
AREA_GROUP "cal_ctl" RANGE = SLICE_X74Y190:SLICE_X85Y203;
```

The first value in the range is X74Y190. This is the RLOC_ORIGIN value and could be missing from MIG v2.2 or older UCF files. The syntax for the constraint that should be present in the UCF file is:

```
INST "infrastructure_top0/cal_top0/tap_dly0/l0" RLOC_ORIGIN = X74Y190;
```

Spartan-3 Bottom Bank Clock Selection

The MIG output UCF includes an AREA_GROUP constraint on "cal_ctl". To calculate the RLOC_ORIGIN constraint, take the first value in the AREA_GROUP RANGE and add 10 to the X coordinate. Here is an example AREA_GROUP from a MIG output UCF:

```
AREA_GROUP "cal_ctl" RANGE = SLICE_X74Y4:SLICE_X85Y17;
```

The first value in the range is X74Y4. The new RLOC_ORIGIN value is X84Y4 (because 10 is added to the X coordinate and could be missing from MIG v2.2 or older UCF files.) The syntax for the constraint that should be present in the UCF file is:

```
INST "infrastructure_top0/cal_top0/tap_dly0/l0" RLOC_ORIGIN = X84Y4;
```

Ensure that the "cal_ctl" AREA_GROUP is not located on the left or right sides of the chip and that the BUFG driving the MPMC_Clk0 port is located in the same side of the chip near the AREA_GROUP.

To verify that the tap delay circuit, "cal_ctl" AREA_GROUP, and BUFG driving the memory clock are all placed correctly, follow these steps:

1. Open the Post-PAR `design.ncd` and `design.pcf` files in FPGA Editor.
2. Select **Routed Nets** from the **List** window located at the top left-hand side. This will show all the routed net names of the design.
3. Enter `*tap_dly*/tap*` in the **Name Filter** window to select the `tap_dly` chain.
4. Select all the displayed nets, and click the **Apply** button on the right-hand side of the **Name Filter** window.
5. Zoom into the area in the **Array** window where the selected routes are highlighted.
6. If the tap delay circuit is properly constrained, the logic will be located in a single column in four sequential Configurable Logic Block (CLBs). This will indicate the `RLOC_ORIGIN` is correct. The following is a screen capture showing an example of the correct placement of the `tap_dly` circuits.

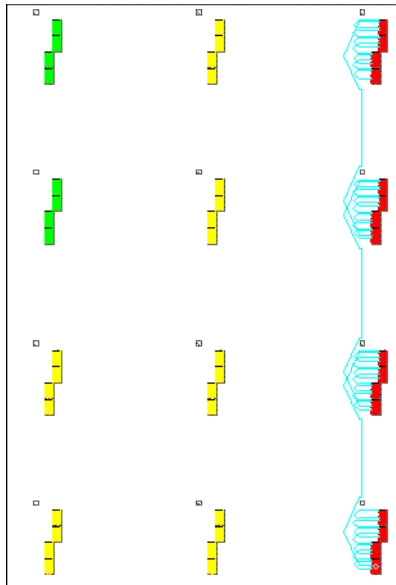


Figure 5: Correct Placement of `tap_dly` Circuits

7. Repeat steps 3, 4, and 5 using `cal_ctl` in the **Name Filter** window. If the `cal_ctl` circuit is properly constrained, the logic will be placed in the area near the tap delay circuit on the same side of the chip.
8. Find the global clock buffer (BUFG) driving the clock to the `MPMC_C1k0` port.
9. Go back to the List window, select **All Components** and sort the List window by Type. Find the `BUFGMUX` components in the Type column. Select the named `BUFG` that is driving the `MPMC_C1k0` port and verify it is located close to the tap delay circuit (on the same side of the chip).

MIG Spartan-3 PHY Template Router and DQ/DQS Data Capture Logic

The proper implementation of the DQ/DQS data capture logic requires specific pinout, timing, and placement constraints which include PIN, LUT, BEL, SLICE and MAXDELAY elements. This also includes the usage of template routes during Place and Route. The correct placement and routing of these components should be verified using FPGA Editor.

Refer to the *MIG User Guide*, under the “DDR2 Debug Guide”-> “Debugging the Spartan-3 FPGA Design” ->Verify Placement and Routing.” The steps needed to check the placement and routing of the Spartan-3 MIG PHY are described in that section.

MIG Spartan-3 PHY MAXDELAY Timing Constraints

When running the place and route tools, Spartan-3 MPMC designs might generate timing errors on the maxdelay constraints in the MPMC UCF. If this occurs, check the `<EDK_project>/implementation/system.twr` file for the errors. If the worst case path is within the range of allowable delays mentioned in the comments of the MIG UCF, the constraint can be relaxed to allow timing to pass. MAXDELAY constraints should not be relaxed beyond the range of values described in the UCF comments. It might take several iterations of the tools to resolve the MAXDELAY timing constraints so the design meets timing. Alternatively, you can proceed with the design and allow for false timing errors in the MPMC Spartan-3 PHY logic.

MIG Spartan-3 PHY Debug Tips and Hints

For Spartan-3 MIG designs that do not work, the following debugging steps could help in the debug process:

1. Perform the placement and routing checks using FPGA Editor.
2. Check that the MicroBlaze processor can connect to XMD and that simple XMD reads and writes to LMB block RAM are working. This establishes that the basic processor subsystem is working.
3. Check the PAR report and ensure that all I/O signals are located.
4. If reads from memory space return the same data value across all memory locations, this is a symptom of a hang in the PHY. This could be caused by any of the following:
 - Incorrect connections on the DQS_DIV_I/DQS_DIV_O loopback trace
 - Problems with the DQS nets
 - Incorrect template router nets (see #1)
 - Problems with the generation or conversion of the MIG UCF
5. For data corruption errors or hangs, additional debug might be required by using the ChipScope™ debugging tool to probe the signals inside the Spartan-3 PHY block. Signals that can be checked are:
 - a. PHY read FIFO data out signals (two FIFOs per byte lane, one for posedge data and one for negedge data). Monitor these signals to check the read data strobed in by the DQS signal.
 Read FIFO Data Out = `*mpmc_phy_if_0/data_path/data_read/fifo_0_data_out_r<*>`
 and
`*mpmc_phy_if_0/data_path/data_read/fifo_1_data_out_r<*>`
 - b. PHY read FIFO control signals. The signal `read_valid_data` is used to qualify read data being returned to the MPMC data path logic.

The `wr_addr` and `rd_addr_out` signals are gray code counters that implement the FIFO Read and Write pointers driving a dual-port LUT-based RAM for each data bit. A different set of FIFO pointers is implemented on each byte lane of data for `posedge` and `negedge` data.

```
*mpmc_phy_if_0/data_path/data_read/read_valid_data
*mpmc_phy_if_0/data_path/data_read/fifo_0_wr_addr_out<*><*>
*mpmc_phy_if_0/data_path/data_read/fifo_1_wr_addr_out<*><*>
*mpmc_phy_if_0/data_path/data_read/fifo_0_rd_addr_out<*><*>
*mpmc_phy_if_0/data_path/data_read/fifo_1_rd_addr_out<*><*>
```

- c. PHY calibration tap values. These calibration values determine what delay is being applied to the DQS signals to align the edges of the DQS strobe signal to the data.

```
*mpmc_phy_if_0/infrastructure/cal_top/cal_ctl/tapForDqs_r1<*>
```

- d. ChipScope tool usage notes:

Attaching ChipScope probes onto signals that are used by the template router could disrupt critical timing/routing nets. The ChipScope analyzer probes should be attached only to signals operating in the `MPMC_C1k0` and `MPMC_C1k90` clock domain.

The use of ChipScope on a Spartan-3 device could cause a conflict with the `BSCAN` element used for XMD access. This might require the use of XMD stub-based debug or might require that a test program be initialized into LMB block RAM to generate the memory transactions.

6. Isolate which byte lanes are causing hangs or data errors and investigate those byte lanes further in greater detail.
7. Implement the MPMC design using the Static PHY, get the static PHY to work, then compare results between static PHY and MIG PHY. For example, write with one PHY and read with the other PHY to help isolate the problem to reads or writes.
8. Run the standalone MIG hardware testbench design. This design consists of MIG PHY, controller, and hardware testbench that loops write/read data to memory. The MIG hardware testbench design includes preconfigured ChipScope probes and an error status LED output. If the standalone MIG testbench is not working, resolve it first. The standalone MIG testbench is a smaller and simpler design that might be easier to debug.
9. Verify that clock, control, data, power, and ground signals and board traces have the proper signal integrity and implementation on the board.

Note: If you are not familiar with creating hardware and software designs as outlined in these steps, or you are unfamiliar with the debug tool flows, practice building these designs using BSB and a supported Spartan-3 evaluation board such as the Spartan-3A Starter Kit or the Spartan-3A DSP 1800 Starter Kit.

Additional Information

Answer Records, Application Notes, and the *MIG User Guide* provide important information about the Spartan-3, Virtex-4, and Virtex-5 MIG PHY, the MIG UCF constraints, and board layout guidelines. These resources will help to debug and bring up MPMC designs using the MIG PHY:

- The Xilinx memory page contains the *MIG User Guide* and other relevant content.
- Answer Records contain useful design, debug, and implementation content.
- Application notes describe the operational theory and implementation of the underlying MIG PHYs.

The "[Reference Documents](#)," page 193 contains links to these resources.

Tips and Hints for Board Bring Up

The following are tips and hints that can be useful during board bring up:

1. During initial bring-up of MPMC on a new board, Xilinx recommends that you start with a single-port MPMC with PLB PIM. This design should be modeled off a simple BSB-based MicroBlaze processor design with caches disabled.
2. During initial bring-up, start running the memory controller at the minimum clock speed that the memory device supports to establish as a working baseline. Then increase the memory clock speed up to the desired frequency.
3. Use the MicroBlaze with a Single-Port MPMC design and connect with the XMD. Then perform simple reads and writes to establish whether basic memory transactions work.
4. Incrementally add complexity to the design by building it up to be closer to your desired MPMC system configurations. For example, after getting the single-port design to work, enable MicroBlaze caches and connect the MicroBlaze IXCL/DXCL ports to a three Port XXP MPMC design (2 XCL + 1 PLB). Establish that memory transactions continue to work with caches off and try the test with caches on to test burst read transactions from memory.
5. Run the MPMC extended memory test application located at
`<EDK_Directory>/sw/XilinxProcessorIPLib/drivers/mpmc_<latestversion>/examples/mpmc_mem_test_example.c`.
6. Make use of the MPMC Debug register functionality described on "[MIG PHY Debug Registers](#)," [page 30](#). These debug registers allow you to read or set the MIG PHY calibration settings manually via a processor accessible interface.
7. Use default MPMC settings wherever possible. Customize or optimize the design parameters after you have a working baseline system.
8. Check the PAR report to ensure all memory IOs are "LOCed". Ensure that map is run with the **-pr b** option to ensure IOB flip-flop packing.
9. Try running a standalone MIG controller design. The MIG tool can produce an example standalone loopback design that may be useful for debugging physical level MIG PHY problems.
10. Check that the maximum clock frequency limitations of the underlying MIG PHY are not exceeded. See "[MIG PHY Supported FMAX](#)," [page 173](#) for more information. Also, be careful not to use a clock frequency below the minimum operating frequency of the memory device. For example some DDR2 devices have a minimum clock frequency specification of 125 MHz.
11. For Dual Rank or Dual DIMM designs, note the warnings that such designs are strongly discouraged. If a Dual Rank or Dual DIMM design is still used, try to establish a working baseline system accessing only one of the ranks of memory.

IDELAY Controller

For Virtex systems using the MIG PHY, instantiation of IDELAY controllers (IDELAYCTRL) is required to ensure that the IDELAY elements in the MPMC physical interface behave properly. The MPMC core parameter, `C_NUM_IDELAYCTRL`, determines the number of IDELAYCTRL elements to instantiate. By default, the MPMC instantiates one IDELAYCTRL without a location constraint (LOC), which causes the ISE implementation tools to replicate the IDELAYCTRL blocks across the entire FPGA.

For the Integrated MIG GUI flow, these parameters are passed automatically into the MPMC design from the MIG GUI but can be set manually to override these automatic values. Refer to "[Multiple MPMC IDELAYCTRL IP Designs](#)," [page 69](#) for an explanation of a special case when this mechanism alone will not work.

Single MPMC IDELAYCTRL Designs

For Virtex-5 designs and Virtex-4 designs with the MAP-timing option enabled, MAP will trim unnecessary IDELAYCTRLs. The default setting is sufficient for systems where there is one MPMC instance only, and no other IP in the system requires the use of IDELAY elements.

Multiple MPMC IDELAYCTRL IP Designs

For systems where there are multiple IP cores each using their own IDELAYCTRL element independently, such as two instances of MPMC or MPMC and PCI in the same system, special consideration must be given to the number of IDELAYCTRLs.

- Instantiate the correct number of IDELAYCTRL blocks for each IP as determined by the clock regions where the associated IDELAY elements are used. For these systems, it is necessary to set the correct value for C_NUM_IDELAYCTRL.
- Ensure the IDELAYCTRL element is associated with the correct clock region positions in the FPGA and is located using the C_IDELAYCTRL_LOC parameter.

For example, a design requiring two IDELAYCTRLs might have a core configuration of:

```
PARAMETER C_NUM_IDELAYCTRL = 2
PARAMETER C_IDELAYCTRL_LOC = IDELAYCTRL_X0Y4-IDELAYCTRL_X1Y3
```

Consult the ISE documentation for more information about IDELAYCTRL. The link to ISE documentation is available in "[Reference Documents](#)," page 193.

MPMC contains the following ports (see [Table 11, page 15](#)) that can be used to chain the IDELAY elements between IP blocks:

- MPMC_Idelayctrl_Rdy_I is AND'd with internal IDELAYCTRL RDY signals inside MPMC and signifies that memory initialization can begin.
- MPMC_Idelayctrl_Rdy_O port signals that the internal IDELAYCTRL RDY signals and the MPMC_Idelayctrl_Rdy_I are all high.

This is useful in situations where two IP blocks have I/O that share a common clock region and IDELAYCTRL element. You can connect the MPMC_Idelayctrl_Rdy_O to the MPMC_Idelayctrl_Rdy_I of downstream MPMC IP or other IP blocks. In this situation, explicitly set the C_NUM_IDELAYCTRL and C_IDELAYCTRL_LOC parameters and follow these instructions even if you are using the integrated MIG GUI flow. If these parameters are explicitly set, they will override the values passed in from the integrated MIG GUI.

The MPMC_Idelayctrl_RDY_0 outputs of upstream IP blocks with IDELAYCTRL can be tied to the MPMC_Idelayctrl_Rdy_I input.

- Ensure that the MPMC_Idelayctrl_Rdy_I and MPMC_Idelayctrl_Rdy_O ports are not connected in a circular manner over one or more IP blocks.
- MPMC_Idelayctrl_Rdy_I and MPMC_Idelayctrl_Rdy_O can be left unconnected when not needed. The EDK XPS tool ties MPMC_Idelayctrl_Rdy_I to high automatically when it is unconnected.

Static PHY Interface

The Static PHY interface available in MPMC is based on DCM phase adjustment. This PHY is used for SDRAM. For DDR/DDR2, it can also be used in cases when a MIG-based PHY is not available or cannot be used. The MPMC Static PHY interface is an alternative to the Memory Interface Generator (MIG)-based PHY interface. The Static PHY uses DCM-based fine phase adjustments to generate a read data capture clock instead of using IDELAYS or LUT-based delays to shift the input read data.

Note: Xilinx recommends that you design for, and use, a MIG-based PHY whenever possible for best results. MIG-based PHY interfaces offer greater timing margin, are more robust, and use fewer global clock buffer resources; the Static PHY is available when MIG-based PHY is not an option.

As an example, the Static PHY might be used when a legacy board that was designed for a different memory controller and does not use a MIG-compatible pinout. All new designs should target the use of the MIG-based PHY.

Static PHY contains the following topics:

- [Static PHY Implementation](#)
- [Static PHY Implementation Considerations](#)
- [Static PHY Interface Register](#)
- [Example Static PHY Calibration Algorithm](#)

Static PHY Implementation

The following figure shows the Static PHY interface read logic. Note that the `RDDATA_SWAP_RISE` stage is not present for SDRAM designs.

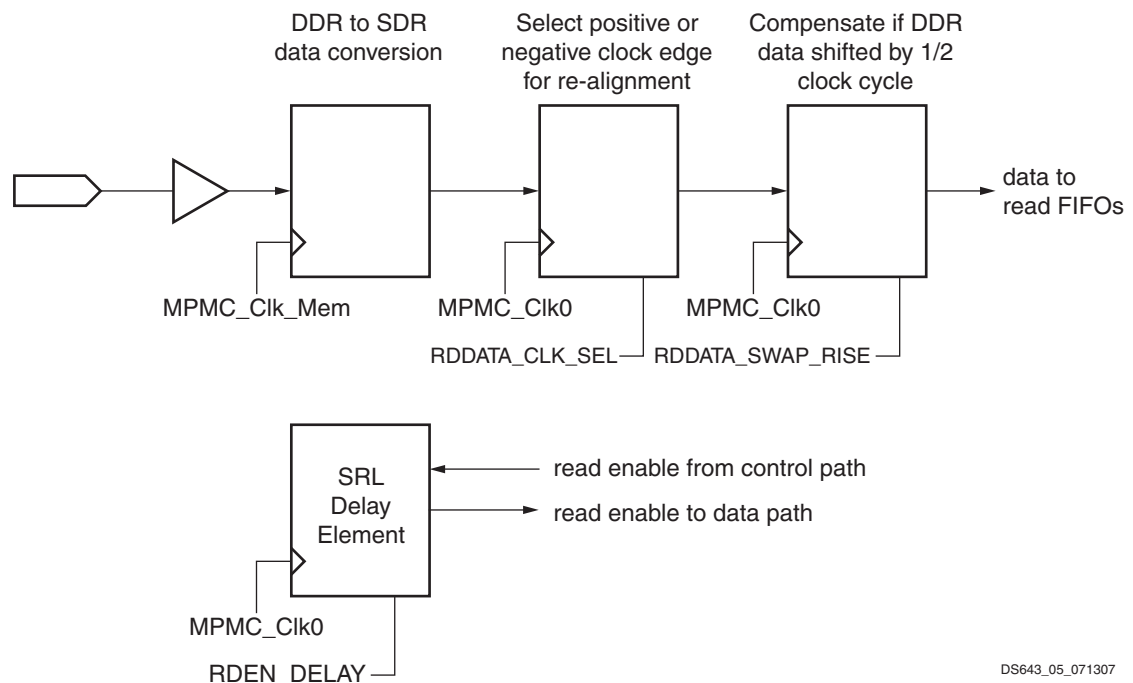


Figure 6: Static PHY Interface Read Logic

DS643_05_071307

The Static PHY processes the read data as follows:

1. First, the Static PHY registers the read data on a clock (`MPMC_Clk_Mem`), which is typically provided by an additional Digital Clock Manager (DCM). This clock is, typically, a phase-shifted version of `MPMC_Clk0`. The phase is set to maximize timing margin on the captured read data. The read data goes through input flip-flops and is converted to Single Data Rate (SDR) data that is aligned on the rising edge of `MPMC_Clk_Mem`.

2. The data is then re-registered into the main MPMC clock domain (`MPMC_Clk0`).

Depending on the phase relationship between `MPMC_Clk_Mem` and `MPMC_Clk0`, you might need to first re-register the data on the negedge of `MPMC_Clk0` before being registered on the positive edge. The selection using positive or negative edges of `MPMC_Clk0` depends on the relative phase of `MPMC_Clk_Mem`:

- If `MPMC_Clk_Mem` is 0 to +180 degrees ahead of `MPMC_Clk0`, register the data on the positive edge of `MPMC_0_Clk0`.
- If `MPMC_Clk_Mem` is 0 to -180 degrees behind `MPMC_Clk0`, register the data on the negative edge of `MPMC_Clk0`.

The `C_STATIC_PHY_RDDATA_CLK_SEL` parameter sets the default value for this selection. You can change this value using the software interface at a later time, if necessary.

3. Next, data from the Static PHY passes through a selector to determine how the DDR data from memory is aligned with respect to positive and negative edges of `MPMC_Clk_Mem`.

Depending on board layout and clock frequency, it is possible that the data that is registered on the posedge of `MPMC_Clk_Mem` should be registered on the negedge of `MPMC_Clk_Mem`.

This clock phase relationship makes the SDR data appear misaligned with respect to the pairs of DDR data from memory. In this case, half of the data appears on the first posedge of `MPMC_0_Clk0`, and half of the data appears on the second posedge of `MPMC_Clk0`. This stage is not present for SDRAM designs.

A multiplexor (MUX) lets you correct for the misalignment by selecting how the DDR data should be arranged into SDR data. The default value of the MUX is controlled by the `C_STATIC_PHY_RDDATA_SWAP_RISE` parameter. You can change this value using the software interface at a later time, if required.

4. The last part of the Static PHY consists of a shift register that can adjust the delay of the read enable signal to the data path (which causes read data to be pushed into the read FIFO).

Depending on the parameter settings, clock frequency, and board layout, the read data to the data path might appear on a different clock cycle relative to the read enable signal from the control path.

The control path sends a read enable to the PHY at the same time that it sends the read command to the PHY. The PHY then must delay this signal for a certain number of cycles to make the read FIFO push signal valid at the same time as the data coming out of the Static PHY.

The `C_STATIC_PHY_RDEN_DELAY` parameter sets the default value for this delay. You can change this value using the software interface at a later time, if required.

If this parameter is set incorrectly, and a read is performed, it is possible that the read data will be pushed into a different FIFO than where the data was intended. This typically will occur only if the read was not issued on port 0. If the data is pushed into the wrong FIFO, the result can be identified by a processor hang that can be recovered from by resetting the system only. The correct setting for this parameter depends on MPMC pipeline configurations, clock frequency, and board layout, but typically this parameter is set to 5, 6, or 7 for DDR and DDR2; and 4, 5, or 6 for SDRAM.

Additionally, if you are generating the `MPMC_Clk_Mem` with a DCM that is enabled to use variable phase shift, the Static PHY Control Register interface provides an easy way to control the PSEN and PSINCDEC ports of the DCM. MPMC provides a control port that can be connected to a DCM control port. This allows you to control the DCM phase adjust via the MPMC control registers. See the "[Static PHY Interface Register](#)," page 74 for more details.

Static PHY Implementation Considerations

The important implementation considerations when using the Static PHY are:

- [Control Register Values](#)
- [Timing Constraints](#)
- [DCM Phase Adjust Port](#)
- [Matching Delay Traces](#)

The following subsections detail the implementation considerations.

Control Register Values

If you already know the control register values needed for the Static PHY to work with your board and this value is stable, you can choose to fix these values so that the PHY operates correctly upon power-on. You can do this by setting the DCM phase adjust and parameters values for `C_STATIC_PHY_RDDATA_CLK_SEL`, `C_STATIC_PHY_RDDATA_SWAP_RISE`, and `C_STATIC_PHY_RDEN_DELAY` as necessary.

Timing Constraints

When using the Static PHY, timing constraints are needed; you must set the UCF constraints to ensure that the maximum delay for data signals passing from `MPMC_Clk_Mem` to `MPMC_Clk0` clock domains is 1/2 the period of `MPMC_Clk0`. The following is an example of such a timing constraint where no dynamic DCM phase adjustment is used:

```
NET <MPMC_instance_name> /*rd_data_rise_in* MAXDELAY = <half_clock_period>;
#DDR/DDR2
NET <MPMC_instance_name> /*rd_data_fall_in* MAXDELAY = <half_clock_period>;
#DDR/DDR2
NET <MPMC_instance_name> /*rd_data_rise_rdclk* MAXDELAY = <half_clock_period>;
# SDRAM
```

If you plan to adjust the DCM phase settings dynamically to locate an optimal DCM clock phase shift, it is recommended to tighten the timing constraint so there is more margin to account for the potential `MPMC_Clk_Mem` phase shift range:

```
NET <MPMC_instance_name> /*rd_data_rise_in* MAXDELAY = 1000 ps; #DDR/DDR2
NET <MPMC_instance_name> /*rd_data_fall_in* MAXDELAY = 1000 ps; #DDR/DDR2
NET <MPMC_instance_name> /*rd_data_rise_rdclk* MAXDELAY = 1000 ps; # SDRAM
```

Note: Ensure that the map is run with the option `-pr b` set to ensure that the tools pack internal flip-flops into the IOBs. You might also need to relax the maxdelay value from the examples above to meet timing depending on the speed of the FPGA.

DCM Phase Adjust Port

The MPMC DCM phase adjust control port lets you increment and decrement the phase adjustment value of the DCM. The phase adjustment that the MPMC performs is relative to the initial phase value set in the DCM. You must instantiate the DCM itself outside of the MPMC.

You must configure the DCM in the proper operating mode with the necessary parameters set for your system. The MPMC provides the control signals to generate commands to change the phase of the DCM only. It does not check that the DCM is configured properly, and does not check if the DCM phase adjustment range is exceeded.

The following is an Microprocessor Hardware Specification (MHS) file example of how a DCM can be connected to MPMC to allow the MPMC Static PHY control registers to control DCM phase.

```
BEGIN mpmc
.
.
.
PORT MPMC_DCM_PSINCDEC = Static_Phy_DCM_PSINCDEC
PORT MPMC_DCM_PSEN = Static_Phy_DCM_PSEN
PORT MPMC_DCM_PSDONE = Static_Phy_DCM_PSDONE
END

BEGIN dcm_module
PARAMETER INSTANCE = dcm_2
PARAMETER HW_VER = 1.00.d
PARAMETER C_CLK0_BUF = TRUE
PARAMETER C_CLKIN_PERIOD = 10.000000
PARAMETER C_CLK_FEEDBACK = 1X
PARAMETER C_DLL_FREQUENCY_MODE = LOW
PARAMETER C_PHASE_SHIFT = 0
PARAMETER C_CLKOUT_PHASE_SHIFT = VARIABLE
PARAMETER C_EXT_RESET_HIGH = 0
PORT CLKIN = MPMC_Clk0
PORT CLK0 = MPMC_Clk_Mem
PORT CLKFB = MPMC_Clk_Mem
PORT RST = DCM_1_lock
PORT LOCKED = DCM_all_locked
PORT PSCLK = MPMC_Clk0
PORT PSINCDEC = Static_Phy_DCM_PSINCDEC
PORT PSEN = Static_Phy_DCM_PSEN
PORT PSDONE = Static_Phy_DCM_PSDONE
END
```

Matching Delay Traces

Because the Static PHY uses a DCM to capture all the read data, ensure that boards designed to use the Static PHY have matched delay traces across all data lanes. This reduces the skew across the data bits and improves timing margin. Place data pins in the same bank or in adjacent banks to further reduce skew across data bits (clock tree skew in the FPGA is smaller if pins are placed together).

Static PHY Interface Register

You can configure the Static PHY interface statically by using the following parameters to set the startup values for the signals that control the Static PHY:

- C_STATIC_PHY_RDDATA_CLK_SEL
- C_STATIC_PHY_RDDATA_SWAP_RISE
- C_STATIC_PHY_RDEN_DELAY

You can then change these values dynamically by using the PLB Control register. Additionally, you can change the phase delay of a DCM that generates MPMC_Clk_Mem using the PLB Control register. The following table describes the Static PHY Interface Register (SPI).

Table 37: Static PHY Interface Register (SPI)

Bit(s)	Name	Core Accesses	Reset Value	Description
0:3	RDEN_DELAY	R/W	C_STATIC_PHY_RDEN_DELAY	Sets the number of cycles to delay the read enable push) to the read FIFOs. See "Static PHY Implementation," page 70 for more information. This value is typically 5, 6, or 7 for DDR/DDR2 and 4, 5, or 6 for SDRAM.
4	RDDATA_CLK_SEL	R/W	C_STATIC_PHY_RDDATA_CLK_SEL	Sets the read data to be re-registered on the positive or negative edge of MPMC_Clk0: 1 = Positive Edge 0 = Negative Edge
5	RDDATA_SWAP_RISE	R/W	C_STATIC_PHY_RDDATA_SWAP_RISE	Sets the DDR read data to be shifted by 1/2 clock cycle relative to the SDR clock: 0 = no shift 1 = 1/2 clock cycle shift. This register is present for DDR/DDR2 designs only.
6	UNUSED		N/A	N/A
7	FIRST_RST_DONE	R	0	Set to 0 during first reset to static PHY; set to 1 afterwards. This prevents the control register from being reset to default values after initial reset.
8	DCM_PSEN	R/W	0	Set to 1 to perform one DCM phase shift increment or decrement. Self Clearing. Direction of phase shift is determined by DCM_PSINCDEC.
9	DCM_PSINCDEC	R/W	0	1 = perform DCM phase shift increments. 0 = perform DCM phase shift decrements. Only valid with DCM_PSEN 1 = enable phase shift by +1 tap 0 = enable phase shift by -1 tap
10	DCM_DONE	R/W	0	Set to 1 when DCM phase shift increment or decrement is complete. This bit must be cleared by MPMC. DCM_PSEN is not set to 1 again until DCM_DONE is set and cleared.
11	INIT_DONE	R	0	Set to 1 when initialization is complete; otherwise set to 0.
12:15	UNUSED	N/A	N/A	N/A
16:31	DCM TAP VALUE	R	0	A 10-bit number (sine-extended to form a 16-bit value) which represents the status of the software-derived DCM tap value. The tap value is relative to the initial phase tap value set in the DCM. The value of this register assumes that the DCM PHASE_SHIFT parameter was initially set to 0; otherwise it reports the relative phase offset only. This value can be incorrect if the DCM phase is shifted beyond the DCM allowable adjustment range.

Example Static PHY Calibration Algorithm

The following steps provide a flow chart on how to use the Static PHY Interface PLB control register to calibrate the PHY interface automatically. The software must be run before any other writes or reads are sent to MPMC. Additionally, the software instructions must be stored in block RAM until the MPMC memory is calibrated.

1. Wait for `INIT_DONE` bit to be set.
2. Ensure `DCM_TAP_VALUE` equals the initial DCM phase shift setting. If it is not equal, stop here, report error, and rebuild hardware with DCM phase shift set to 0.
3. Set `RDDATA_CLK_SEL` to 0.
4. Set `RDDATA_SWAP_RISE` to 0 (DDR and DDR2 only.)
5. Set `RDEN_DELAY` to the minimum value, typically 0. (This can cause a processor hang if you are not using port 0 for calibration reads. If this is the case, you might need to increment the minimum start value and maximum end value for this parameter. See "Static PHY Implementation," page 70 for more details.)
6. Set DCM phase shift to the minimum value using `DCM_PSEN`, `DCM_PSINCDEC`, and `DCM_DONE`.
7. Write and verify pattern in memory without data cache. If there is a mismatch, clear the valid count, and skip to step 11.

Note: Using a large pattern that introduces a lot of data bus toggling might improve calibration results, but will also increase calibration time. Also note that the data written to memory for the calibration pattern should ensure a high amount of data bit toggling for the memory width being used.

For example, writing `0x00000000` followed by `0xFFFFFFFF` will successively toggle all bits for a 32-bit memory but will not cause as many data bit transitions on an 8- or 16-bit memory.

8. Enable data cache.
9. Read pattern back and verify. Flush and invalidate the data cache.
10. Increment the DCM phase shift, keeping track of how many patterns were read back correctly.
11. Repeat 8 through 11 until maximum phase shift value is reached.
12. If the number of correctly read patterns in a row is greater than an acceptable threshold, set DCM phase to midpoint of largest found acceptable range.
13. Increment `RDEN_DELAY` by 1. 17. Repeat steps 3 through 13 until maximum `RDEN_DELAY`.
14. Set `RDDATA_SWAP_RISE` to 1. Repeat steps 5 through 14 (DDR and DDR2 only.).
15. Set `RDDATA_CLK_SEL` to 1. Repeat steps 4 through 15.
16. If this step is reached, calibration was not possible. Stop and report error. Your settings are saved until you power down your board, even with an MPMC system reset. Ensure that a system reset will not force your DCM phase shift setting to revert back to the initial value. If this is the case, you need to modify your reset structure, rebuild your hardware, and re-run the calibration.

Note: You might want to build a simple single port MPMC system with a Static PHY, use a software program to characterize the ideal set of static PHY settings for that board, then initialize those settings into the design or reduce the search range of your calibration program.

SDRAM PHY Interface

The SDRAM PHY is the interface between the Single Data Rate Access Memory (SDRAM) and the MPMC control path, address path, and data path. The interface supports Virtex-4, Virtex-5 and Spartan-(3/3A/3E/3AN/3ADSP) platforms. The SDRAM PHY uses the DCM phase adjustment based read data capture scheme used for the Static PHY. Refer to the "[Static PHY Interface](#)," page 70 for more information.

Features

The SDRAM PHY includes the following supported features:

- Column Access Strobe (CAS) latencies of 2, 2.5, and 3
- SDRAM data widths of 8, 16, 32, and 64
- DIMMs (both registered and unregistered)
- Multiple Memory Ranks
- ECC support

The PHY interface works from `MPMC_Clk0`, which issues the control, data, and address signals to memory. To clock SDRAM memory, the PHY uses an inverted version of `MPMC_Clk0`. This gives a 1/2 clock period setup and hold time for memory control and address signals.

To drive the data bus with valid Read data, the memory requirement is $CAS_LATENCY - 1 + T_{ac}$ time (Access Time) after registering the read command. The following figure illustrates the SDRAM Read data timing.

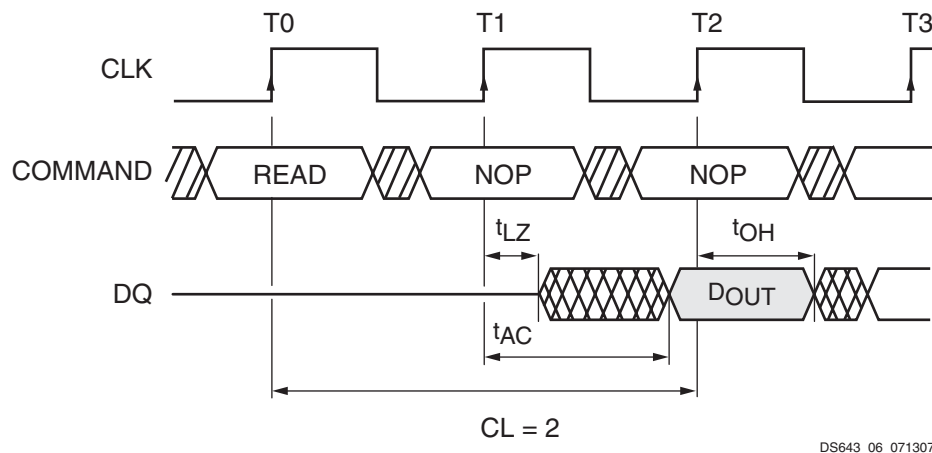


Figure 7: SDRAM Read Data Timing

The T_{ac} is typically between 5 and 6ns. The data capturing logic works on the positive edge of `MPMC_Clk_Mem`, which can be varied using the Static PHY interface or connected to a fixed clock source with the correct relative phase. This capture clock can be adjusted to maximize the size of the timing window for capturing data. The captured data is pushed into the read Data path FIFOs on `MPMC_Clk0`.

The SDRAM PHY performs the power up initialization sequence and configuration of SDRAM memory with user-specified values from the MHS file.

When MPMC is configured for SDRAM, the Static PHY is instantiated automatically. It is not necessary to set `C_USE_STATIC_PHY = 1` also.

Low Frequency SDRAM Clock and DCM Phase Adjustment Limits

In lower frequency SDRAM designs, generally under 100 MHz, the fine phase adjustment range of the DCM might limit the available search range of the capture clock. This can limit the amount of margin in the Read data capture window depending on board delays and part delays. If the Read data capture window is too small because the DCM fine adjustment range is limited, consider using DCM outputs of CLK90, CLK180, or CLK270 to best center the DCM phase search window around the Read data. This might require board delay or oscilloscope-based analysis.

For very low frequency SDRAM designs such as below 50 MHz, a fixed DCM output of CLK0, CLK90, CLK180, or CLK270 can usually be directly connected to MPMC_Clk_Mem because this results in a sufficient Read data capture window. In this case you might need to use experimentation or oscilloscope-based analysis to find the best fixed clock phase (0, 90, 180, or 270). A modified static PHY calibration program or experimentation can then be used to find the optimal settings for C_STATIC_PHY_RDDATA_CLK_SEL and C_STATIC_PHY_RDEN_DELAY for the given MPMC_Clk_Mem clock. CLK90 is a recommended starting point for very low frequency designs.

Connecting Memory to the PHY Interface

The PHY interface permits connections to 8-, 16-, 32-, and 64-bit SDRAM, DDR, or DDR2 memories.

The following table lists the unique signals and parameters for each available PHY layer (excluding standard memory and clock signals):

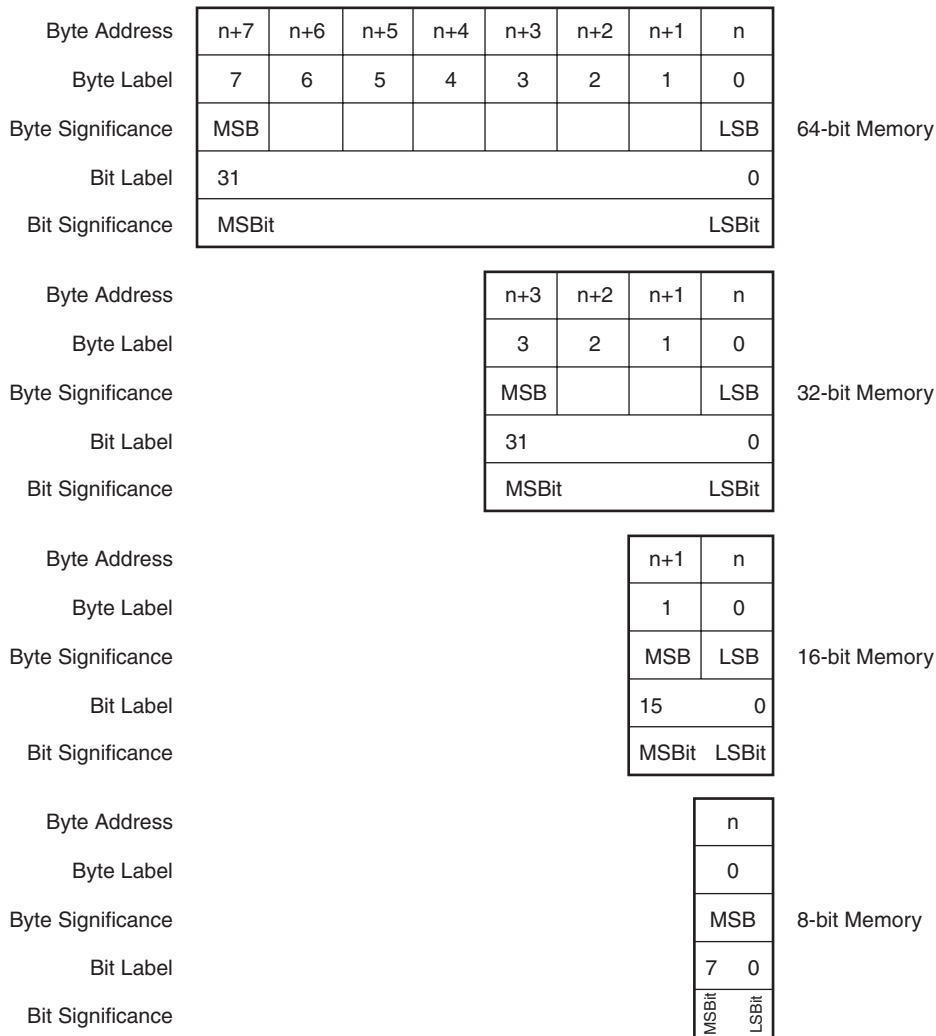
Table 38: Signals and Parameters per PHY Layer

PHY Layer	Signals	Parameters
MIG-Based Virtex-5 DDR2 PHY	MPMC_Clk0_DIV2 MPMC_Clk_200MHz MPMC_Idelayctrl_Rdy_I (optional) MPMC_Idelayctrl_Rdy_O (optional)	C_NUM_IDELAYCTRL C_IDELAYCTRL_LOC
MIG-Based Virtex-5 DDR PHY MIG-Based Virtex-4 DDR2 PHY MIG-Based Virtex-4 DDR PHY	MPMC_Clk_200MHz MPMC_Idelayctrl_Rdy_I (optional) MPMC_Idelayctrl_Rdy_O (optional)	C_NUM_IDELAYCTRL C_IDELAYCTRL_LOC
MIG-Based Spartan-3 DDR PHY	DDR_DQS_Div_I DDR_DQS_Div_O	N/A
MIG-Based Spartan-3 DDR2 PHY	DDR2_DQS_Div_I DDR2_DQS_Div_O	N/A
SDRAM PHY	MPMC_Clk_Mem MPMC_DCM_PSEN MPMC_DCM_PSINCDEC MPMC_DCM_PSDONE	C_STATIC_PHY_RDDATA_CLK_SEL C_STATIC_PHY_RDENDELAY
Static PHY	MPMC_Clk_Mem MPMC_DCM_PSEN MPMC_DCM_PSINCDEC MPMC_DCM_PSDONE	C_STATIC_PHY_RDDATA_CLK_SEL C_STATIC_PHY_RDDATA_SWAP_RISE C_STATIC_PHY_RDENDELAY

The memory data and address signals are marked with little-endian labeling.

The following figures (Figure 8 and Figure 9) show the little-endian and big-endian formats. Table 39, page 79 describes the little-endian bit and byte labeling for the data and control signals in the external memory.

Note: Use caution with the connections to the external memory devices to avoid incorrect data and address connections. The bit ordering used in the MPMC memory interface is reversed from the bit ordering used in the memory controllers named `plb_ddr`, `plb_ddr2`, `opb_ddr`, `mch_opb_ddr`, and `mch_opb_ddr2`. The BSB can be used to create MPMC designs to illustrate correct memory interface connections.



DS43_56_072607

Figure 8: Little-Endian Memory Data Types

Little-Endian Label Settings

Table 39: Little-Endian Bit and Byte Label Settings

Description	Memory Type	MPMC Signal [MSB:LSB]	Memory Signal [MSB:LSB]
Data Bus	All	<memory_type>_DQ [C_MEM_DATA_WIDTH-1:0]	DQ[C_MEM_DATA_WIDTH-1:0]
Bank Address	All	<memory_type>_BankAddr [C_MEM_BANKADDR_WIDTH-1:0]	BA[C_MEM_BANKADDR_WIDTH-1:0]
Address	All	<memory_type>_Addr [C_MEM_ADDR_WIDTH-1:0]	A[C_MEM_ADDR_WIDTH-1:0]
Data	All	<memory_type>_DQ [C_MEM_DATA_WIDTH-1:0]	DQ[C_MEM_DATA_WIDTH-1:0]
Data Strobe	DDR / DDR2	<memory_type>_DQS [C_MEM_DQS_WIDTH-1:0]	UDQS,LDQS (Replicate for number of memory parts)
Differential Data Strobe	DDR2	<memory_type>_DQS_n [C_MEM_DQS_WIDTH-1:0]	UDQS#,LDQS# (Replicate for number of memory parts)
Data Mask	All	<memory_type>_DM [C_MEM_DM_WIDTH-1:0]	UDM,LDM (Replicate for number of memory parts)
ECC Check Bits	All	<memory_type>_DQ [C_MEM_DATA_WIDTH +C_MPMC_CTRL_DATA_WIDTH -1:C_MEM_DATA_WIDTH]	DQ_ECC[C_MPMC_CTRL_DATA_WIDTH-1:0]
ECC Data Strobe	DDR / DDR2	<memory_type>_DQS [C_MEM_DQS_WIDTH +C_MPMC_CTRL_DQS_WIDTH- 1:C_MEM_DQS_WIDTH]	DQS_ECC [C_MPMC_CTRL_DQS_WIDTH-1:0]
Differential ECC Data Strobe	DDR2	<memory_type>_DQS_n [C_MEM_DQS_WIDTH +C_MPMC_CTRL_DQS_WIDTH -1:C_MEM_DQS_WIDTH]	DQSn_ECC [C_MPMC_CTRL_DQS_WIDTH-1:0]
ECC Data Mask	All	<memory_type>_DM [C_MEM_DM_WIDTH +C_MPMC_CTRL_DM_WIDTH -1:C_MEM_DM_WIDTH]	DM_ECC [C_MPMC_CTRL_DM_WIDTH-1:0]

Big-Endian Memory Data Types

Byte address	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7	Double Word
Byte label	0	1	2	3	4	5	6	7	
Byte significance	MS Byte						LS Byte		
Bit label	0							63	
Bit significance	MS Bit						LS Bit		

Byte address	n	n+1	n+2	n+3	Word
Byte label	0	1	2	3	
Byte significance	MS Byte		LS Byte		
Bit label	0			31	
Bit significance	MS Bit		LS Bit		

Byte address	n	n+1	Halfword
Byte label	0	1	
Byte significance	MS Byte	LS Byte	
Bit label	0	15	
Bit significance	MS Bit	LS Bit	

Byte address	n	Byte
Byte label	0	
Byte significance	MS Byte	
Bit label	0 7	
Bit significance	MS Bit LS Bit	

DS643_64_103007

Figure 9: **Big-Endian Memory Data Types**

Connecting Memory to a DDR2 MPMC Design Example

The following figure illustrates an example of connecting memory to a DDR2 MPMC design.

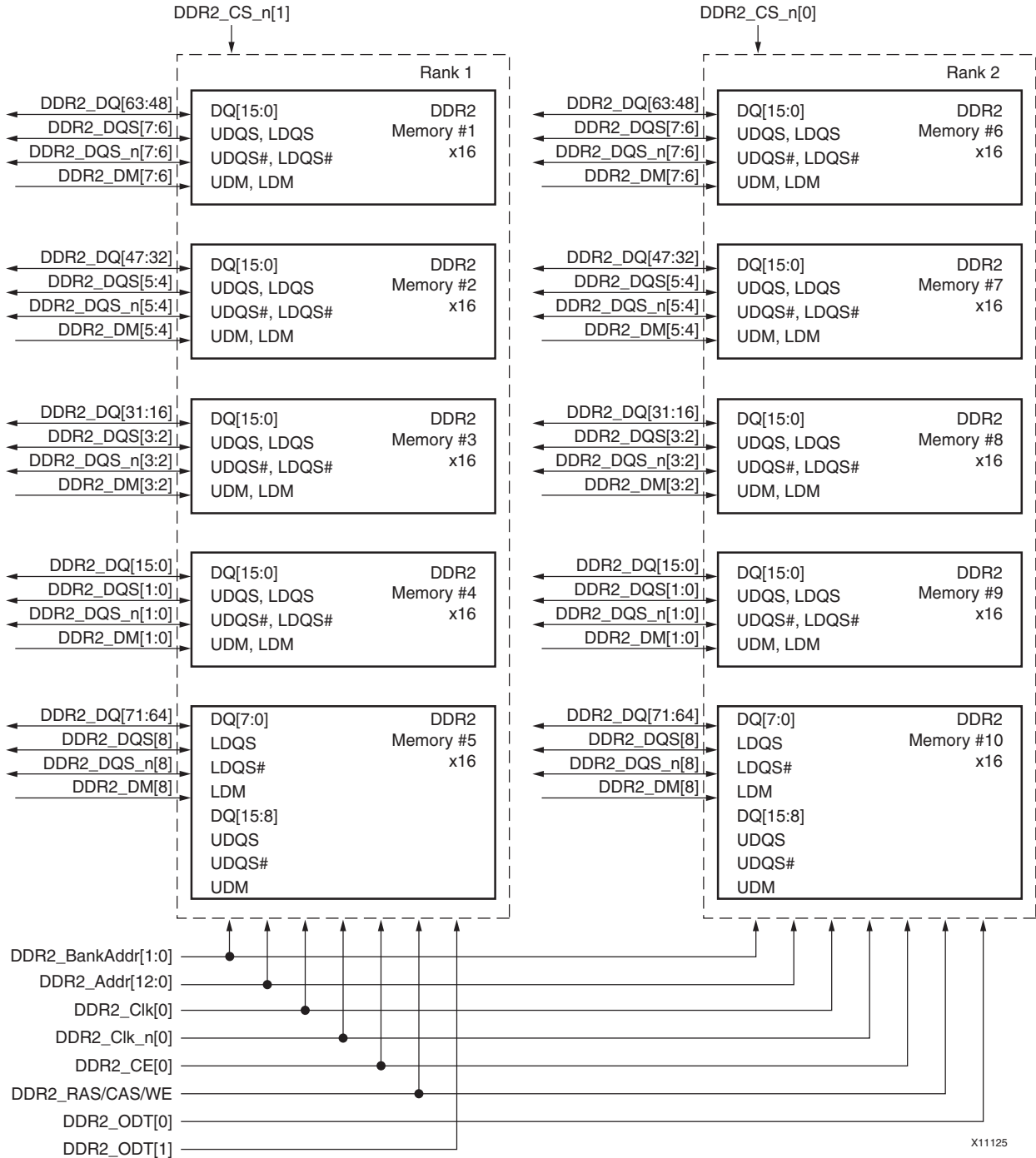


Figure 10: Example DDR2 Memory Connections

The example in [Figure 10, page 81](#) has the following specified parameters:

```
C_MEM_TYPE = "DDR2"  
C_INCLUDE_ECC_SUPPORT = 1  
C_MEM_CLK_WIDTH = 1  
C_MEM_ODT_WIDTH = 2  
C_MEM_CE_WIDTH = 1  
C_MEM_CS_N_WIDTH = 2  
C_MEM_ADDR_WIDTH = 13  
C_MEM_BANKADDR_WIDTH = 2  
C_MEM_DATA_WIDTH = 64  
C_MEM_NUM_RANKS = 2  
C_MEM_NUM_DIMMS = 1
```

In this example:

- The DDR2_CS_n[0] signal is connected to one rank of memory and DDR2_CS_n[1] is connected to the other rank of memory. The DDR2_ODT[0] signal is connected to one rank of memory and DDR2_ODT[1] is connected to the other rank of memory.
- Within a particular rank, the signals DDR2_CS_n and DDR2_ODT are connected to each memory in the rank. Some memory manufactures have specific recommendations for how these signals should be connected. For example, when using a dual rank DIMM, the recommendation might be to assert DDR2_ODT on the same rank that DDR2_CS_n is asserted (as shown in [Figure 10, page 81](#)). Other recommendations might be to assert DDR2_ODT on to opposite rank that the DDR2_CS_n is asserted, which can be achieved by swapping DDR2_ODT[0] and DDR2_ODT[1].
- All clock, addresses, and other control signals are also connected to each memory.
- The DDR2_DQ, DDR2_DQS, DDR2_DQS_n, and DDR2_DM signals do not go to all discrete memory components. Instead, 16 bits of DDR2_DQ go to one memory in each rank and 2 bits of DDR2_DQS, DDR2_DQS_n, and DDR2_DM go to one memory in each rank. The only exception is the ECC memory. When C_MEM_DATA_WIDTH is set to 64, there are 8 ECC bits. Because the memory has 16 bits; 8 of the DQ bits, the UDQS, the UDQS numbers, and the UDM are left unconnected. All other pins are connected on the memory.

The parameters C_MEM_CLK_WIDTH, C_MEM_CE_WIDTH, C_MEM_CS_N_WIDTH, and C_MEM_ODT_WIDTH can be used to replicate the CLK, CKE, CSn, and ODT outputs of MPMC to match the board schematics. Care must be taken with C_MEM_CS_N_WIDTH and C_MEM_ODT_WIDTH as these parameters must be an integer multiple of C_NUM_RANKS*C_NUM_DIMMS.

Clock and Reset Logic

MPMC has four system clock inputs and one clock for each PIM. Depending on the MPMC configuration, not all system clocks must be connected.

Note: For behavioral simulation, the `MPMC_Clk` and all `<PIM>_Clks` must be completely phase-aligned. This requirement is not as strict for actual implementation as any clock skew will be correctly analyzed by static timing analysis tools.

The basic MPMC core and each of the MPMC PIMs have a reset input. Internally these resets are OR'ed together to create the master reset for the entire MPMC (including PIMs).

Note: It is not possible to reset an individual PIM or PORT of the MPMC without resetting everything.

The master reset is internally registered and synchronized before being distributed throughout MPMC; therefore, the MPMC reset is a fully synchronous reset. Reset should be held for a minimum of eight cycles of the slowest PIM clock. After reset, there should not be access to any of the ports or control interfaces for 20 cycles of the `MPMC_Clk`. The following table provides a summary of the clocks and resets.

Table 40: Clock and Reset Summary

Clock/Reset Name	Description
<code>MPMC_Clk0</code>	Main MPMC clock; used to generate memory clock.
<code>MPMC_Clk0_DIV2</code>	<code>MPMC_Clk0</code> , divided by 2; used in MIG-based Virtex-5 DDR2 PHY only.
<code>MPMC_Clk90</code>	Main MPMC clock, phase shifted by 90 degrees. DDR/DDR2 only.
<code>MPMC_Clk_200MHz</code>	200 MHz clock; used in Virtex-4 and Virtex-5 architectures for IDELAY Control logic only. Only valid when using MIG-based Virtex-4 or Virtex-5 DDR/DDR2 PHY.
<code>MPMC_Clk_Mem</code>	Main MPMC clock, phase-shifted by n degrees. Used with Static PHY Interface only. See "Static PHY Interface," page 70 for more details.
<code><PIM>_Clk</code>	See specific PIM documentation for more details. Generally, these clocks will be the same as <code>MPMC_Clk</code> , or the <code>MPMC_Clk</code> synchronously divided by 2. The XCL, PLB, and SDMA PIMs support only 1:1 or 1:2 synchronous clock ratios. The XCL and PLB PIMs will detect automatically which clock ratio is being used. The SDMA requires that the clock ratio be specified by the <code>C_SDMA<Port_Num>_PI2LL_CLK_RATIO</code> parameter.
<code>MPMC_Rst</code>	Main MPMC reset.
<code><PIM>_Rst</code>	See specific PIM documentation for more details.

See the "System I/O Signals," page 15 for more information on clocks and reset signals.

Transaction Ordering and Memory Coherency

In the MPMC architecture, transactions are executed to memory in the order that the transactions are acknowledged with respect to a single port; consequently, on a single port, transactions are completed in the same order as requested.

Across multiple ports of MPMC there is no guarantee that the transactions issued by different ports will complete in the request order. You can modify the arbitration algorithms so that a given port is favored over another port. This can be used as a mechanism to influence transaction ordering but might not guarantee a specific order.

MPMC allows write transactions to be buffered inside the MPMC. Because of the buffering, there is an undefined time between when a write transaction has completed over NPI and when the write completes to memory.

Because transaction ordering is not guaranteed across ports, a port doing a read from an address location being written to by another port might read the new or the old memory value. In some applications it is important to know that a write has completed to memory before issuing a read of that location.

There are three methods that can ensure coherency:

1. The NPI interface can monitor the Write FIFO empty flag:
 - The empty flag is asserted when the write has completed to memory.
 - The design can wait for the empty flag to go high before signaling that a read can be performed.
2. The design can take advantage of the fact that transactions complete in order on a given port:
 - After a write to a sensitive part of memory, the device can issue a dummy read and wait for the dummy read to complete and return data.
 - The completion of the dummy read ensures that the previous write has completed to memory.
3. The arbitration algorithm can be adjusted:
 - If the port performing the writes can always be set to have higher priority than the ports doing the reads, this should also ensure that the write completes before the read across the two ports. Care should be taken with this method if there is a possibility that the PIM can have “bubble” cycles between the write and the read request.

Note: Using any of these methods to ensure coherency could result in reduced system performance; employ these methods when necessary only.

The DPLB and PLB PIMs follow the PLB CoreConnect method for handling memory coherency (for example, between PowerPC 405 processor instruction and data PLB interface). The PLB `BUSY` signal is asserted on writes until the `Write_FIFO_Empty` flag is asserted indicating the write transaction has completed to memory. The processor normally ignores the `BUSY` flag unless an instruction is executed such as `Sync` or `Enforce Instruction Execution In Order (EIEIO)`. When the `Sync` or `EIEIO` instruction is executed, the processor waits for PLB `BUSY` to be de-asserted before issuing another transaction.

Error Correction Code

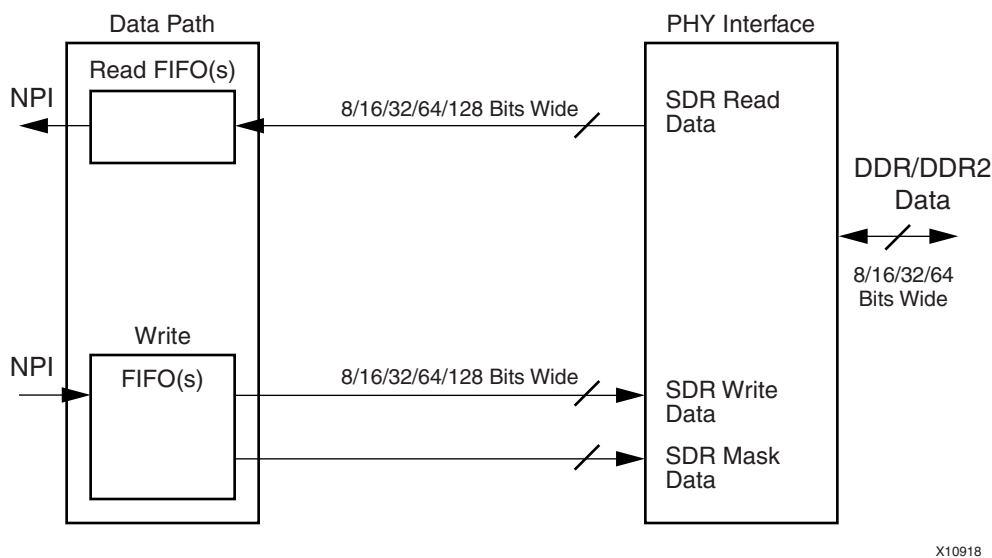
The Error Correction Code (ECC) is optionally enabled via the `C_INCLUDE_ECC_SUPPORT` parameter control.

The ECC features are:

- Supports 8-, 16-, 32-, and 64-bit wide SDRAM, DDR, and DDR2 memories.
- Provides Single Error Correction (SEC) and Double Error Detect (DED).
- Can generate interrupts when the number and type of errors reach a programmed threshold value.
- Can generate interrupts when the number and type of errors reach a programmed threshold value.
- Supported on Spartan-3 MIG PHY for 8-, 16-, and 32-bit wide data.
- Supported with Static PHY.

Implementation

ECC functionality is implemented by inserting the ECC decode and encode logic between the Physical Interface (PHY) and Data Path of the MPMC. The following figure shows the current MPMC PHY Data Path connection for DDR/DDR2 memory.



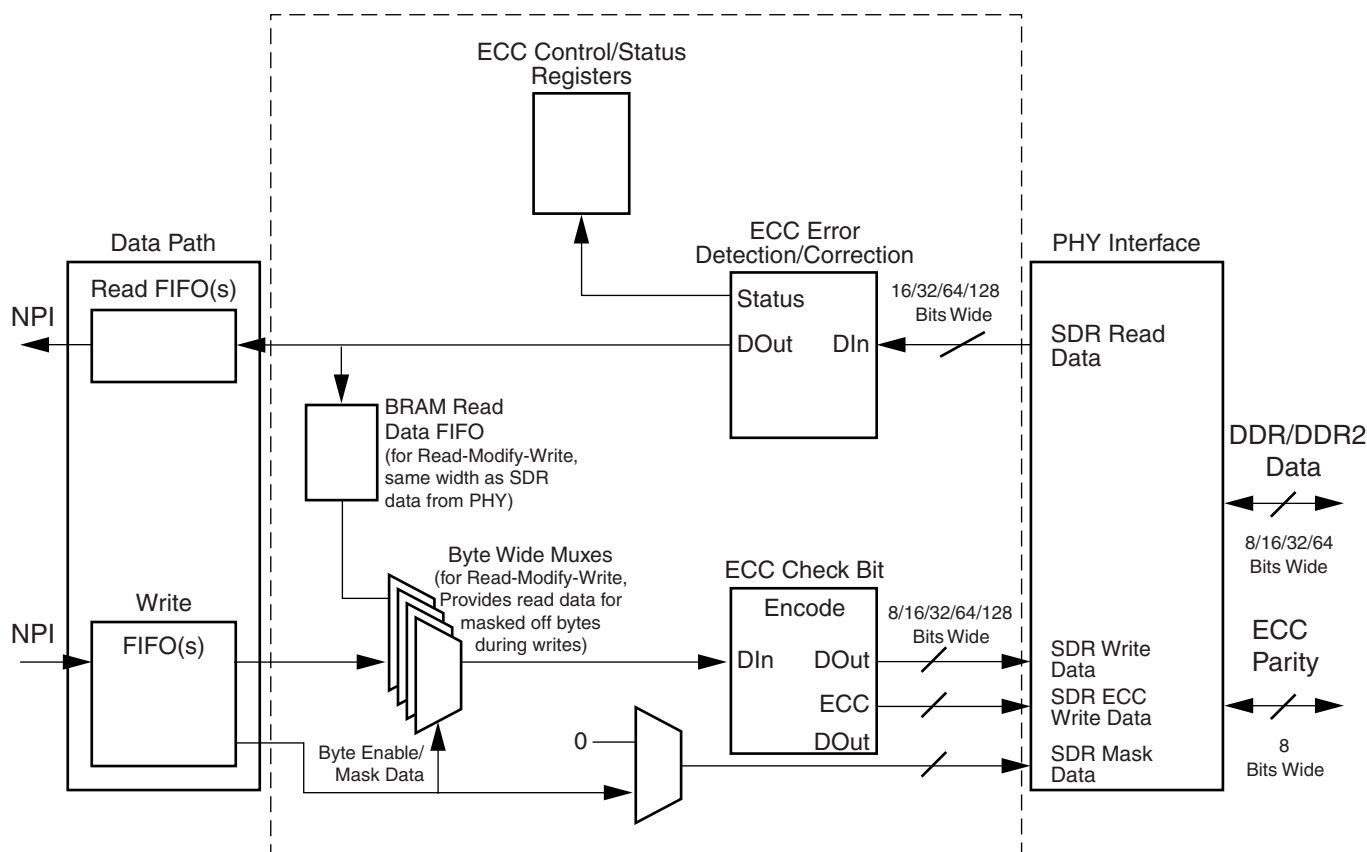
X10918

Figure 11: MPMC PHY-Data Path Connection

The PHY interface converts the Double Data Rate (DDR) data from the memory into a Single Data Rate (SDR) bus that is twice as wide as the memory width. For SDRAM, data stays the same width through the PHY.

Because MPMC supports 8-, 16-, 32-, and 64-bit SDRAM and DDR/DDR2 memory, this results in an SDR bus that is 8-, 16-, 32-, 64-, or 128-bit wide going to the data path module. The data path module implements the per-port FIFOs used by MPMC.

Figure 12, page 86 illustrates the changes that are made when ECC is enabled with DDR/DDR2 memory.



X10919

Figure 12: Data Path with ECC Enabled

When ECC functionality is enabled, several blocks are turned on to implement control and status registers, ECC decode and encode, and support for Read-Modify-Write (RMW) operations (needed for handling byte enables).

Note: All writes are RMW in the MPMC core once ECC is enabled.

The blocks are inserted between the PHY and data path so only one instance of the ECC logic is needed for multiple port configurations. The ECC decode and encode is performed in the SDR domain.

The ECC Control and Status registers module controls when interrupts are generated and provides control and status data access with respect to ECC. Refer to [Table 24, page 30](#) for more information on ECC Control and Status registers.

Read Data Handling

On Reads, data passes through the ECC Error Detection and Correction block. The data is checked for errors. If there are no errors, data passes through as normal and goes into the Read FIFO as a normal read. If a single error is detected, the error is corrected automatically (SEC) and sent to the Read FIFO to complete as a normal read. If two errors are detected (DED), the data is not corrected but passed through unchanged and the problem is reported to the Control and Status registers.

Need for Read Modify Write

During writes to memory, a new set of ECC check bits must be written to memory along with write data. The ECC encoding process is handled by the ECC Check Bit Encode block. Read-Modify-Writes (RMW) operations are needed for write transactions where the byte enables that span the width of the ECC word on the SDR bus are not all On or Off. Because many ECC DIMMs do not have Data Mask (DM) pins, the RMW is required.

When byte enables do not exist or are not all On or all Off, the correct value of the ECC check bits are not known because not all the data across the ECC word is present.

The RMW operation first fetches the data value of the masked off byte lanes so the correct ECC check bits for the whole ECC word can be computed and written. RMW is accomplished by taking read data (after ECC decode and correction) and storing the data in a FIFO that is the same width as the SDR data bus. Later, as write data comes out of the Write FIFO, the masked off byte enables activate an MUX that routes read data to fill in the corresponding “holes” in the write data. The result is a complete set of data with which to compute the new ECC check bits. All resultant data is written to memory. Consequently, writes with any byte enables turned Off result in writes with all byte enables turned On. This has the side effect that masked-off write data is read, scrubbed, and written back.

The Control state machine is modified during a RMW operation to perform a full Read and then a full Write. Because the data is on the same memory page, an optimization is to skip the precharge and activate commands between a read and a write. To optimize the space in the control path block RAM state machines, RMW operations will reuse the corresponding Read and Write state machines for the given transfer size. Special block RAM state machine bits are used to implement the chained RMW operation and to skip precharge and activate commands.

The RMW process does add latency to the design because a full Read must precede a Write. This can double the transaction time for Writes. Writes with all byte enables On or all Off are faster than Writes with mixed byte enables due to need for RMW. A flag bit is introduced into the NPI interface to allow the transactions to be qualified as needing or not needing RMW.

The `C_PIM<Port_Num>_RdModWr` flag bit informs the MPMC if transactions require RMW (= 1) or do not require RMW (= 0). The `C_PIM<Port_Num>_RdModWr` signal must be asserted with `PIM<Port_Num>_AddrReq`. The `C_PIM<Port_Num>_RdModWr` signal should be asserted with respect to the memory burst length of 4 and the ECC word size of the memory interface being used. Any NPI transfer sizes which are less than a 4 beat memory burst must assert `C_PIM<Port_Num>_RdModWr`.

For NPI interfaces that do not support the RMW, or are not able to set it dynamically, the value should default to 1 to ensure correct Write operations under all conditions. Dynamic RMW allows for write performance to be optimized when RMW is known to not be needed.

If all the byte enables are Off but a RMW operation is performed, data is still read, scrubbed, and written back. This would permit a custom NPI device to perform burst writes with byte enables all Off (and `RdModWr = 1`) to scrub memory. MPMC does not require the memory to support DM pins with ECC. DM pins can be connected, if needed, or left unconnected.

ECC encode functionality adds latency to Writes. RMW support adds one cycle of latency to multiplex in Read data (in addition to latency of the full Read transaction). The Control state machines support the additional cycles of latency between Write FIFO pop and data appearing at the PHY interface.

For debugging and testing, the ECC encode process allows you to insert single or double bit errors into the data being written for test purposes. The error insertion logic can be parameterized out.

Memory Organization and ECC Word Size

The following table describes the ECC word size and number of extra memory data bits needed for different organizations of memory. The `RdModWr` flag must be set according to the alignment across the ECC word size over a memory burst length of 4. The PHY requires a full ECC byte lane to be present on the board even though fewer bits might be used by ECC algorithm due to the requirements of the data calibration algorithms used in the PHY.

Note: The PHY data calibration algorithm requires that a full ECC byte lane be present on the board although fewer bits might be used by the ECC algorithm.

Table 41: ECC Word Size

Memory Type	Memory Data Width	Physical Memory ECC Data Width (C_NUM_ECC_BITS)	ECC Word Size
DDR/DDR2	8	8	2 Instances of 8 + 5 Check Bits
DDR/DDR2	16	8	2 Instances of 16 + 6 Check Bits
DDR/DDR2	32	8	2 Instances of 32 + 7 Check Bits
DDR/DDR2	64	8	2 Instances of 64 + 8 Check Bits
SDRAM	8	5	8 + 5 Check Bits
SDRAM	16	6	16 + 6 Check Bits
SDRAM	32	7	32 + 7 Check Bits
SDRAM	64	8	64 + 8 Check Bits

ECC Registers

Table 24, page 30 summarizes the ECC related registers which are included with the ECC logic for the MPMC when ECC is enabled (`C_INCLUDE_ECC_SUPPORT = 1`). The following subsections describe each register in greater detail and provide the bit definitions.

ECC Control Register

The ECC Control Register (ECCC) determines if ECC check bits are generated during memory write operation and checked during a memory read operation. The ECCC also defines testing modes if enabled by the parameter `C_INCLUDE_ECC_TEST`.

The following table describes the bit values for the ECCC register.

Table 42: ECCC Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:26				Reserved
27	FORCE_PE	R/W	0	Force Parity Field Bit Error ⁽¹⁾ : Available for testing and determines if parity field bit errors are forced in the data stored in the memory. See ECC Testing for more information. 0 = No parity field bit errors are created 1 = Parity field bit errors are forced in stored data

Table 42: ECCC Register Bit Definitions (Cont'd)

Bit(s)	Name	Core Access	Reset Value	Description
28	FORCE_DE	R/W	0	Force Double-bit Error ⁽¹⁾ : Available for testing and determines if double-bit errors are forced in the data stored in the memory. 0 = No double-bit errors are created 1 = Double-bit errors are forced in the stored data
29	FORCE_SE	R/W	0	Force Single-bit Error ⁽¹⁾ : Available for testing and determines if single-bit errors are forced in the data stored in the memory. 0 = No single-bit errors are created 1 = Single-bit errors are forced in the stored data
30	RE	R/W	1 ⁽²⁾	ECC Read Enable: 0 = ECC read logic is bypassed 1 = ECC read logic is enabled
31	WE	R/W	1 ⁽²⁾	ECC Write Enable: 0 = ECC write logic is bypassed 1 = ECC write logic is enabled

1. This bit is available only if C_INCLUDE_ECC_TEST = 1
2. Reset value is determined by parameter C_ECC_DEFAULT_ON.
If C_ECC_DEFAULT_ON = 1 then this bit is equal to 1
If C_ECC_DEFAULT_ON = 0, then this bit is equal to 0

ECC Status Register

The ECC Status register (ECCS) in combination with the ECC Error Address register (ECCADDR) records the first occurrence of an error and latches information about the error until the error is cleared by writing to the ECCS.

The following table describes the bit values for the ECC Status register.

Table 43: ECCS Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:15				Reserved
16:19	ECC_ERR_SIZE	R/ROW ⁽¹⁾	0000	ECC Error Transaction Size. Records the size of the NPI transaction where the error occurred. For ECC_ERR_SIZE field values see Table 18, page 21 under the signal name PIM<Port_Num>_Size.
20	ECC_ERR_RNW	R/ROW ⁽¹⁾	0	ECC ERROR Transaction Read/Write: Indicates if error occurred on a read transactions or a write transaction that employed a read-modify-write operation. 0 = Write Error 1 = Read Error
21:28	ECC_ERR_SYND	R/ROW ⁽¹⁾	00000000	ECC Error Syndrome: Indicates the ECC syndrome value of the most recent memory transaction in which a single-bit error was detected. The 8-bit syndrome value indicates the data bit position in which an error was detected and corrected.

Table 43: ECCS Register Bit Definitions (Cont'd)

Bit(s)	Name	Core Access	Reset Value	Description
29	PE	R/ROW ⁽¹⁾	0	Parity Field Bit Error: During a memory transaction an error was detected in a parity field bit. 0 = No parity field bit errors detected. 1 = Parity field bit error detected and corrected.
30	DE	R/ROW ⁽¹⁾	0	Double-Bit Error: During a memory transaction a double-bit error was detected and is not correctable. 0 = No double-bit errors were detected. 1 = Double-bit error was detected.
31	SE	R/ROW ⁽¹⁾	0	Single-Bit Error: During memory transaction a single-bit error was detected and corrected. 0 = No single-bit errors were detected. 1 = Single-bit error detected and corrected.

1. ROW = Reset On Write. Any write operation to the ECCSR will reset the register.

ECC Single-Bit Error Count Register

The ECC Single-Bit Error Count register (ECCSEC) records the number of ECC single-bit errors that occurred during the memory transaction on the data bits only. When using the force error feature, the number of errors detected may not be as expected because the force error feature counts the number of memory data beats that have errors, not the number of NPI data beats that have errors. Because the Read_Modify_Write might read more data than NPI requested, the count can be misleading. The ECC logic will correct the detected single-bit errors. When the value in this register reaches 4095 (the max count), the next single-bit error detected is not counted. This count consumes 12-bits.

Note: Single bit errors occurring on ECC check bits are covered by the ECCPEC register.

The following table describes the bit values for the ECC Single-Bit Error Count register.

Table 44: ECCSEC Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:19				Reserved
20:31	SEC	R/ROW ⁽¹⁾	0	Single-Bit Error Count. Indicates the number of single-bit errors that occurred during the pervious memory transactions. The maximum error count is 4095.

1. ROW = Reset On Write. Any write operation to the ECCSEC register will reset the register.

ECC Double-Bit Error Count Register

The ECC Double-Bit Error Count Register (ECCDEC) records the number of ECC double-bit errors that occurred during the memory transaction. ECC cannot correct double-bit errors detected. When the value in this register reaches 4095 (the max count), the next double-bit error detected is not counted. When using the force error feature, the number of errors detected may not be as expected because the force error feature counts the number of memory data beats that have errors, not the number of NPI data beats that have errors. Because the Read_Modify_Write might read more data than NPI requested, the count can be misleading. The ECC logic will correct the detected single-bit errors. When the value in this register reaches 4095 (the max count), the next single-bit error detected is not counted.

The following table describes the bit values for the ECCDEC.

Table 45: ECCDEC Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:19				Reserved
20:31	DEC	R/ROW ⁽¹⁾	0	Double-Bit Error Count: Indicates the number of double-bit errors that occurred during the pervious memory transactions. The maximum error count is 4095.

1. ROW = Reset On Write. Any write operation to the ECCDEC register will reset the register.

ECC Parity Field Bit Error Count Register

The ECC Parity Field Bit Error Count Register (ECCPEC) records the number of bit errors that occurred in the ECC parity field during the memory transaction. ECC logic will correct detected parity field bit errors. When the value in this register reaches 4095 (the maximum count), the next parity field bit error detected is not counted. When using the force error feature, the number of errors detected may not be as expected because the force error feature counts the number of memory data beats that have errors, not the number of NPI data beats that have errors. Because the Read_Modify_Write might read more data than NPI requested, the count can be misleading. The ECC logic will correct the detected single-bit errors. When the value in this register reaches 4095 (the max count), the next single-bit error detected is not counted.

The following table describes the bit values for the ECCPEC.

Table 46: ECCPEC Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:19				Reserved
20:31	PEC	R/ROW ⁽¹⁾	0	Parity Field Bit Error Count: Indicates the number of errors that occurred in the parity field bits during the last memory transactions. The maximum error count is 4095.

1. ROW = Reset On Write. Any write operation to the ECCPEC register will reset the register.

ECC Error Address Register

The following table describes the bit values for the ECC Error Address (ECCADDR) register.

Table 47: ECCADDR Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:31	ECCERRADDR	RO	N/A	ECC Error Address: Indicates the physical ECC address corresponding to an ECC error reported by the ECC Status register (ECCS). This register value is only valid when an error is actively reported in the ECCS.

Device Global Interrupt Enable Register

The Device Global Interrupt Enable Register (DGIE) is used to globally enable the final interrupt output from the ECC interrupt service. The following table describes the bit values for the DGIE.

Table 48: DGIE Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0	GIE	R/W	0	Global Interrupt Enable: 0 = Interrupts disabled. 1 = Interrupts enabled.
1-31				Reserved

IP Interrupt Status Register

The IP Interrupt Status register (IPIS) is the interrupt capture register for the ECC logic. The following table describes the bit values for the IPIS.

Table 49: IPIS Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:28				Reserved
29	PE_IS	R/TOW ⁽¹⁾	0	Parity Field Bit Error Interrupt Status: Indicates a parity field bit error has occurred during the memory data transaction. In the ECC module, parity field bit errors will be corrected as data is read from memory. This interrupt is for system monitoring only and does not indicate corrupt data. 0 = Parity field bit error count is less than C_ECC_PEC_THRESHOLD. 1 = Parity field bit error count is more than C_ECC_PEC_THRESHOLD.
30	DE_IS	R/TOW ⁽¹⁾	0	Double-Bit Error Interrupt Status: Indicates a double-bit data error has occurred during the memory transaction. In the ECC module, double-bit errors can be detected, but not corrected. When this interrupt is asserted, the data read from memory is not valid. 0 = Double-bit error count is less than C_ECC_DEC_THRESHOLD. 1 = Double-bit error count is more than C_ECC_DEC_THRESHOLD.
31	SE_IS	R/TOW ⁽¹⁾	0	Single-Bit Error Interrupt Status: Indicates a single-bit error has been detected during the memory transaction. In the ECC module, single-bit errors will be detected and corrected. This interrupt is for system monitoring only and does not indicate corrupt data. 0 = Single-bit error count is less than C_ECC_SEC_THRESHOLD. 1 = Single-bit error count is more than C_ECC_SEC_THRESHOLD.

1. TOW is Toggle On Write.

Writing a 1 to a bit position within the register causes the corresponding bit position in the register to toggle.

IP Interrupt Enable Register

The IP Interrupt Enable register (IPIE) has an enable bit for each defined bit of the IP Interrupt Status register. The following table describes the bit values for the ECC IPIE.

Table 50: IPIE Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0:28				Reserved
29	PE_IE	R/W	0	Parity Field Bit Error Interrupt Enable: Enables assertion of the interrupt for indicating parity field bit errors have occurred. 0 = Disabled 1 = Enabled
30	DE_IE	R/W	0	Double-bit Error Interrupt Enable: Enables assertion of the interrupt for indicating double-bit data errors have occurred. 0 = Disabled 1 = Enabled
31	SE_IE	R/W	0	Single-bit Error Interrupt Enable: Enables assertion of the interrupt for indicating single-bit data errors have occurred. 0 = Disabled 1 = Enabled

ECC Testing

To enable testing on the ECC core logic, set `C_INCLUDE_ECC_TEST = 1`. The ECC Control register (ECCCR), described in "ECC Control Register," page 88, includes control bits for enabling or disabling the test logic.

The following list describes possible forcing errors combinations. If any other combination is attempted, no errors will be forced on the data written to memory.

- No bit error forcing.
- Force single-bit data errors (ECC Control register (ECCCR): `FORCE_SE = 1`).
- Force double-bit data errors (ECC Control register (ECCCR): `FORCE_DE = 1`).
- Force parity bit errors (ECC Control register (ECCCR): `FORCE_PE = 1`).
- Force single-bit data and single parity field bit errors (ECC Control register (ECCCR): `FORCE_SE = 1` and `FORCE_PE = 1`).

For single-bit error testing, a mask shift register forces single-bit errors on the data written to memory. The data mask shift register has the least significant single-bit equal to one. When testing is enabled during a memory write, the shift register clocks the one toward the Most Significant Bit (MSB). You must be careful when trying to predict the pattern. More memory data beats may be written than expected because the Read_Modify_Write will read a full "memory cacheline," merge this with the data sent from the PIM, then write back the entire "memory cacheline." This will result in a different rotation than you might expect.

For double-bit error testing, a data mask shift register forces double-bit errors on the data written to memory. The data mask shift register has two adjacent bits equal to one (starting in the two least significant bit positions) and rotates the "11" pattern in the shift register (towards the two most significant bits) on each memory write.

When parity field bit error testing is enabled, a mask shift register forces single-bit errors on the check bits stored in memory. The parity mask shift register has the least significant single-bit equal to one and rotates the one (toward the MSB) on each memory write.

Performance Monitoring

The MPMC Performance Monitor (PM) is an optional per-port feature that can be used to count transaction lengths across each NPI interface. The PM provides the ability to instrument and measure the performance of the MPMC by collecting transactional statistics at the NPI level relative to the memory clock rate. Each PM is capable of capturing the duration, size, read or write, and dead cycle counts of each transaction. These are stored into a block RAM and are retrievable over the MPMC_CTRL Slave PLB v.46 interface. To use PMs on a specific port, that port must have the Address Acknowledge Pipeline enabled to ensure correct operation. PM cycle counts only represent transactions across the NPI interface and do not include latency of attached PIMs or user NPI logic.

Performance Monitoring Features

The PM provides:

- An optional configurable length global cycle counter.
- An optional configurable length arbitration dead cycle counter for each port.
- The MPMC_CTRL Slave PLB v.46 interface to control and collect data from the Performance Monitors.
- A performance monitor state machine and data collector consuming one block RAM per port enabled.
- PM registers.

This section contains the following subsections:

- [Performance Monitor Operation](#)
- [Performance Monitor Measurement Methodology](#)
- [Performance Monitoring Usage Example](#)
- [Performance Monitor Registers](#)

Performance Monitor Operation

The following subsections provide an outline of the PM measurement methodology and give an example of the steps in a PM monitoring session.

Performance Monitor Measurement Methodology

The cycle counts recorded in each data bin is recorded as following:

- Data Bin Read Transactions: Number of memory clock cycles from NPI_<PortNum>AddrReq until last data is popped from data FIFO.
- Data Bin Write Transactions: Number of memory clock cycles from NPI_<PortNum>AddrReq until the later of NPI_<PortNum>AddrAck or the last push into the data FIFO.
- Global Cycle Counter: Counts each memory clock cycle after at least one PM is enabled.
- Dead Cycle Counter: Counts the number of memory cycles from NPI_<PortNum>AddrReq until granted by NPI_<PortNum>AddrAck.

Performance Monitoring Usage Example

An example PM session includes the following steps:

1. Disable any currently enabled PMs by writing zeros to the Performance Monitor Control Register.
2. Clear the PM counters by writing ones to the Performance Monitor Clear Register.
3. Wait for the clear operation to complete by polling the Performance Monitor Status Register for all PMs bits set as one.
4. Clear PM status by writing ones to the Performance Monitor Status Register.
5. Enable PM counting by writing ones to the Performance Monitor Control Register.
6. Execute user task to be monitored.
7. Stop the PM counting by writing ones to the Performance Monitor Control Register.
8. Read each data bin value, typically looping through each set of 32 data bins, read/write set, qualifier set, and finally each PM.
9. Read the Performance Monitor Global Cycle Count Register for total test time.
10. Read the Performance Monitor Dead Cycle Count Register of each PM for cycles lost to arbitration grant delays.
11. Prepare for next task monitoring by either re enabling the PMs or executing the clear process.

Performance Monitor Registers

PM registers detailed in the following subsections are available that perform these actions:

- Capture performance statistics (PMCTRL)
- Monitor the status of the clear issued to the PM data bins (PMSTATUS)
- Contain the current value of the global cycle counter (PMGCC)
- Contain the current value of the dead cycle counter for each port (PMx_DCC)
- Contain the transaction information for the performance monitors (PMx_DATA_BINx)

The PM registers are:

- [Performance Monitor Control Register](#)
- [Performance Monitor Clear Register](#)
- [Performance Monitor Status Register](#)
- [Performance Monitor Global Cycle Count Register](#)
- [Performance Monitor Dead Cycle Count Register](#)
- [Performance Monitor Data Bin Registers](#)

Performance Monitor Control Register

The Performance Monitor Control Register (PMCTRL) sets the enable bits for performance monitors to capture statistics and place them in the data bins. Initially, the performance monitors are not enabled and the data bins are initialized to 0.

The performance monitor enable bit should be set to 0 when reading back data to maintain consistent data across all the data bins. Setting the performance monitor enable bit to 1 will immediately start capturing data and also enable the dead cycle counters, if the parameter is enabled.

If the global cycle counter is enabled, it will count when any of the enable bits are high. The following table describes the PMCTRL register bits.

Table 51: PMCTRL Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0	PM0_ENABLE	R/W	0	1 = Enable PM0 0 = Disable PM0
1	PM1_ENABLE	R/W	0	1 = Enable PM1 0 = Disable PM1
2	PM2_ENABLE	R/W	0	1 = Enable PM2 0 = Disable PM2
3	PM3_ENABLE	R/W	0	1 = Enable PM3 0 = Disable PM3
4	PM4_ENABLE	R/W	0	1 = Enable PM4 0 = Disable PM4
5	PM5_ENABLE	R/W	0	1 = Enable PM5 0 = Disable PM5
6	PM6_ENABLE	R/W	0	1 = Enable PM6 0 = Disable PM6
7	PM7_ENABLE	R/W	0	1 = Enable PM7 0 = Disable PM7
8:14	Reserved			Reserved
15	PM_GC_CLR	W	X	1 = Clear Global Cycle Counter 0 = No Action
16	PM0_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM0 0 = No Action
17	PM1_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM1 0 = No Action
18	PM2_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM2 0 = No Action
19	PM3_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM3 0 = No Action
20	PM4_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM4 0 = No Action
21	PM5_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM5 0 = No Action
22	PM6_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM6 0 = No Action
23	PM7_DCC_CLR	W	X	1 = Clear Dead Cycle Counter PM7 0 = No Action
24-31	Reserved			Reserved

Performance Monitor Clear Register

The Performance Monitor Clear Register (PMCLR) facilitates clearing the block RAMs used for the PM data bins back to 0. It also used to clear the dead cycle counters and global cycle counter to 0. This register is a write only register. Writing a 1 to the clear bits will immediately clear the counters, and start the clearing of the data bins. Because the data bins are stored in a block RAM, it will take approximately 512 clock cycles to clear the data bins to 0. To monitor the status of the clear, please see the PMSTATUS register.

The following table describes the PMCLR register bits.

Table 52: PMCLR Register Bit Definitions

Bit(s)	Name	Core Access	Reset Value	Description
0	PM0_DATABIN_CLR	W	X	1 = Clear all data bin storage PM0
1	PM1_DATABIN_CLR	W	X	1 = Clear all data bin storage PM1
2	PM2_DATABIN_CLR	W	X	1 = Clear all data bin storage PM2
3	PM3_DATABIN_CLR	W	X	1 = Clear all data bin storage PM3
4	PM4_DATABIN_CLR	W	X	1 = Clear all data bin storage PM4
5	PM5_DATABIN_CLR	W	X	1 = Clear all data bin storage PM5
6	PM6_DATABIN_CLR	W	X	1 = Clear all data bin storage PM6
7	PM7_DATABIN_CLR	W	X	1 = Clear all data bin storage PM7
8:14	Reserved	NA	NA	Reserved
15	PM_GCC_CLR	W	X	1= Clear Dead Cycle Counter
16	PM0_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 0
17	PM1_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 1
18	PM2_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 2
19	PM3_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 3
20	PM4_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 4
21	PM5_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 5
22	PM6_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 6
23	PM7_DCC_CLR	W	X	1= Clear Dead Cycle Counter Port 7

Performance Monitor Status Register

The Performance Monitor Status Register (PMSTATUS) is used to monitor the status of the clear issued to the pm data bins. When all data bins in a PM have been cleared, the PM clear status bit is 1. This is used to ensure that the performance monitors have been successfully cleared before enabling them. The PM clear status bit will remain 1 until it is cleared by writing a 1 to toggle the bit.

The following table describes the PMSTATUS register bits.

Table 53: PMSTATUS Register Bit Definition

Bit(s)	Name	Core Access	Reset Value	Description
0	PM0_CLR_STATUS	R/TOW	X	1 = Data bin for PM0 is finished clearing 0 = Data bin for PM0 is not finished clearing, or clear has not been started
1	PM1_CLR_STATUS	R/TOW	X	1 = Data bin for PM1 is finished clearing 0 = Data bin for PM1 is not finished clearing, or clear has not been started
2	PM2_CLR_STATUS	R/TOW	X	1 = Data bin for PM2 is finished clearing 0 = Data bin for PM2 is not finished clearing, or clear has not been started
3	PM3_CLR_STATUS	R/TOW	X	1 = Data bin for PM3 is finished clearing 0 = Data bin for PM3 is not finished clearing, or clear has not been started
4	PM4_CLR_STATUS	R/TOW	X	1 = Data bin for PM4 is finished clearing 0 = Data bin for PM4 is not finished clearing, or clear has not been started
5	PM5_CLR_STATUS	R/TOW	X	1 = Data bin for PM5 is finished clearing 0 = Data bin for PM5 is not finished clearing, or clear has not been started
6	PM6_CLR_STATUS	R/TOW	X	1 = Data bin for PM6 is finished clearing 0 = Data bin for PM6 is not finished clearing, or clear has not been started
7	PM7_CLR_STATUS	R/TOW	X	1 = Data bin for PM7 is finished clearing 0 = Data bin for PM7 is not finished clearing, or clear has not been started
8:31	Reserved			Reserved

Performance Monitor Global Cycle Count Register

The Performance Monitor Global Cycle Count Register (PMGCC) is a read-only 64-bit register that contains the current value of the global cycle counter. This register is only available if the global cycle counter is enabled. Its width can be up to 64 bits, and is padded to the left with zeros, if the counter width is smaller than 64 bits. The PMGCC counts the total number of memory clock cycles that have elapsed since at least one PM has been enabled. The PMGCC is a useful absolute time base of a performance monitoring session.

Performance Monitor Dead Cycle Count Register

The Performance Monitor Dead Cycle Count Registers (PMx_DCC) is a read-only 64-bit register that contains the current value of the dead cycle counter for each port. This register is only available if the dead cycle counter has been enabled for that the port. Its width can be up to 64 bits, and is padded to the left with zeros, if the counter width is smaller than 64 bits.

Each dead cycle count represents a NPI clock cycle where a request was not immediately acknowledged by the arbiter. This is useful to measure how many cycles a particular port has waited since performance monitoring was first enabled.

Dead cycles typically occur due to the MPMC controller processing older or higher priority port accesses or the memory being unavailable during external memory maintenance.

Performance Monitor Data Bin Registers

The Performance Monitor Data Bin Registers (PMx_DATA_BINx) contain the transaction information for the performance monitors. Each performance monitor contains 512 36-bit data bins. These values are read-only 36-bit registers that are padded with zeros on the left to expand them to 64 bits. Therefore, each performance monitor data bin will be 8 kb of data.

Each bin contains a count of transactions for a specific PM port, the NPI transaction type, the transaction direction, and the exact number or range of clock cycles elapsed. For example, one data bin on PIM port 2 might contain the total quantity of 32-word burst-type write transactions which lasted between 8 and 15 clock cycles since the PM was first enabled.

Data Bin Organization

Each PM is divided into eight qualifiers, indicating the transaction size. These qualifiers are then subdivided into Writes and Reads. This allows 32 bins per qualifier each write and read. The bins themselves represent the transaction length of that particular qualifier. The 36-bit number the bin contains represents how many times that transaction length has been counted.

Qualifier Definitions

The following table describes the qualifier definitions.

Qualifier	Description
0	Byte – Double Words
1	Cache Line 4
2	Cache Line 8
3	Burst 16
4	Burst 32
5	Burst 64
6	Reserved
7	Reserved

Data Bin Organization

The following table describes the Data Bin organization.

Qualifier Type	Access Type	Bin No.	Offset from PMx_DATABINx
0	Write	0	0x000
		1	0x008
	
		31	0x0F8
	Read	0	0x100
		1	0x108
	
		31	0x1F8
1	Write	0	0x200
		1	0x208
	
		31	0x2F8
	Read ...	0	0x300
		1	0x308
	
		31	0x3F8
7	Write	0	0xE00
		1	0xE08
	
		31	0xEF8
	Read	0	0xF00
		1	0xF08
	
		31	0xFF8

The bin number represents the length of the transaction in cycles shifted by the `C_PM_SHIFT_BY` parameter. This allows a trade between the maximum length of a transaction that can be recorded and the granularity of the transaction lengths. The number the bin contains is the number of times that particular transaction length was recorded.

The following table shows the C_PM_SHIFT_BY relationship of Bin Number to Transaction Length.

C_PM_SHIFT_BY	Bin Number	Transaction Length (cycles)
0	0	0
	1	1

	31	31+
1	0	0-1
	1	2-3

	31	62-63+
2	0	0-3
	1	4-7

	31	124-127+
3	0	0-7
	1	8-15

	31	248-258+

Personality Interface Modules

The Personality Interface Module (PIM) architecture comprises the following interfaces:

- [Xilinx CacheLink PIM](#)
- [Soft Direct Memory Access Controller for LocalLink Interfaces](#)
- [Processor Local Bus Version 4.6 PIM](#)
- ["Native Port Interface PIM"](#)
- ["PowerPC 440 Memory Controller PIM"](#)
- ["Video Frame Buffer Controller PIM"](#)

The following subsections describe the PIMs. The PIM design parameters, I/O signals, and control and status register summaries are in ["Design Parameters," page 1](#), ["MPMC I/O Signals," page 15](#), and ["Control and Status Registers," page 29](#), respectively.

Note: Throughout the document, the size of a word is 32-bits.

Xilinx CacheLink PIM

The Xilinx CacheLink (XCL) PIM allows connection from an XCL bus interface to MPMC. The XCL PIM is described in the following subsections:

- [XCL Features](#)
- [XCL Overview](#)
- [Connecting XCL to a MicroBlaze Soft Processor](#)
- [XCL Configuration Options](#)
- [XCL Line Size and Write Transfers](#)
- [XCL Pipeline Stages](#)
- [XCL Clock Requirements](#)
- [XCL Additional Information](#)

XCL Features

The XCL PIM supports the following features:

- 1-, 4-, 8- and 16-word reads and writes.
- Auto-detection (at reset) for clock ratios of 2:1 and 1:1 of NPI (the MPMC Memory Clock) to XCL.
- Read-only optimizations.
- 32-bit XCL data width and 32-bit NPI data width.
- 32-bit address offset (the optional address offset is added to the XCL transaction address to compute the physical memory address to be accessed.)
- Auto-detection of MicroBlaze parameters.
- Optimized state machine and data paths when connected to MicroBlaze processor.
- Configurable pipeline stages for latency vs. frequency optimizations.
- Support for two XCL buses connected to one NPI port. The design allows for connecting MicroBlaze processor D-side and I-side XCL to a single NPI port for reduced resources with minimal performance hit.
- Target word first reads (except in the case of the `DXCL2` and `IXCL2` SUBTYPES.)

XCL Overview

The Xilinx CacheLink (XCL) PIM is highly optimized and configurable that allows you to connect a MicroBlaze soft processor XCL bus interface to SDRAM, DDR, and DDR2 memories with MPMC.

The XCL PIM translates XCL commands to NPI transactions to perform reads and writes to memory. Each XCL PIM instantiation connects to MPMC NPI using a 32-bit data path. The XCL bus interface supports fixed burst size reads and either fixed burst writes, or word, half-word, byte transaction sizes. The XCL signaling is a simple FIFO-style interface based on the FSL bus interface protocol.

Transactions are initiated by the master by pushing commands into the XCL PIM. If the operation is a write, then the master also pushes the correct number of data beats into the FIFO immediately following the write command. If the operation is a read, then the data will be returned by another FSL channel where the data is popped out of the FIFO-style interface.

The following figure is a block diagram of the NPI to XCL translation using a MicroBlaze processor.

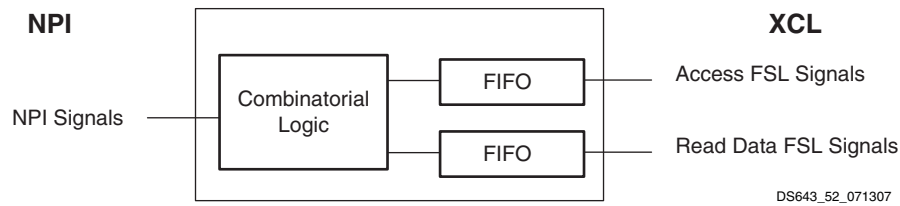


Figure 13: MicroBlaze Processor XCL Block Diagram

Connecting XCL to a MicroBlaze Soft Processor

When connecting the MicroBlaze IXCL and DXCL buses to the MPMC XCL PIM, the tools automatically optimize the configuration and set many of the parameters. The following table lists the automatically configured parameters and their values when connected to a MicroBlaze

Table 54: Auto-Computed Parameter Values Inherited from MicroBlaze Bus Connections

MPMC Parameter Name	Auto-computed MPMC Parameter Value	MicroBlaze Bus Interface Name	MicroBlaze Parameter Settings
C_PIM<Port_Num>_SUBTYPE	IXCL	IXCL	C_ICACHE_INTERFACE = 0
	IXCL2	IXCL	C_ICACHE_INTERFACE = 1 ⁽¹⁾
	DXCL	DXCL	C_DCACHE_INTERFACE = 0
	DXCL2	DXCL	C_DCACHE_INTERFACE = 1 ⁽¹⁾
C_XCL<Port_Num>_LINESIZE	4	IXCL	C_ICACHE_LINESIZE = 4
	8	IXCL	C_ICACHE_LINESIZE = 8
	4	DXCL	C_DCACHE_LINESIZE = 4
	8	DXCL	C_DCACHE_LINESIZE = 8
C_XCL<Port_Num>_WRITEXFER	0	IXCL	None
	1	DXCL	None

1. Not available in MicroBlaze versions earlier than v7.20a

Typical MicroBlaze transfers over the XCL PIM consists 4- or 8-word reads (determined at synthesis) and single word, half-word, or byte writes (if using a data-side channel.) When connected to a MicroBlaze version 7.20.a or greater and the MicroBlaze parameter C_DCACHE_INTERFACE = 1, the XCL PIM also supports cache-line writes of either 4- or 8-word. The memory to XCL and by extension MicroBlaze clock frequency can either be 1:1 or 2:1 and is auto-detecting. The XCL PIM contains minimal buffering and configurable pipeline stages for low resource utilization and provides support for high frequencies. The XCL PIM with the addition of the B port (see "Dual XCL Buses on One XCL PIM," page 105) supports connecting two XCL buses to one MPMC Port. This provides lower utilization of resources with minimal impact on performance when connecting instruction-side XCL and that-side XCL MicroBlaze buses to MPMC.

XCL Configuration Options

Dual XCL Buses on One XCL PIM

For resource optimization when connecting two XCL buses to MPMC it is recommended that the MPMC parameter, `C_XCL<Port_Num>_B_IN_USE`, is set to 1. This enables a second XCL bus to connect to MPMC by the bus that is designated as `XCL<Port_Num>_B`, while only consuming one of the available ports on MPMC. For example, if Port 0 of MPMC is configured as XCL and `C_XCL0_B_IN_USE = 1`, then connect the MicroBlaze IXCL bus to MPMC `XCL0` bus. Similarly, connect the MicroBlaze DXCL bus to the MPMC `XCL0_B` bus. The XCL PIM will internally arbitrate between the two buses with priority given to the `XCL0_B` bus if they are both idle and receive XCL requests at the same time. If both ports are requesting continuously, round-robin arbitration goes between the two ports to avoid starvation, which allows you to connect up to 16 XCL buses to MPMC. The relationship between the IXCL and DXCL transactions on MicroBlaze results in a minimal performance impact. If you are using custom XCL masters that have high bandwidth requirements, it might be more suitable to use two separate XCL PIMs.

When `C_XCL<Port_Num>_B_IN_USE` is set to 1, three additional parameters are available to be set:

- `C_PIM<Port_Num>_B_SUBTYPE`
- `C_XCL<Port_Num>_B_LINESIZE`
- `C_XCL<Port_Num>_B_WRITEXFER`

These parameters function identically to the parameters of the same name with a suffix of `_B`, consequently they are not explicitly discussed. The bus interface `XCL<Port_Num>_B` becomes visible as a valid XCL bus interface target when dual XCL mode is enabled.

XCL PIM Subtypes

The `C_PIM<Port_Num>_SUBTYPE` parameter is automatically detected when connected to a MicroBlaze processor. The subtype settings of IXCL, IXCL2, DXCL and DXCL2 change the operation of XCL slightly to improve maximum frequency and reduce latency when connected to MicroBlaze.

The following table provides a summary of the differences between the subtypes:

Table 55: XCL Supported Features by Subtype

XCL SUBTYPE	Supports Target Word First Read Transactions	Supports Word Write Operations	Supports Cacheline Write Operations	Supports Linesize 1 & 16	Supports Linesize 4 & 8	Supports Standard FSL Handshaking
XCL	Yes	Yes ⁽¹⁾	Yes ⁽²⁾ ,	Yes	Yes	Yes
IXCL	Yes	No	No	No	Yes	Yes
DXCL	Yes	Yes	No	No	Yes	Yes
IXCL2	No	No	No	No	Yes	No
DXCL2	No	Yes	Yes	No	Yes	No

1. Valid if `C_XCL<Port_Num>WRITEXFER = 1`
2. if `C_XCL<Port_Num>WRITEXFER = 2`

XCL Line Size and Write Transfers

The `C_XCL<Port_Num>_LINESIZE` parameter specifies whether the XCL line size is 1-, 4-, 8-, or 16-words. This line size is set for all XCL reads. The 4-, 8-, or 16-word transfers are target word first cache-line transfers. For Writes, the `C_XCL<Port_Num>_WRITEEXFER` parameter specifies whether XCL Write transactions are:

- 0 - Disabled
- 1 - One word only
- 2 - The same size as the Reads specified in `C_XCL<Port_Num>_LINESIZE`

If the XCL PIM is used, the NPI data width is fixed automatically at 32 bits. XCL data and address are labeled with big-endian bit and byte ordering as described in [Figure 9, page 80](#).

XCL Pipeline Stages

The `C_XCL<Port_Num>_PIPE_STAGES` parameter can be used to adjust the number of pipeline stages in the XCL PIM. There are three different pipelines with four possible settings for

`C_XCL<Port_Num>_PIPE_STAGES`:

- 0 - No pipelines are enabled.
- 1 - One pipeline is enabled: This setting enables a pipeline register on the output of the XCL Read data FIFO. This pipeline helps cross clock boundaries on the read channel. Adds one cycle of Read latency.
- 2 - Two pipelines are enabled: This setting enables an additional pipeline on the NPI Read FIFO Empty signal. This helps alleviate timing path on the NPI read FIFO control signals, and adds one cycle of Read latency.
- 3 - Three pipelines are enabled: This setting enables an additional pipeline on the output of the XCL Access FIFO.

Note: Pipelines added in with values 1 and 2 each add a cycle of latency to each XCL Read. The pipeline added with value 3 might add one cycle of latency to both XCL Reads and XCL Writes.

XCL Clock Requirements

The XCL PIM runs at an integer ratio of the MPMC memory clock rate. This clock ratio is automatically detected during reset and can be a ratio of either 1:1 or 2:1. The XCL PIM clock must be synchronous and rising edge-aligned to the MPMC memory clock.

XCL Additional Information

For additional details on XCL, including signaling protocol and waveforms, see the *MicroBlaze Processor Reference Guide*. The "[Reference Documents](#)," [page 193](#) contains a link to the document.

Soft Direct Memory Access Controller for LocalLink Interfaces

The Soft Direct Memory Access (SDMA) Controller PIM, which is integrated into MPMC, provides high-performance Direct Memory Access (DMA) for streaming data. The SDMA provides two channels, one for receiving data and one for transmitting data. Transmit and Receive are accomplished through two LocalLink interfaces.

This section contains the following subsections:

- [SDMA Features](#)
- [SDMA Overview](#)
- [SDMA Operation](#)
- [DMA Operation Descriptors](#)
- [SDMA Error Conditions](#)
- [Managing Descriptors](#)
- [SDMA LocalLink Interface](#)
- [SDMA Interrupts and Errors](#)
- [SDMA Transaction Timing](#)
- [SDMA Registers](#)

SDMA Features

The SDMA Controller for LocalLink interfaces contains the following features:

- Direct plug-in to MPMC
- Simultaneous, independent Transmit and Receive DMA operations
- Per-channel Interrupt Event reporting
- Interrupt Coalescing
- Xilinx PLBv4.6 interface for control as a register access
- User-defined LocalLink headers and footers⁽¹⁾
- Dynamic Scatter Gather Buffer Descriptor modification
- SDMA supports the following configurations only:
 - 32- and 64-bit for DDR
 - 64-bit for SDRAM

SDMA Overview

The SDMA uses a Native Port Interface (NPI), two LocalLink interfaces, and a PLB interface. The NPI connects the SDMA controller into the MPMC PIM. The two LocalLink interfaces, a Transfer (TX) and a Receive (RX), provide full duplex LocalLink device access to the SDMA. The PLB interface allows the CPU to interact with the SDMA for initiating DMA processes or status gathering. The PLB and LocalLink data and address signals are labelled with big-endian bit/byte ordering as described in [Figure 9, page 80](#).

1. For use with functions such as checksum off loading

The following figure provides a high-level block diagram of SDMA.

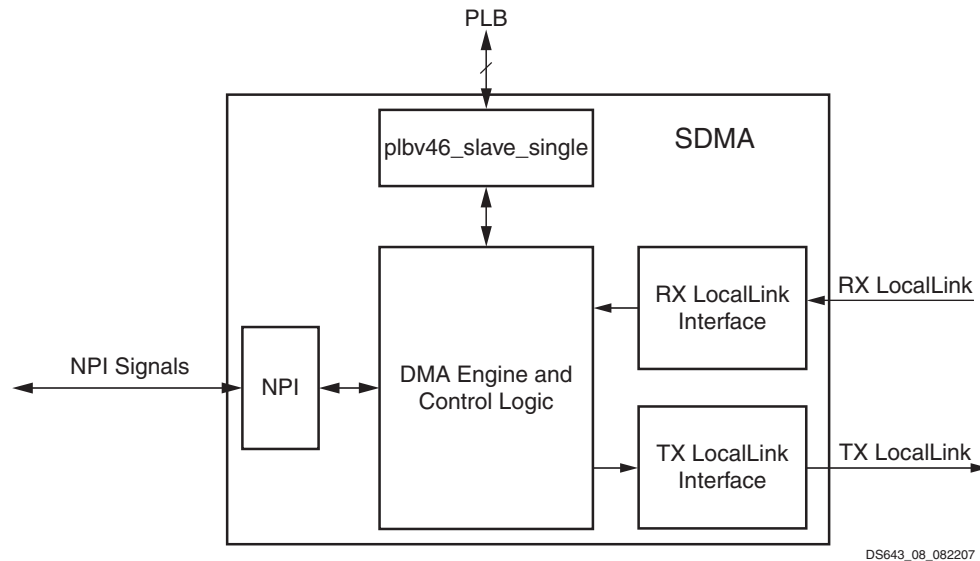


Figure 13: High Level SDMA Block Diagram

SDMA Operation

Scatter Gather Operation

Scatter Gather operation has the concept of descriptor chaining which allows a packet to be described by more than one descriptor. Typical use for this feature is to allow storing or fetching of ethernet headers from one location in memory and payload data from another location. Software applications that can take advantage of this can improve throughput.

SDMA uses Start of Packet bit (SOP) and End of Packet bit (EOP) to delineate packets in a buffer descriptor chain. When the DMA fetches a descriptor with the `STS_CTRL_APP0.SOP` bit set this triggers the start of a packet. The packet continues fetching subsequent descriptors until a descriptor with the `STS_CTRL_APP0.EOP` bit is set.

For the receive channel, when a packet has been completely received the SDMA acquires the footer fields of the LocalLink stream and writes these values to APP0 through APP4 fields of the last descriptor. SDMA also sets `STS_CTRL_APP0.EOP=1` indicating to the software that the current receive buffer as described by the descriptor contains the last of the packet data.

For the transmit channel, SDMA uses the `STS_CTRL_APP0.EOP` bit. When SDMA determines that the `STS_CTRL_APP0.EOP` bit is set in `STS_CTRL_APP0`, the SDMA completes the currently requested transfer and then terminates the LocalLink transfer with a LocalLink End of Frame (EOF).

Starting and Stopping DMA Operations

DMA operations can be started by writing an address to the respective `TAILDESC_PTR` register. When the start condition is met, `CHNL_STS.EngBusy` of the respective channel is set and the SDMA fetches the first descriptor that is pointed to by the address in the `CURDESC_PTR` register.

DMA descriptor processing will continue until a descriptor that has the `TAILDESC_PTR = CURDESC_PTR` for the respective channel has finished being processed.

DMA Operation Descriptors

SDMA operation requires a common memory-resident data structure that holds the list of DMA operations to be performed. This list of instructions is organized into what is referred to as a Descriptor Chain. The descriptor shown in the following table is the basis for organizing the DMA operations as a Linked List. descriptors are fetched through the NPI. A similar mechanism is used for performing descriptor updates. The following table lists the SDMA descriptors:

Table 56: SDMA Descriptors

Name	Description	Purpose
NXTDESC_PTR	Next Descriptor Pointer	Specifies where in memory the next descriptor is to be fetched.
CURBUF_ADDR	Buffer Address	Specifies where in memory the buffer is for receiving or transmitting data.
CURBUF_LENGTH	Buffer Length	For Transmit, specifies the amount of data in bytes that are to be transmitted. For Receive, indicates the amount of space in bytes that is available to receive data.
STS_CTRL_APP0	Status/Control and App Data 0	Status/Control for controlling and providing status to the DMA transfer of application specific data.
APP1	Application Data 1	Application specific data.
APP2	Application Data 2	Application specific data.
APP3	Application Data 3	Application specific data.
APP4	Application Data 4	Application specific data.

The following table shows the STS_CTRL_APP0 register bits.

Table 57: STS_CTRL_APP0

Bit(s)	Name	Type	Description
0	Error	Status	Set by the DMA when a error is encountered.
1	IrqOnEnd	Control	When set, causes the DMA to generate an interrupt event when the current descriptor has been completed.
2	Reserved	N/A	Undefined
3	Completed	Status	When set, indicates that the current descriptor has been completed.
4	SOP	Status/Control	Transmit Channel (Control): Set by software in the descriptor indicating this buffer descriptor is the first descriptor of a packet. Receive Channel (Status): Set by DMA in the descriptor indicating that a start of packet was received on LocalLink.
5	EOP	Status/Control	Transmit Channel (Control): Set by software in the descriptor indicating this buffer descriptor is the last descriptor of a packet. Receive Channel (Status): Set by DMA in the descriptor indicating that an end of packet was received on LocalLink.
6	Reserved	N/A	Undefined.
7	Reserved	N/A	Undefined.
8:31	Application Data 0	N/A	Application specific data.

Each field of the descriptor is 4 bytes in length and corresponds to either one of the DMA channel registers or user application fields.

- For transmit channels, the application data fields (App0 to App4) of the first descriptor are transmitted as part of the Header of the LocalLink Transmit Data stream.
- For receive channels, the Application Data Fields of the last buffer descriptor will be updated with a portion of the Footer of the LocalLink Receive Data stream.

See "LocalLink Headers and Footers," page 118 for more information on LocalLink headers and footers.

The following figures shows descriptors organized into a linked list. SDMA will successively perform the DMA operations specified in the descriptors up to and including the descriptor with the CURDESC_PTR = TAILDESC_PTR.

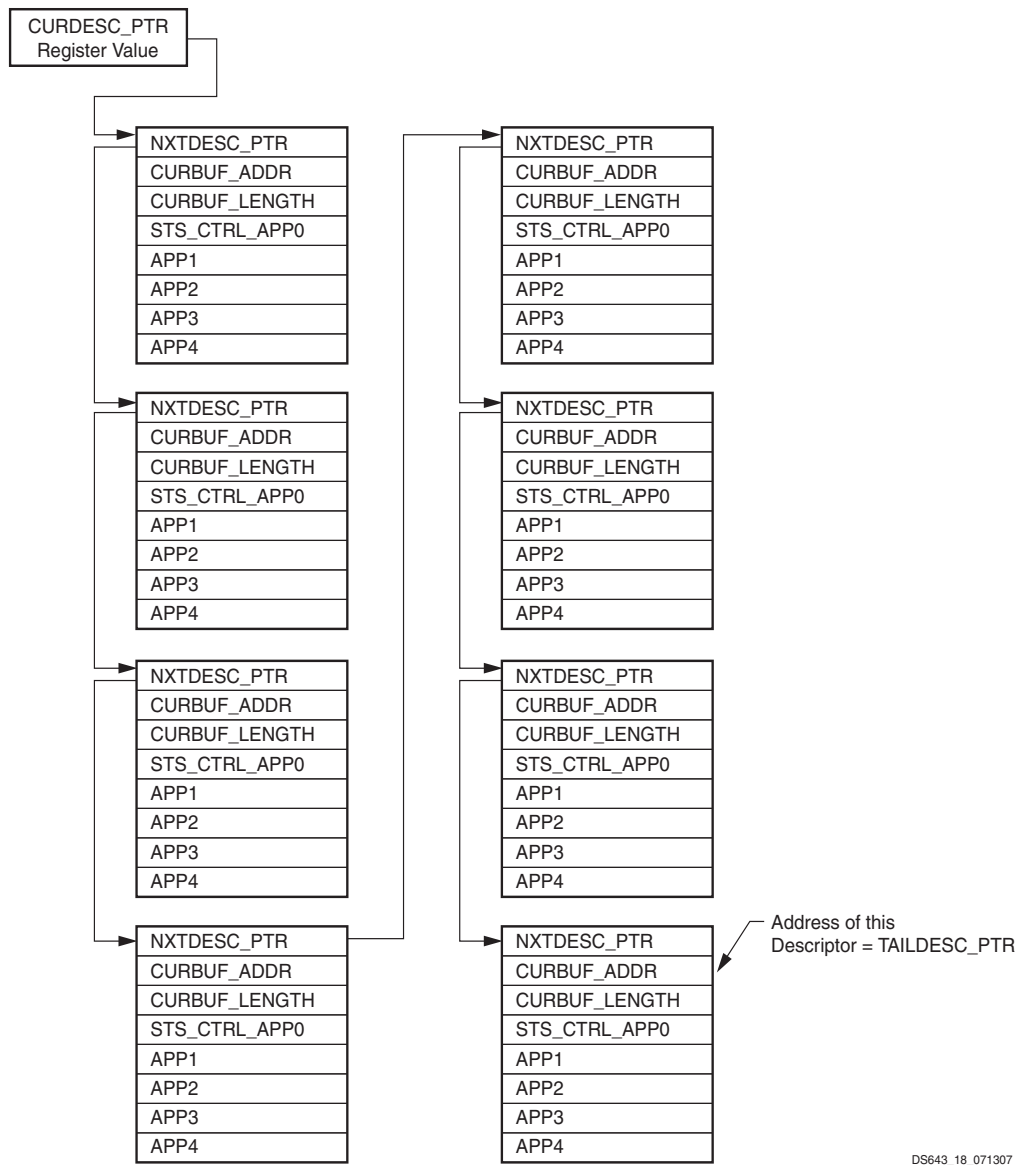


Figure 14: Linked List of Descriptors

The following figure shows descriptors organized into a buffer ring for dynamic descriptor update. The buffer ring is for a Transmit channel as evidenced by `STS_CTRL_APP0.SOP=1` and `STS_CTRL_APP0.EOP=1` tags. Note that packet 4 is specified by a single descriptor and others by more than one consecutive descriptor. The address of the last ready packet is equal to the `TAILDESC_PTR`, giving a sentinel position in the ring.

Note: Even if descriptors are contiguously allocated, they are required to be linked through the `NXTDESC_PTR` field.

Note: For receive channels, `STS_CTRL_APP0.SOP` and `STS_CTRL_APP0.EOP` are set by SDMA and updated to memory for use by the software application.

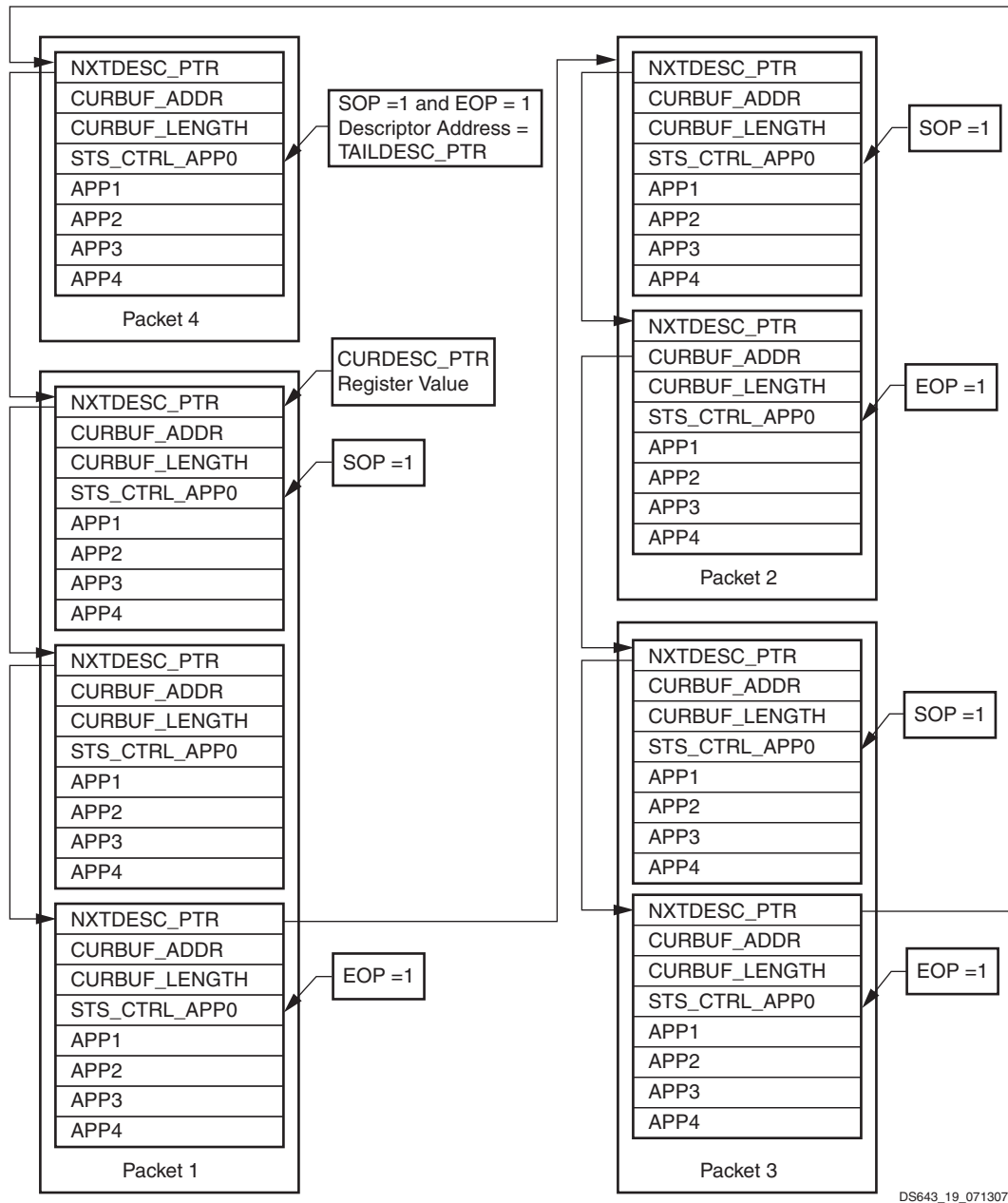


Figure 15: Descriptors Organized Into a Buffer Ring

Transmit Channel Operation

The following steps are required to execute a packet transmit operation:

1. The software creates a chain of descriptors:
 - a. Specify in the descriptor the packet boundaries using the `STS_CTRL_APP0.SOP` and `STS_CTRL_APP0.EOP` bits.
 - b. Specify the address to the data buffer to transmit in the `CURBUF_ADDR` field.
 - c. Specify the amount of data to transfer for each descriptor in the `CURBUF_LENGTH` field.
 - d. Specify a pointer to the next descriptor in `NXTDESC_PNT`.
2. Software prepares the DMA Channel Registers (Order of steps **a** and **b** are not critical):
 - a. Set up interrupts if so desired by writing to the `TX_CHNL_CTRL` register, specifying interrupt coalescing information (if enabled).
 - b. Set a pointer to the first descriptor in the `TX_CURDESC_PTR` register.
3. The software starts SG automation by writing the pointer to the last descriptor to fetch into the `TAILDESC_PTR` register.
4. SDMA requests the first descriptor pointed to by the `TX_CURDESC_PTR` register.
5. Upon completion of the descriptor fetch, the DMA cycle begins.
6. If the currently fetched descriptor has `STS_CTRL_APP0.EOP` set, the data of that descriptor is transmitted and the Master completes the packet on the LocalLink with and End of Payload, (EOP) and End of Frame (EOF). If the currently fetched descriptor does *NOT* have `STS_CTRL_APP0.EOP` set the packet continues.
7. At the completion of each descriptor the channel register information is updated to the corresponding descriptor memory location.
8. This process continues until the descriptor `TX_CURDESC_PTR = TX_TAILDESC_PTR` is completed processing.

Receive Channel Operation

The following steps are required to execute a receive operation:

1. The software creates a chain of descriptors.

Note: SDMA supports multiple descriptors being used to describe a single packet. The `STS_CTRL_APP0.EOP` bit is set by SDMA in the descriptor associated to the buffer containing the last byte of the received packet.

 - a. Specify in the `CURBUF_ADDR` field of each descriptor the address to the start of the associated buffer for receiving data.
 - b. Specify in the `CURBUF_LENGTH` field of each descriptor the available size of the associated buffer for receiving data. The sum total of the Length field/s in the descriptors must specify a byte count that is large enough to hold an entire packet.
 - c. Specify a pointer to the next descriptor in `NXTDESC_PNT` field.
2. The software prepares the DMA Channel registers (Order of steps **a** and **b** are not critical):
 - a. Set the pointer to the first descriptor in `RX_CURDESC_PTR` register.
 - b. Set up Interrupts if required by writing to the `RX_CHNL_CTRL` register, specifying interrupt coalescing information (if enabled).
3. The software starts SG Automation by writing the pointer to the last descriptor to fetch into the `RX_TAILDESC_PTR` register.

4. SDMA requests the first descriptor pointed to by the `RX_NXTDESC_PTR` register. For receive channels, the User Application fields are updated in the descriptor during the descriptor update phase of processing.
5. Upon completion of the descriptor fetch, the DMA cycle begins.

This process continues until the descriptor `RX_CURDESC_PTR = RX_TAILDESC_PTR` has completed processing.

SDMA Error Conditions

SDMA performs several error checking functions to ensure proper operation of the DMA engine. If an error occurs then the channel on which the error is detected is halted, and the channel status register Error bit for the channel is set to 1. If possible, the Error bit for the current descriptor is set to 1 also; although, depending on the error condition, this might not get updated to remote memory.

To recover from an error condition the SDMA must be reset either by a hard reset or by issuing a soft reset (such as setting `SwReset = 1` in the DMA Control register. The following table lists the possible errors that can be flagged and their causes.

Table 58: Descriptor Errors

Error	Description
TX_CHNL_STS.CurPErr RX_CHNL_STS.CurPErr	<p>Current Descriptor Pointer Error This error occurs if the Current Descriptor Pointer does not fall within the <code><prefix>_BASEADDR</code> to <code><prefix>_HIGHADDR</code> range⁽¹⁾. Descriptors must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.</p>
TX_CHNL_STS.TailPErr RX_CHNL_STS.TailPErr	<p>Tail Descriptor Pointer Error This error occurs if the Tail Descriptor Pointer does not fall within the <code><prefix>_BASEADDR</code> to <code><prefix>_HIGHADDR</code> range. Descriptors must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.</p>
TX_CHNL_STS.NxtPErr RX_CHNL_STS.NxtPErr	<p>Next Descriptor Pointer Error This error occurs if the Next Descriptor Pointer does not fall within the <code><prefix>_BASEADDR</code> to <code><prefix>_HIGHADDR</code> range⁽¹⁾. Descriptors must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.</p>
TX_CHNL_STS.AddrErr RX_CHNL_STS.AddrErr	<p>Buffer Address Error This error occurs if the Buffer Address does not fall within the <code><prefix>_BASEADDR</code> to <code><prefix>_HIGHADDR</code> range⁽¹⁾. All transmit and receive data buffers must reside within memory as mapped by the base address and high address of the NPI. Addresses outside this range are detected and flagged as errors.</p>

Table 58: Descriptor Errors (Cont'd)

Error	Description
TX_CHNL_STS.CmpErr RX_CHNL_STS.CmpErr	<p>Complete Bit Error</p> <p>This error occurs if a descriptor is fetched with the Complete bit set to 1 (as in <code>STS_CTRL_APP0.Completed=1</code>).</p> <p>This error indicates that a descriptor, which had been already used by SDMA, is being processed again, before the software application has had a chance to process the descriptor and associated data buffer.</p> <p>This error checking can be disabled or enabled by setting/clearing <code>C_COMPLETED_ERR_TX</code> and <code>C_COMPLETED_ERR_RX</code> for the associated channel.</p> <p>Setting the parameters to 1 enables checking and setting the parameters to 0 disables checking.</p>
TX_CHNL_STS.BsyWr RX_CHNL_STS.BsyWr	<p>Busy Write Error</p> <p>This error occurs if the Current Descriptor Pointer register is written to via the PLB v4.6 slave port while the SDMA engine is busy.</p> <p>Examining <code>TX_CHNL_STS.EngBusy</code> and <code>RX_CHNL_STS.EngBusy</code> for the respective channel will indicate to the software application whether or not the channel is busy.</p> <p>If <code>EBsy = 1</code> then the channel is busy and The Current Descriptor Pointer should not be written to by the software application.</p>

1. where: `C_ALL_PIMS_SHARE_ADDRESSES =`
`0:<prefix> = C_MPMC`
`1:<prefix> = C_PIM<Port_Num>`

Managing Descriptors

Prior to starting DMA operations, the software application must set up a descriptor or chain of descriptors. After the descriptors are set, SDMA begins processing the descriptors, fetches, processes, and then updates the descriptors. By analyzing the descriptors, the software application can read status on the associated DMA transfer, fetch user information on receive channels, and determine completion of the transfer. With this information the software application can manage the descriptors and data buffers.

When a descriptor has been processed by the SDMA, the transfer status is updated into the `STS_CTRL_APP0` field. When the software application sets up the descriptor chain, the status bits in the Control/Status field of each descriptor must be set to zero. The status bits are: Error, Completed, and EngBusy.

For receive channels SOP and EOP are status bits and must be set to zero. For transmit channels SOP and EOP are control bits set by the software application. This allows the software application to determine when a descriptor has been processed and if there were errors during processing.

As each descriptor is updated into remote memory, `STS_CTRL_APP0.Completed` bit is set to 1. By checking the `STS_CTRL_APP0.Completed` bit and walking through the descriptor chain, the software application can determine which descriptors have been completed.

Figure 16, page 115 shows remote memory where software has constructed a descriptor chain. The SDMA, shown on the right, fetches descriptors, processes them, and then updates the descriptor in remote memory providing status. As shown the `STS_CTRL_APP0.Completed=1` for the descriptors that have been processed.

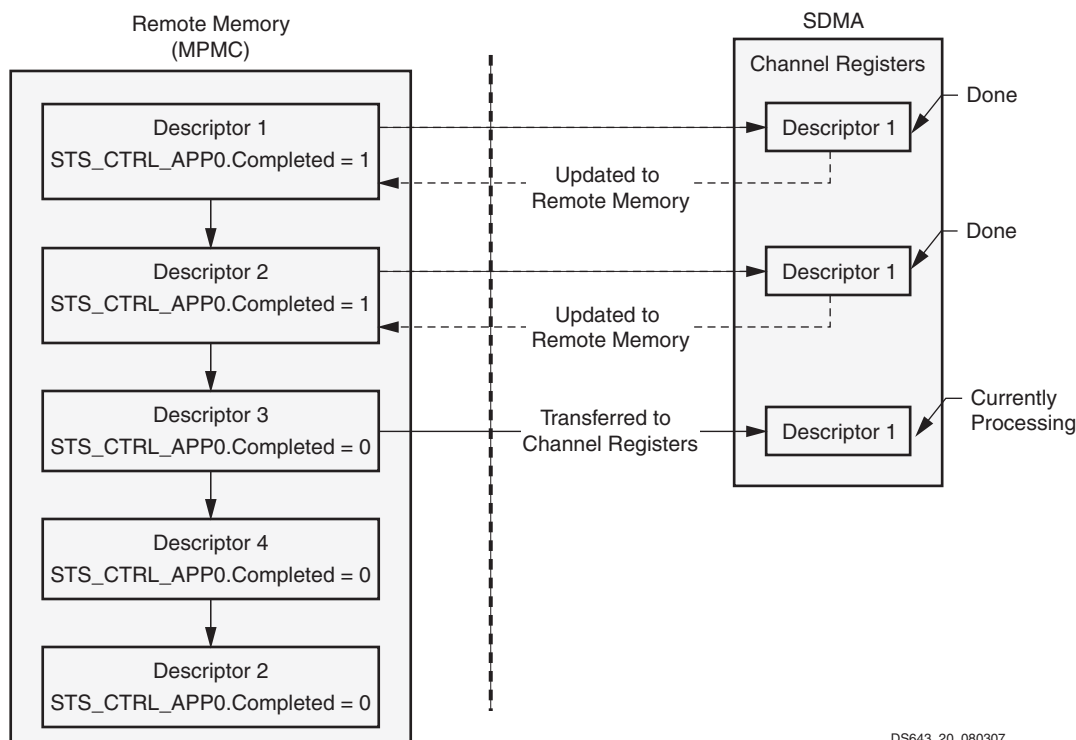


Figure 16: Software - Hardware Descriptor Processing

For further clarity on Receive channels, by monitoring the `STS_CTRL_APP0.Completed` bit in the descriptor, the software application can determine which data buffers, as described by the descriptor, have received data and need to be processed. By looking for `STS_CTRL_APP0.SOP` and `STS_CTRL_APP0.EOP` the software application can determine the start and end buffers containing a packet.

For Transmit channels, `STS_CTRL_APP0.Completed=1` indicates to the software application that the data in the associated buffer has been transmitted and is free to be modified.

If an error occurs during the DMA transfer, either during the descriptor fetch and update or during the actual requested DMA transfer, the `STS_CTRL_APP0.Error` bit is set.

If an error occurs during a transfer, an `IRQ_REG.ErrIrq` interrupt is generated, the respective `CHNL_STATUS.EngBusy` is cleared to 0, and DMA operations are halted. At this point the Software application must issue a reset to the SDMA to reset and resume DMA operations by writing a 1 to the `DMA_CONTROL.SwReset` bit.

Dynamic Descriptors

At times it might be necessary to update a descriptor chain while the SDMA is actively processing the buffer descriptors. This can be achieved when a channel is configured to operate in TailPointer mode.

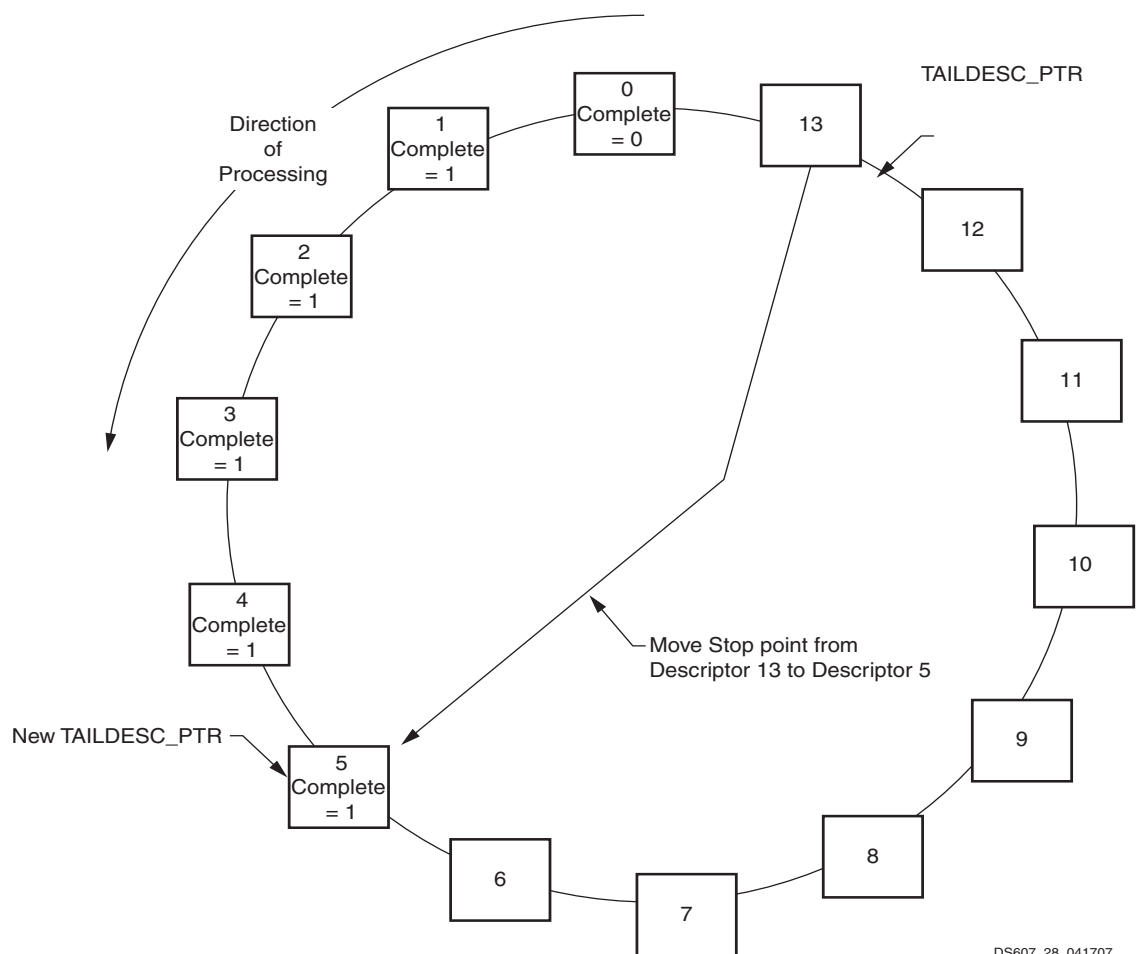
Appending a Descriptor Chain

The SDMA is designed so software applications can append new buffer descriptor chains into an already active chain with minimal effort on the part of the software application. This can be accomplished by updating the `TAILDESC_PTR` if the descriptors are arranged in a ring, or by the use of a dummy descriptor where `NXTDESC_PTR` and `TAILDESC_PTR` can be updated to point to the start and end of the appended descriptor chain. The following subsections describe these methods.

Modifying Descriptors Using A Dummy Descriptor

To append a descriptor or set of descriptors to a descriptor chain that is not arranged in a ring, two updates need to occur, one to the `NXTDESC_PTR` and one to the `TAILDESC_PTR`. To avoid a race condition, one in which the SDMA has already fetched the `NXTDESC_PTR` of the last descriptor in the chain before the software application can update the last descriptor to point to the newly appended descriptors, a dummy descriptor is used.

As an example, assume the software application has set up the following descriptor chain, Descriptor 0 to Descriptor 9 with a dummy descriptor, DMY, at the end of the chain. Also assume that the `TAILDESC_PTR` register contains the address of Descriptor 9. The following figure illustrates an append to a descriptor chain.



DS607_28_041707

Figure 17: Descriptor Chain Append

If no updates were made to the descriptor chain, the DMA controller would process Descriptor 0 through Descriptor 9 and would stop after Descriptor 9 because the original `TAILDESC_PTR` would equal the `CURDESC_PTR` of Descriptor 9. The `NXTDESC_PTR` that SDMA has stored would be pointing to the dummy descriptor.

For this example, assume that the software application has started DMA operation by setting the `TAILDESC_PTR` address to Descriptor 9. The SDMA will fetch Descriptor 0 and begin processing Descriptor 0. At this point the software determines that it is necessary to append Descriptors 10, 11, and 12 to the already active chain.

To append the new descriptors:

1. Set up Descriptors 10, 11, and 12 in remote memory.
2. Update the `NXTDESC_PTR` of the dummy descriptor in memory space to point to descriptor.
3. Update the `TAILDESC_PTR` register with the address of Descriptor 12.

If SDMA was in the middle of processing Descriptor 9 and there was no dummy descriptor, a race condition would occur. The `NXTDESC_PTR` that SDMA would pick up would be pointing to a potentially invalid location.

If the software application were to update `NXTDESC_PTR` in memory space and then update the `TAILDESC_PTR` in the SDMA, the SDMA would not update its `NXTDESC_PTR` register from memory space and, consequently, would fetch the next descriptor from an invalid place.

With `NXTDESC_PTR` pointing to a dummy descriptor, the SDMA fetches the dummy descriptor, after the `TAILDESC_PTR` is updated, with the correctly updated `NXTDESC_PTR` pointing to the newly appended descriptors. Because the dummy descriptor has an SOP, an EOP, and a length set to zero, the descriptor is dropped and the next descriptor is fetched.

Modifying a Descriptor Using A Descriptor Ring

When using a descriptor ring to modify a descriptor, no new descriptors are added. Consider a situation where 14 descriptors are arranged in a loop and the stop point from Descriptor 13 must be moved to Descriptor 5 without stopping the chain. It is assumed that the new descriptor chain has been created in remote memory and that the SDMA is actively processing the original descriptor chain. To perform that operation: Modify the already processed descriptors, such as the descriptor with the field setting `STS_CTRL_APP0.Completed = 1`. Then move `TAILDESC_PTR` to point at Descriptor 5.

SDMA LocalLink Interface

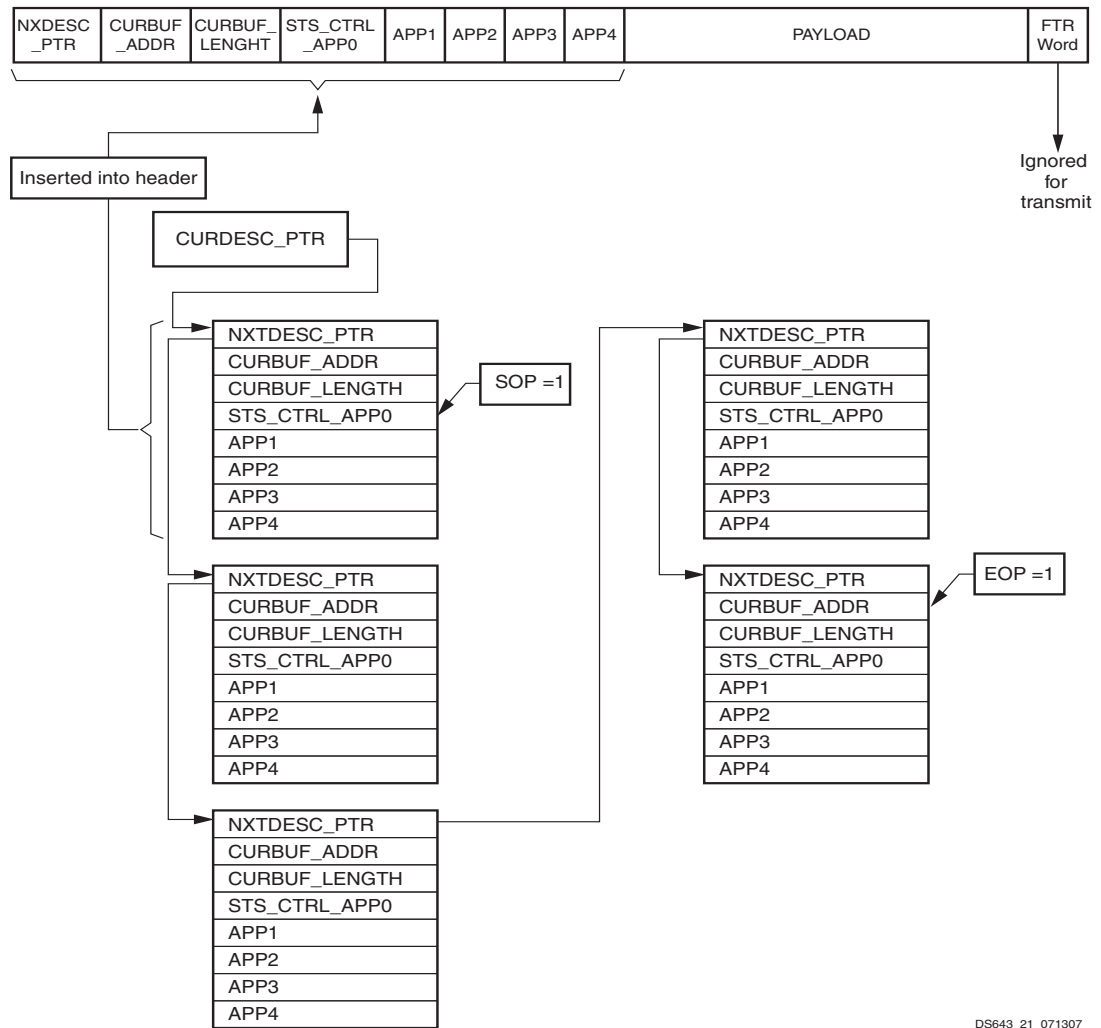
The LocalLink Interface comprises:

- [LocalLink Headers and Footers](#)
- [Transmit LocalLink Interface](#)
- [Receive LocalLink Interface](#)

LocalLink Headers and Footers

SDMA uses LocalLink headers and footers to pass data in and out of the Buffer Descriptor User Application (APP#) fields. This allows the software application to pass user-defined data to and from the user IP using the LocalLink data stream.

For the transmit channel, the first descriptor describing a packet which includes APP0 to APP4, is transferred in the header of the LocalLink data stream as shown in the following figure. For Transmit channel the Header is always 8 words long and the footer is always one word long. The footer data should be ignored. The following figure illustrates the headers and footers.



DS643_21_071307

Figure 18: LocalLink Transmit Data Stream Header Assignment

For the receive channel as shown in the following figure:

- The header must be only one word long and the data is discarded by the SDMA.
- The footer must be exactly eight words long.
- The last descriptor is populated with the LocalLink footer user App Fields, App0 and App1 to App4.

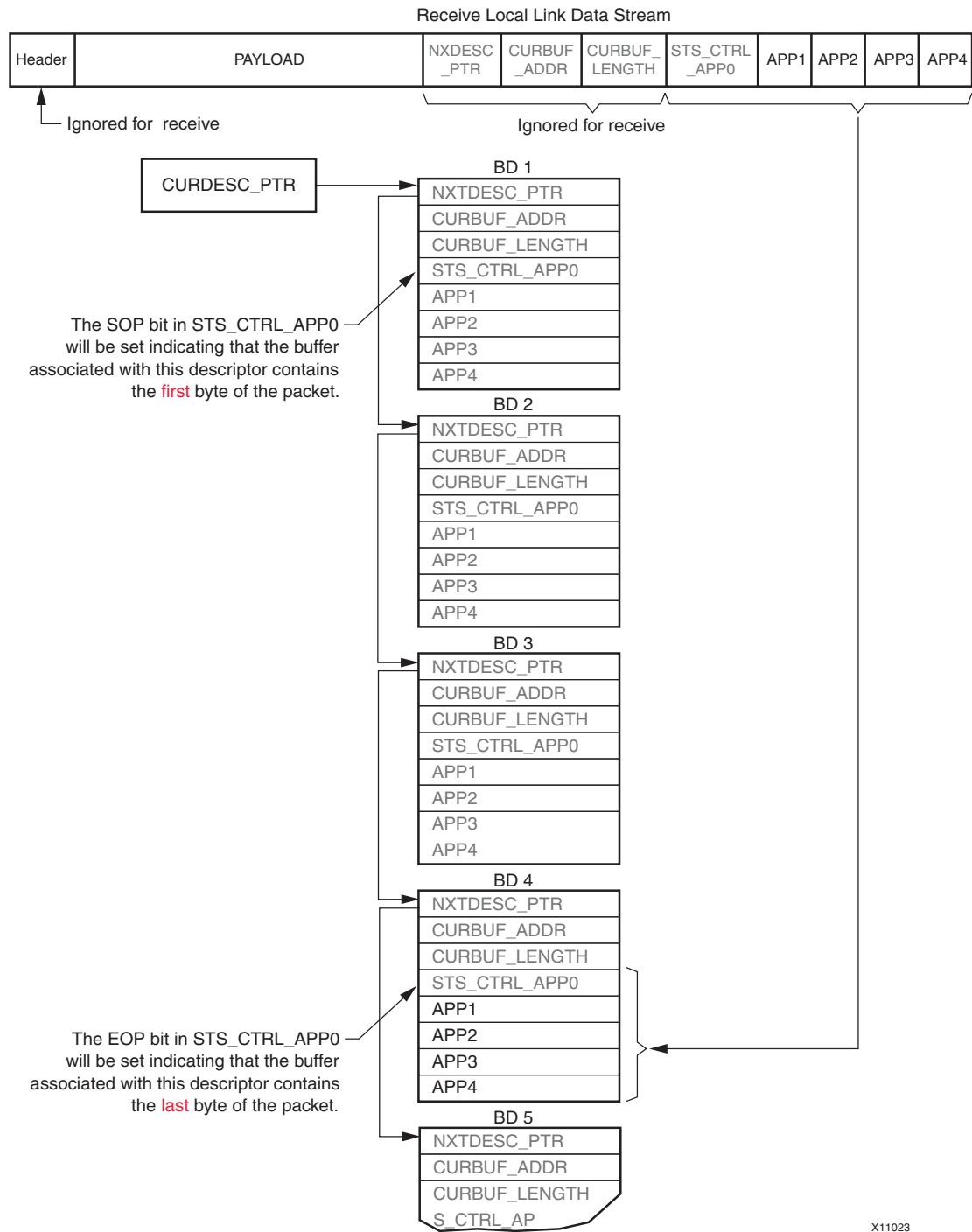


Figure 19: LocalLink Receive Data Stream Footer Assignment

Transmit LocalLink Interface

The Transmit LocalLink and Byteshifter logic take data from the appropriate place in memory and move the data across the LocalLink interface. This concept is shown in [Figure 20, page 121](#).

In this example:

1. The SDMA reads the descriptor in the address range p to $p+1C$ and sends it to the LocalLink as the header. The payload is 136 bytes and starts at address $m+79$.
2. The Transmit Byteshifter sends data acknowledges to the memory controller while keeping the `Src_Rdy` signal to the LocalLink de-asserted, because address $m+79$ is not 32-word aligned.
3. Data from the address range m to $m+78$ is discarded.
4. Data is offset by 78 bytes, so the first byte of data occurs on the second byte location on the posedge of the DDR SDRAM.
5. The Transmit Byteshifter takes the posedge (x 0 1 2) and negedge (3 4 5 6) data from DDR SDRAM, which are both present at the time, recombines them to form a new, correctly shifted word (0 1 2 3), and sends it over the LocalLink as the payload.
6. At the end of the first 32-word burst read (B32R), three bytes are left over and kept in the Byteshifter.
7. When the second burst occurs, the three bytes from the Byteshifter are combined with the first byte of the second burst and sent over LocalLink. This happens again between the second and third burst.
8. The fourth burst is generated due to a second descriptor. It too describes a buffer that begins at an odd boundary; for example, offset $0x7E$. Bytes $r0$ and $r1$ are combined with the leftover bytes from the previous burst, n and $n+1$.
9. On the last word of the payload, the `Rem` signal is set to indicate which bytes of the word are valid. `Rem` is $0x3$ in this example to indicate that only the first two bytes are valid.
10. After byte $n+1$ is sent, the FIFOs in MPMC, which hold all 32 words of the burst, are reset to avoid extra data acknowledge.
11. For Tx transfer, the footer is not used. The status bits are written back to the status field of the descriptor.



DS607_16_073007

Figure 20: Transmit Byte Shift Example

Receive LocalLink Interface

The LocalLink and Byteshifter Rx logic receives data from the LocalLink interface and moves the data to the appropriate place in memory. This concept is shown in [Figure 21, page 123](#).

In this example:

1. The SDMA always ignores the Rx LocalLink Header, which must be only 1 word long.
2. The Payload is processed by the Rx Byteshifter and pushed into the MPMC Write FIFOs. The Payload is 270 bytes.
3. The data is pushed into the FIFOs in bursts of 32 words (B32W).
4. Data is stuffed into the FIFOs from address m through address $m+0x7C$ because the Payload is written to address $m+0x79$, which is not a 32-word aligned address.
5. When these bytes are written to memory, the byte enables are turned Off.
6. In the second B32W, all of the data is valid.
7. In the third B32W, only three bytes need to be written to memory. This means that the remaining 125 bytes need to be stuffed into the FIFOs at the end of the burst.
8. The length of buffer 1 was specified to be 138 bytes long in the Receive descriptor. After the first three bytes, B32Ws buffer 1 is full.
9. The remainder of the payload is transferred to buffer 2.
10. The fourth, fifth, and sixth transfers are similar to the first three transfers in that the first valid byte is not at an even boundary.
11. SDMA begins the fourth B32W at $r+00$, setting the byte enables to Off for all the bytes up to the first valid byte at $r+7E$.
12. The fifth B32W has all bytes valid, and, in the final B32W, only two bytes are valid.
13. The remaining 126 bytes are pushed to the FIFO with the byte enables set to Off.
14. After the Payload has been processed, the footer is processed and written to memory at address p .
15. The SDMA changes the byte enables of the first three words to prevent the Next Descriptor Pointer, Buffer Address, and Buffer Length values from being overwritten. Thus, only the values of STS_CTRL_APP0, APP1, APP2, APP3, and APP4 are updated in the memory space.

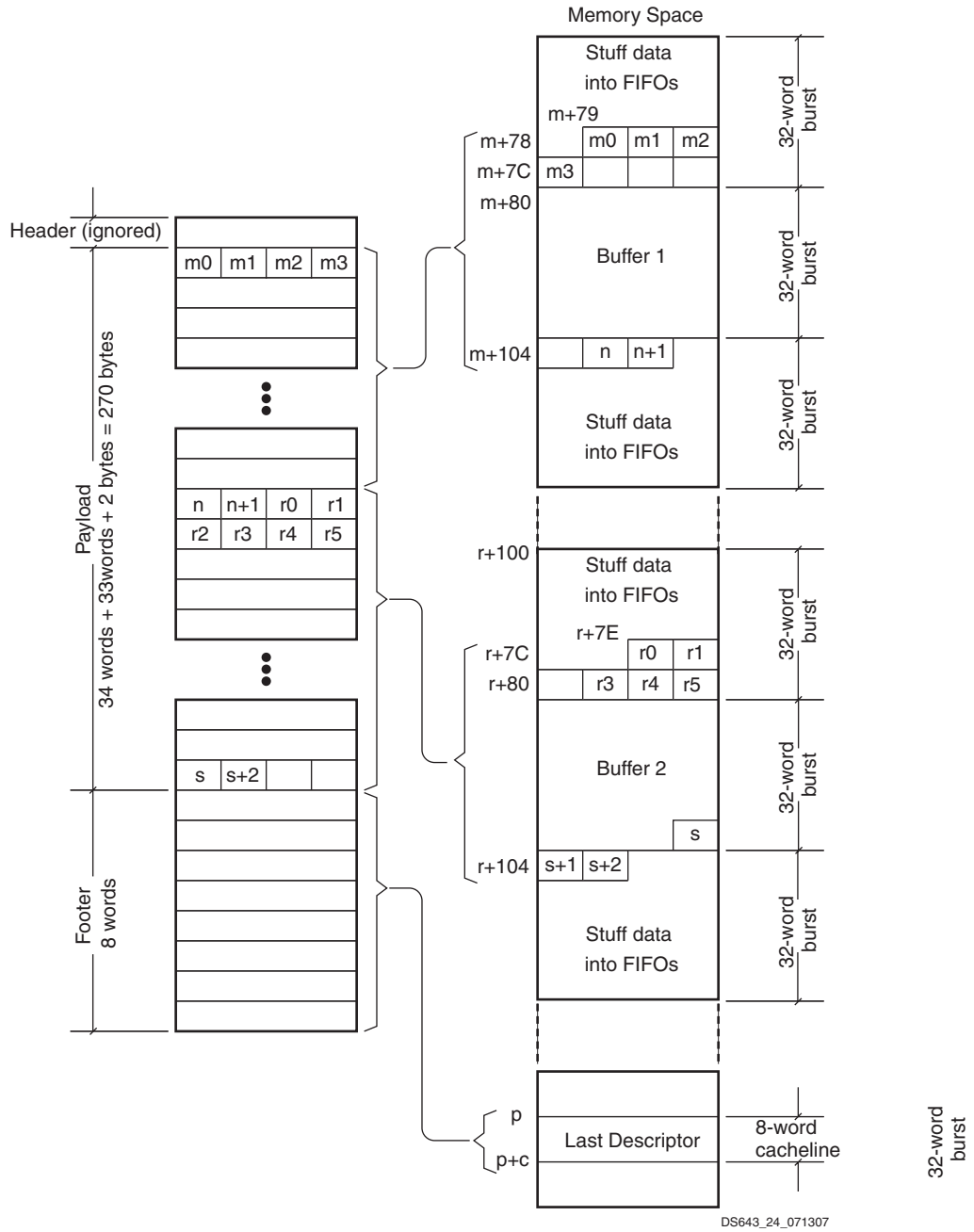


Figure 21: Receive Byte Shift Example

SDMA Interrupts and Errors

The following subsections describe DMA interrupts and errors:

- [DMA Controller Interrupt Description](#)
- [SDMA Error Events](#)
- [SDMA Interrupt On End Event](#)
- [SDMA Interrupt Coalescing and Delay Timer](#)
- [SDMA Engine Reset](#)

DMA Controller Interrupt Description

Each channel can generate one or more events. The Interrupt registers (IRQ_REG) report events for the individual channels; see the following figure.

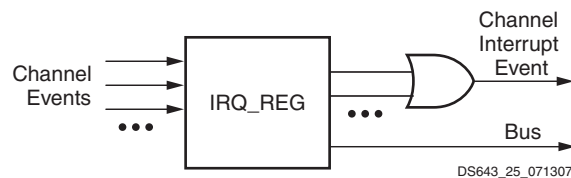


Figure 22: Interrupt Status Register

The IRQ_REG generates a system interrupt when ERRIrq, DlyIrq, or ClscIrq is set and the corresponding enable bit is set in the CHANNEL_CTRL register.

Reading the individual IRQs allows the software to determine which event occurred and for which channel the event was logged. Writing a 1 to the bit position of the event that was logged either clears that event, in the case of the IRQ_REG.ErrIrq interrupt, or decrements the ClscCnt or DlyCnt, depending upon to which bit the data is written. When the IRQ_REG.ClscCnt is zero, the IRQ_REG.ClscIrq is cleared to zero. Likewise, when the IRQ_REG.DlyCnt is zero, the IRQ_REG.DlyIrq is cleared to zero.

SDMA Error Events

An error event occurs when an error is detected during SDMA operations. If an error is detected, IRQ_REG.ErrIrq is set, and if CHANNEL_CTRL.IrqEn = 1 and CHANNEL_CTRL.IrqErrEn = 1, the DMA Controller also generates an interrupt, or increments the Coalescing Event counter. This event can be cleared by writing a 1 to IRQ_REG.ErrIrq.

When a DMA transfer error occurs for a particular channel, that channel is shut down and no more DMA processing occurs for that channel.

SDMA Interrupt On End Event

An Interrupt On-End Event (IOE) occurs when SDMA has completed processing of a buffer descriptor with the IOE bit set in the STS_CTRL_APP0 field or an end of packet (EOP) is received or transmitted. If the DMA has completed operations, the IRQ_REG.CoalIrq is set and, if enabled, an interrupt is also generated. Otherwise, the Coalescing Event counter increments.

SDMA Interrupt Coalescing and Delay Timer

The delay timer is needed because of the interrupt coalescing in the DMA. For example, if the RX coalescing counter is set to 10, an interrupt event is generated for every 10 packets received. If five packets are received on the ethernet and the channel then goes idle (no traffic), the CPU never processes the five packets because no interrupt was generated and this interrupt will happen only when (or if) five more packets arrive.

To avoid this latency, a timer must fire when a packet has been received, some configurable time has elapsed, and there are no more packets received during this time.

The purpose of this timer is to avoid large latencies in the received packet (which is sitting in main memory) from being processed by the CPU when there is non-continuous traffic.

As shown in the following figure, the Clock Divider module uses a 10-bit value, `C_PRESCALAR`, to determine how many LocalLink clock cycles to count before generating a single `Timer_ce` pulse. For a typical LocalLink clock speed of 200MHz and `C_PRESCALAR=1023`, this translates to a 5.12 usec `Timer_ce` period. Therefore, the 8-bit timer is able to count up to a maximum of $256 * 5.12 = 1.3$ msec, before generating an interrupt.

When the Coalescing Counter fires, the delay timer automatically clears.

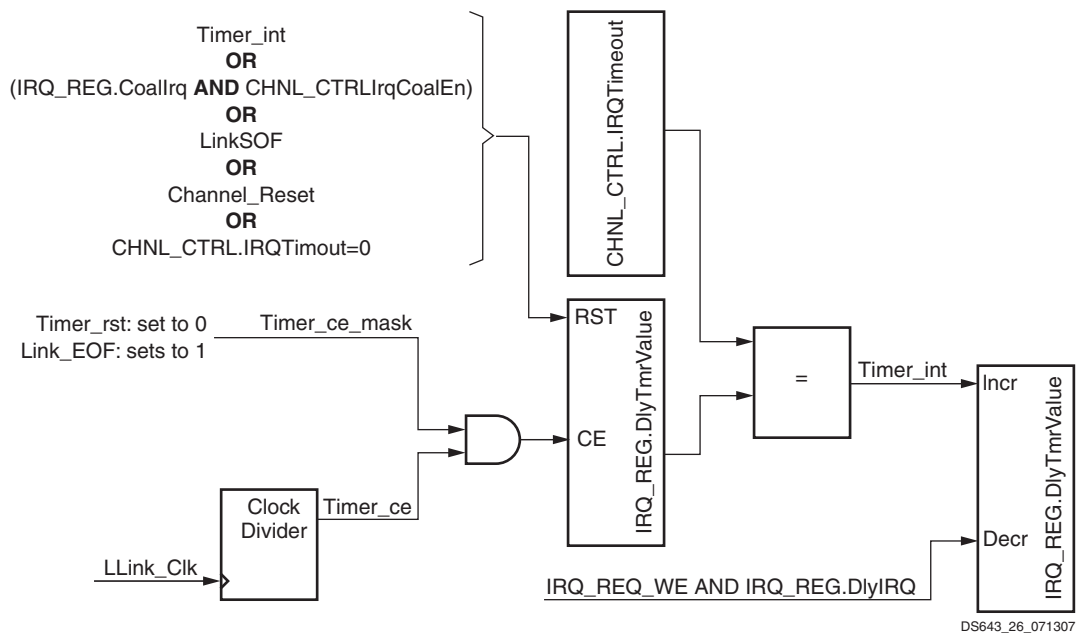


Figure 23: Delay Timer Interrupt Scheme

The Interrupt Coalescing counter is an additional mechanism for interrupt handling. It can relieve the need for the CPU to service an interrupt at the end of every packet. Instead, a pre-loadable number of interrupt events (up to 256) generate a single interrupt to the CPU. The following figure shows the mechanism used for the Tx coalescing counter interrupt generation.

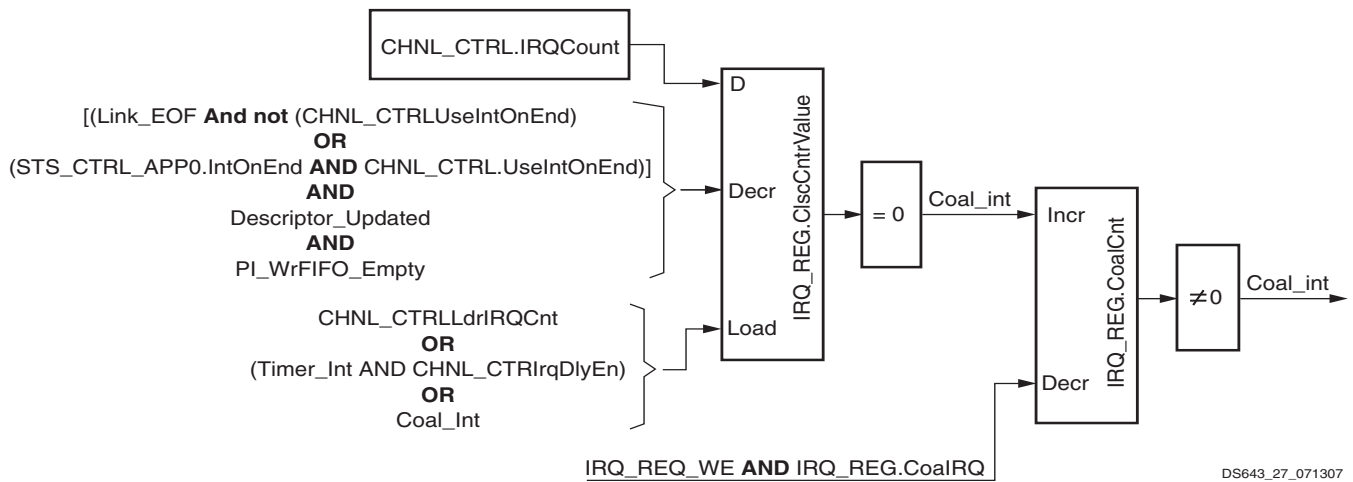


Figure 24: Coalescing Counter Interrupt Scheme

In this example:

- Upon reset, the `CHNL_CTRL.IRQCount` value is used to load the coalescing counter. Subsequently, the register field, `TxIrqCountReg[0:7]`, can be programmed with any 8-bit value.
- On every EOP or `irq-on-end` (selected by `CHNL_CTRL.UseIntOnEnd`), the counter will decrement. When the coalescing counter reaches 0, the DMA increments the 4-bit interrupt counter.
- When the 4-bit interrupt counter is non-zero, it generates an interrupt to the CPU (if the `irq_enable` bit of the respective channel is set).
- When the interrupt is acknowledged (with a control register write value of 1), the 4-bit interrupt counter is decremented.
- The contents of the `TxIrqCountReg[0:7]` register is reloaded when the 4-bit interrupt counter increments.

There is also a means provided for the CPU to force the counter to load the contents of the `TxIrqCountReg[0:7]` register. This is achieved when the CPU writes the `ldIrqCnt` field.

Note: When the delay timer fires, the Coalescing Counter automatically reloads.

SDMA Engine Reset

A capability is included to reset a particular SDMA engine (both RX and TX channels simultaneously) whenever a “lockup” situation arises or an error is detected.

The software reset bit, `DMA_CONTROL_REG.SwReset` allows the software application to reset SDMA. When you write a 1 to `DMA_CONTROL_REG.SwReset`, it initiates the reset sequence for that SDMA. At the same time, the `SDMA_RstOut` output is asserted, synchronous to the LocalLink clock. This output can be used as an external logic reset.

After a soft reset is initiated, software needs to poll the `DMA_CONTROL_REG.SwReset` bit until it is sampled as de-asserted. This indicates that the reset sequence has completed and the pipeline is flushed.

Simultaneously with the `DMA_CONTROL_REG.SwReset` bit being cleared, the `SDMA_RstOut` is automatically de-asserted.

Note: When the DMA engine reset function is used, there is no guarantee that the current descriptor completed correctly. The assumption should be that the descriptor did not complete and it should be restarted again using the normal CPU technique for starting a new DMA operation.

SDMA Transaction Timing

The following subsections describe transaction timing for SDMA.

- [SDMA Descriptor Fetch](#)
- [SDMA Descriptor Update](#)
- [SDMA Transmit Data Read](#)
- [SDMA Receive Data Write](#)
- [SDMA Transmit LocalLink](#)
- [SDMA Receive LocalLink](#)

SDMA Descriptor Fetch

The following figure shows NPI port interface timing for a descriptor fetch. The SDMA uses 8-word, cacheline reads to perform a descriptor fetch. The timing from when the SDMA makes a request of the NPI will depend on arbitration in MPMC. The SDMA monitors `PI_RdFIFO_Empty` to determine when to begin to pop data out of the MPMC read FIFO.

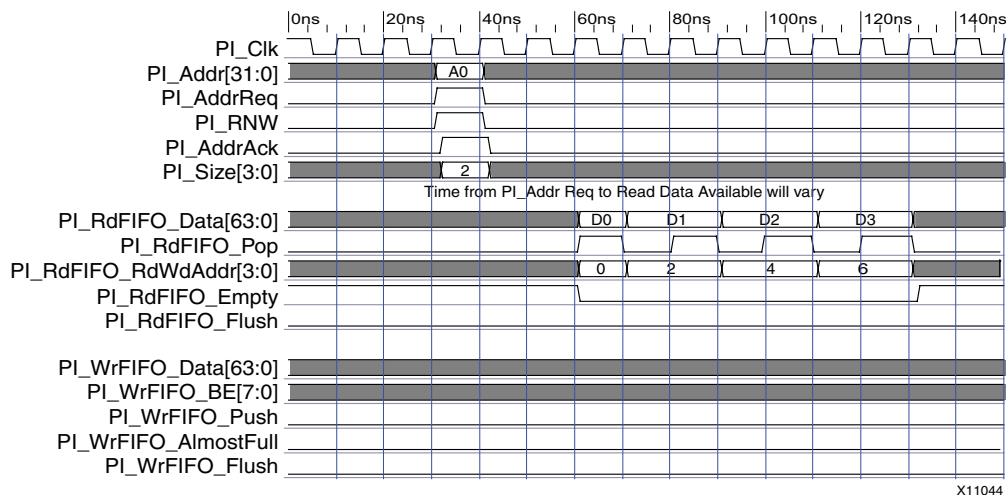


Figure 25: Descriptor Fetch Timing

SDMA Descriptor Update

The following figure shows the MPMC port interface timing for a descriptor fetch. The SDMA uses 8-word cacheline writes to perform a descriptor update. The SDMA will push the data into the MPMC write FIFO if there is space available. After the data has been pushed to the FIFO, the SDMA makes a write request to the MPMC.

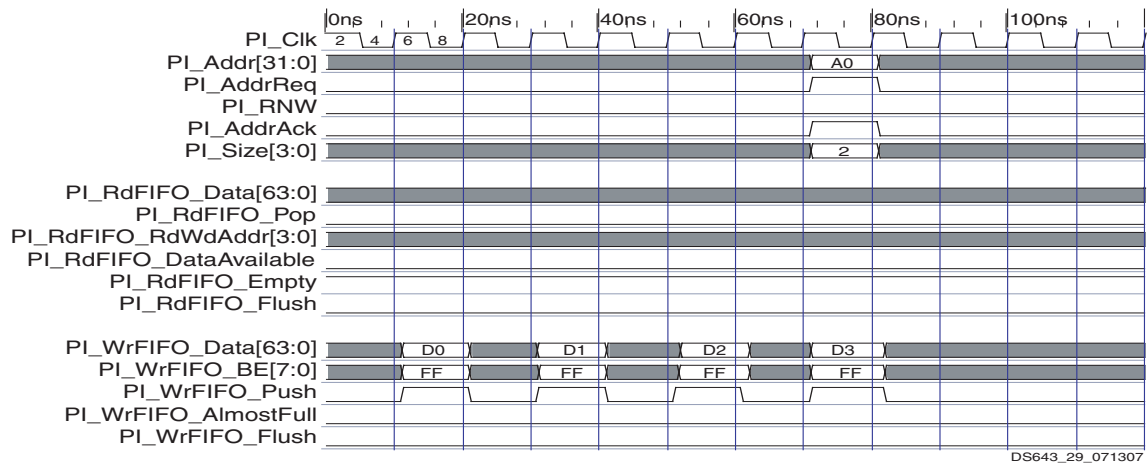


Figure 26: Descriptor Update Timing

SDMA Transmit Data Read

The following figure shows a transmit data read (fetch) from the MPMC. The SDMA uses 32-word (or 16-double word) reads to fetch data for transmitting across LocalLink. Reads always begin at 32-word boundaries. The SDMA uses the data starting with the current buffer address only; additional invalid data fetched due to the 32-word boundary restriction is ignored.

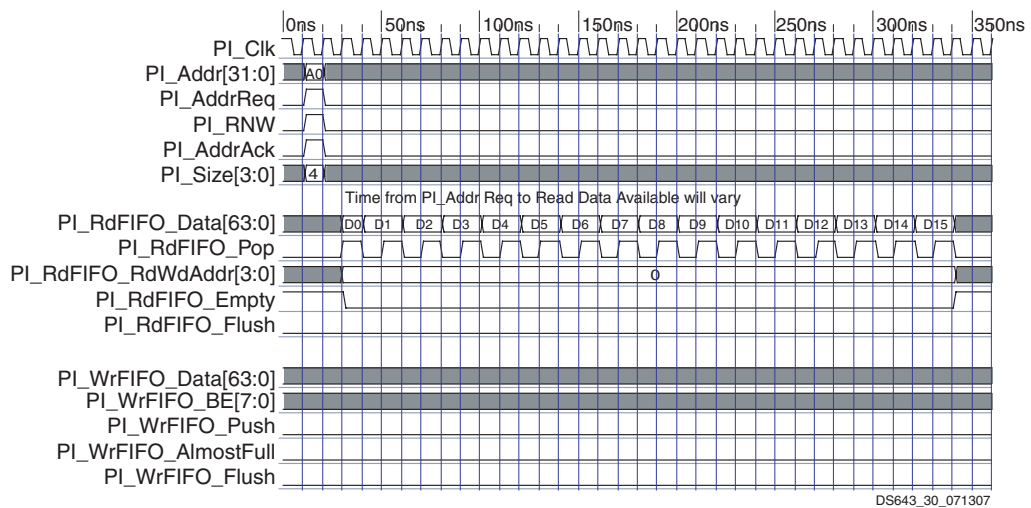


Figure 27: Transmit Data Read

SDMA Receive Data Write

The following figure shows received data being written to the MPMC. The SDMA uses 32-word burst writes to write data to MPMC. The transfers are always 32-word aligned. The PI_WrFIFO_BE bus is used to indicate which bytes of the 32 words are valid. The first two bytes are shown as being invalid and the last byte is also invalid.

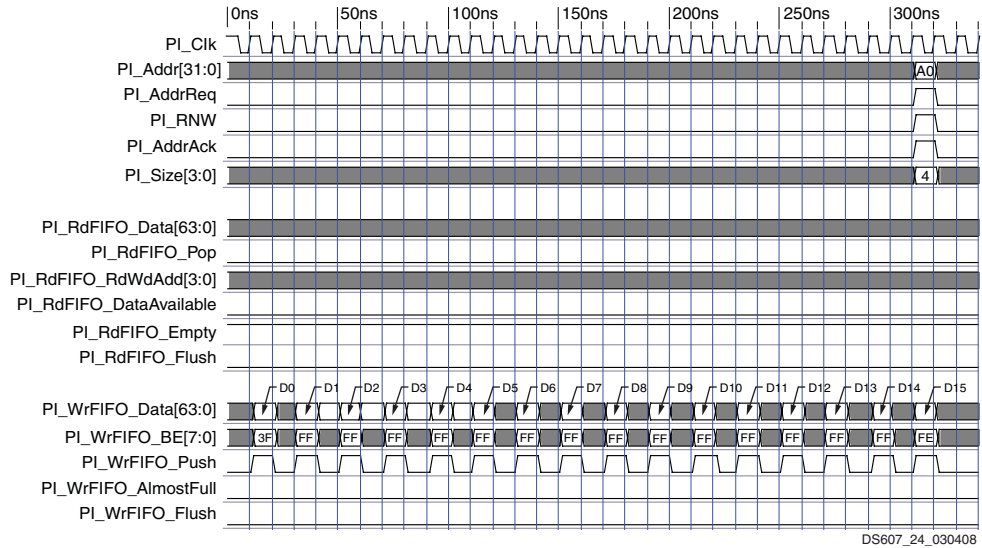


Figure 28: Receive Data Write

SDMA Transmit LocalLink

The following figure shows an example transmit LocalLink transfer of 8 words.

Note: During a transmit the first buffer descriptor is transferred in the header of the LocalLink data stream.

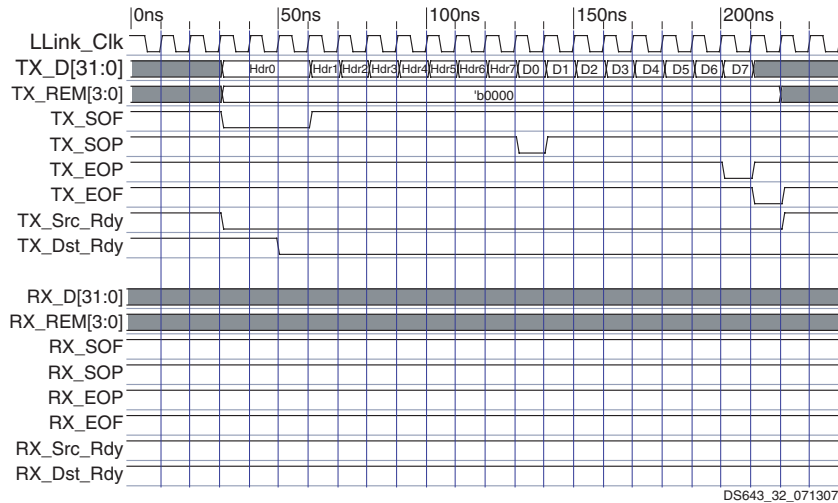


Figure 29: Transmit LocalLink Timing

SDMA Receive LocalLink

An example receive LocalLink transfer of 8 words is shown in the following figure.

Note: During a receive request the last buffer descriptor of a packet is populated with the APP fields of the LocalLink footer.

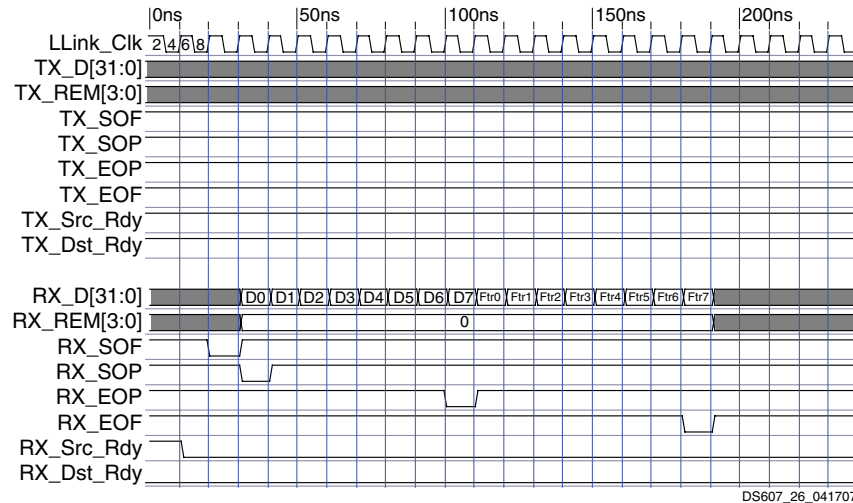


Figure 30: Receive LocalLink Timing

SDMA Registers

The SDMA registers are detailed in the following subsections:

- Next Descriptor Pointer (TX_NXTDESC_PTR and RX_NXT_DESC_PTR) Offsets: 0x00 and 0x20
- Current Buffer Address (TX_CURBUF_ADDR and RX_CURBUF_ADDR) Offsets: 0x04 and 0x24
- Current Buffer Length (TX_CURBUF_LENGTH, RX_CURBUF_LENGTH) Offsets: 0x08 and 0x28
- Current Descriptor Pointer (TX_CURDESC_PTR, RX_CURDESC_PTR) Offsets: 0x0C and 0x2C
- Tail Descriptor Pointer (TX_TAILDESC_PTR and RX_TAILDESC_PTR) Offsets: 0x10 and 0x30
- Channel Control Register (TX_CHNL_CTRL and RX_CHNL_CTRL) Offsets: 0x14 and 0x34
- Interrupt Status Register (TX_IRQ_REG and RX_IRQ_REG) Offsets: 0x18 and 0x38
- Channel Status Register (TX_CHNL_STS and RX_CHNL_STS) Offsets: 0x1C and 0x3C
- DMA Control Register Offset: 0x40

Next Descriptor Pointer (TX_NXTDESC_PTR and RX_NXT_DESC_PTR) Offsets: 0x00 and 0x20

The Next Descriptor Pointers TX_NXTDESC_PTR (Transmit) and RX_NXT_DESC_PTR (Receive), are loaded from the Next Descriptor Pointer field in the current descriptor for the respective channel. This value is kept in the respective SDMA register until the SDMA has completed all DMA transactions within the DMA transfer. After DMA transactions are complete, the current descriptor is complete and the SDMA_COMPLETED bit is set in the respective status register. The current descriptor is written to update the status of the STS_CTRL_APP0 field within the descriptor.

Then the SDMA evaluates if there is a halt condition that occurs when the Current Descriptor Pointer equals the Tail Descriptor Pointer.

If there is no halt condition, the address contained in the Next Descriptor Pointer register is evaluated as follows:

- If a NULL (0x00000000) is contained in the Next Descriptor Pointer register, the SDMA engine stops processing buffer descriptors.
- If the address contained in the Next Descriptor Pointer register is not 8-word aligned or reaches beyond the range of available memory, the SDMA halts processing and sets the SDMA_ERROR bit in the respective status register (TX_CHNL_STATUS or RX_CHNL_STATUS).
- If the Next Descriptor Pointer register contains a valid address, the contents are moved to the respective Current Descriptor Register (TX_CURDESC_PTR or RX_CUR_DESC_PTR).

Then the SDMA begins another DMA transaction. [Table 59, page 131](#) describes the Next Description Pointer (TX_ and RX_) register bits.

Table 59: Next Descriptor Pointer Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0:31	TX_NXTDESC_PTR and RX_NXT_DESC_PTR	Read	0x00000000	8 word aligned pointer to the next buffer descriptor in the chain. If NULL (0x00000000), DMA engine stops processing buffer descriptors.

**Current Buffer Address (TX_CURBUF_ADDR and RX_CURBUF_ADDR)
Offsets: 0x04 and 0x24**

The Current Buffer Address register, one for transmit and one for receive, maintains the contents of the address in memory where the DMA operation is conducted next. This value is originally loaded into the SDMA when the descriptor is read by the SDMA.

When set by the current buffer descriptor, the SDMA periodically transfers this value to an internal Address Counter that then updates the value for each DMA transaction completed.

Upon termination of the transaction, the SDMA overwrites the value of the Current Buffer Address register with the last value of the Address Counter.

This process continues repeatedly until the SDMA has completed the current descriptor. The reason for this mechanism is so the SDMA can maintain multiple temporal channels of DMA at a substantially reduced hardware cost. It is not recommended that software use the Current Buffer Address register to determine SDMA progress because it changes dynamically. The following table describes the Current Buffer Address register bits.

Table 60: Current Buffer Address Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0:31	TX_CURBUF_ADDR and RX_CUR_BUF_ADDR	Read	0x00000000	Address to the current buffer being processed by SDMA.

Current Buffer Length (TX_CURBUF_LENGTH, RX_CURBUF_LENGTH)

Offsets: 0x08 and 0x28

The Current Buffer Length register, one for transmit and one for receive, maintains the contents of the remaining length of the data to be transferred by the SDMA. The value is originally loaded into the SDMA when the descriptor is read by the SDMA. When set by the current descriptor, the SDMA periodically transfers this value to an Internal Length Counter, which then updates the value for each DMA transaction completed. Upon termination of the transaction, the SDMA overwrites the value of the Current Buffer Length register with the last value of the internal length counter. This process continues repeatedly until the SDMA has completed the current descriptor. The following table describes the Current Buffer Length register bits.

Table 61: Current Buffer Length Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0:31	TX_CURBUF_LENGTH and RX_CURBUF_LENGTH	Read	0x00000000	Length in bytes of the current buffer being processed by SDMA.

Current Descriptor Pointer (TX_CURDESC_PTR, RX_CURDESC_PTR)

Offsets: 0x0C and 0x2C

The Current Descriptor Pointer register, one for transmit and one for receive, maintains the pointer to the buffer descriptor that is currently being processed. The value was set either by the CPU when it first initiated a DMA operation, or is copied from the Next Descriptor Pointer register upon completion of the prior descriptor. This value is maintained by the SDMA as a pointer so that the SDMA can update the status and application dependent fields of the descriptor after the buffer descriptor has been fully processed. The following table describes the Current Descriptor Pointer register bits.

Table 62: Current Descriptor Pointer Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0:31	TX_CURDESC_PTR and RX_CURDESC_PTR	Read/Write	0x00000000	An 8-word aligned pointer to the current buffer descriptor being processed by the SDMA.

Tail Descriptor Pointer (TX_TAILDESC_PTR and RX_TAILDESC_PTR)

Offsets: 0x10 and 0x30

The Tail Descriptor Pointer register, one for transmit and one for receive, maintains the pointer to the buffer descriptor chain tail. Tail Pointer Mode is always enabled; consequently, DMA operations will halt when processing of the buffer descriptor pointed to by TAILDESC_PTR is completed. Writing to this register will start DMA operations. The following table describes the Tail Descriptor Pointer register bits.

Table 63: Tail Descriptor Pointer Register Description

Bit(s)	Name	Core Access	Reset Value	Description
0:31	TX_TAILDESC_PTR and RX_TAILDESC_PTR	Read/Write	0x00000000	An 8-word aligned pointer to the tail descriptor. The software application writes to this field to initiate a DMA transfer.

Channel Control Register (TX_CHNL_CTRL and RX_CHNL_CTRL)
Offsets: 0x14 and 0x34

The Channel Control register, one for transmit and one for receive, controls interrupt processing for the particular channel. The following figure illustrates the Channel Control register, and the following table describes the Channel Control register bits.

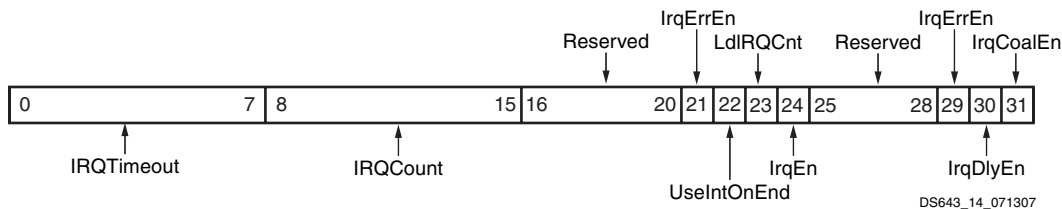


Figure 31: Channel Control Register

Table 64: Channel Control Register

Bit(s)	Name	Core Access	Reset Value	Description
0:7	IRQTimeout	Read/Write	0	Interrupt Delay Time-out Value The maximum amount of time that an unreported packet is required to wait until generating a Delay Interrupt (DlyIRQ) event. (Must remain unchanged for the duration of an DMA operation.)
8:15	IRQCount	Read/Write	00	Interrupt Coalescing Threshold Count Value The number of packets that must be received to generate a Coalescing Interrupt (ClscIrq) event. This value is loaded into the packet threshold counter when LdIrqCnt = 1 and subsequently re-loaded whenever the threshold count is reached.
16:20	Reserved			Reserved - Read as zero.
21	Use1BitCnt	Read/Write	0	Use 1 Bit Wide Counters. <i>Currently Not Used.</i>
22	UseIntOnEnd	Read/Write	0	Use Interrupt On End: Selects between using the interrupt-on-end mechanism or using the EOP mechanism for interrupt coalescing. 1 - Selects the interrupt-on-end mechanism 0 - Selects the EOP mechanism
23	LdIRQCnt	Write	0	Load IRQ Count: Writing a 1 to this field forces the loading of the Interrupt Coalescing counters from the CHANNEL_CTRL.IrqCount[0:7] field. This is a self-clearing field. Read as zero
24	IrqEn	Read/Write	0	Master Interrupt Enable: When set, indicates that the DMA channel is enabled to generate interrupts. This is the master enable for the channel. Individual sources can be enabled and disabled separately.
25:28	Reserved			Reserved - Read as zero
29	IrqErrEn	Read/Write	0	Interrupt on Error Enable: When set, indicates that an interrupt will be generated if an error occurs
30	IrqDlyEn	Read/Write	0	Interrupt on Delay Enable: When set, indicates that an interrupt will be generated when the time-out value is reached.
31	IrqCoalEn	Read/Write	0	Interrupt on Count Enable: When set, indicates that an interrupt will be generated when the interrupt coalescing threshold value is reached.

Interrupt Status Register (TX_IRQ_REG and RX_IRQ_REG)

Offsets: 0x18 and 0x38

The Interrupt Status register, one for transmit and one for receive, indicates interrupt pending and interrupt coalescing count values. This register is used by the software application also to acknowledge pending interrupts by writing a 1 to clear the pending interrupts. The following figure illustrates the Interrupt Status register, and the table that follows describes the Interrupt Status register bits.

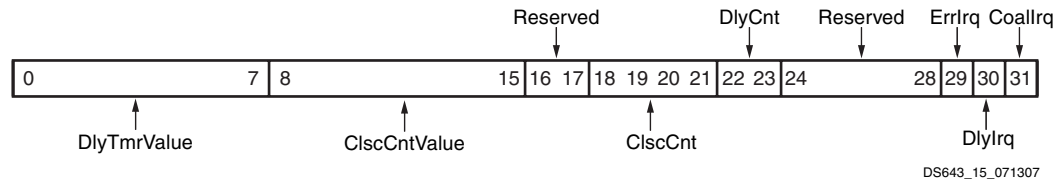


Figure 32: Interrupt Status Register

Table 65: Interrupt Status Register

Bit(s)	Name	Core Access	Reset Value	Description
0:7	DlyTmrValue	Read	0	Delay Timer Value This field contains the real time delay timer value.
8:15	ClscCntrValue	Read	FF	Coalesce Counter Value This field contains the real time coalesce counter value.
16:17	Reserved			Reserved - read as zero.
18:21	ClscCnt	Read	0	Coalesce Interrupt Count Indicates the number of events due to reaching the interrupt coalesce threshold.
22:23	DlyCnt	Read	0	Delay Interrupt Count Indicates the number of events due to reaching the wait bound delay time.
24:28	Reserved			Reserved - read as zero
29	ErrIrq	Read/Write	0	Error Interrupt Event Indicates that an error has occurred. Writing a 1 to this bit clears the interrupt.
30	DlyIrq	Read/Write	0	Delay Interrupt Event Indicates that delay time-out event has occurred. Writing a 1 to this bit clears the interrupt.
31	Coallrq	Read/Write	0	Coalesce Interrupt Event. Indicates that an interrupt event threshold count has been reached. Writing a 1 to this bit clears the interrupt.

Channel Status Register (TX_CHNL_STS and RX_CHNL_STS)
Offsets: 0x1C and 0x3C

The Channel Status register, one for transmit and one for receive, contains status for a particular channel. The following figure illustrates the Channel Status register, and the table that follows describes the Channel Status register bits.

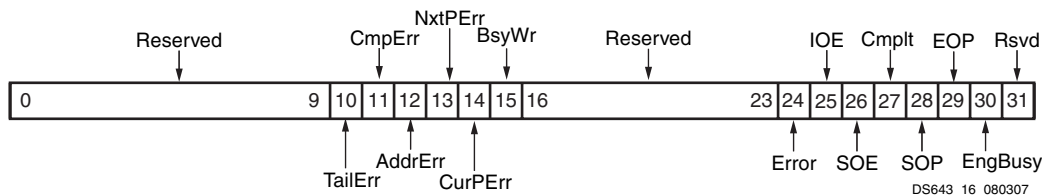


Figure 33: Channel Status Register

Table 66: Channel Status Register

Bit(s)	Name	Core Access	Reset Value	Description
0:9	Reserved			Reserved - Read as zero
10	TailPErr	Read	0	Tail Pointer Error: This bit indicates that Tail Pointer is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
11	CmpErr	Read	0	Complete Error: This bit indicates a descriptor was fetched with the Cmplt = 1 in the STS_CNTRL_APP0 field of the descriptor. This error check is enabled by setting C_COMPLETED_ERR_RX and/or C_COMPLETED_ERR_TX to 1 for the respective channel.
12	AddrErr	Read	0	Address Error: This bit indicates the Current Buffer Address is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
13	NxtPErr	Read	0	Next Descriptor Pointer Error: This bit indicates the Next Descriptor Pointer is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
14	CurPErr	Read	0	Current Descriptor Pointer Error: This bit indicates the Current Descriptor Pointer is NOT a valid address. Valid addresses are between C_PI_BASEADDR and C_PI_HIGHADDR.
15	BsyWr	Read	0	Busy Write Error: This bit indicates the Current Descriptor Pointer register was written to while the DMA Engine was busy.
16:23	Reserved			Reserved - Read as zero
24	Error	Read	0	DMA Error: This bit indicates that an error occurred during DMA operations. This bit is an OR'ing of error bits 10 to 15.
25	IOE	Read	0	Interrupt On End: This bit is a copy of the corresponding bit in the STS_CTRL_APP0 field of the descriptor.
26	SOE	Read	0	Stop On End: This bit is a copy of the corresponding bit in the STS_CTRL_APP0 field of the descriptor.
27	Cmplt	Read	0	Complete: When set indicates that the DMA has transferred all data defined by the current descriptor.
28	SOP	Read	0	Start of Packet (SOP): Start of Packet (SOP): For transmit, the CPU sets this bit in the descriptor to indicate that this is the first descriptor of a packet to be transmitted. For Receive, this bit being set indicates that a valid SOP was received on the LocalLink side.

Table 66: Channel Status Register (Cont'd)

Bit(s)	Name	Core Access	Reset Value	Description
29	EOP	Read	0	End of Packet (EOP): For Transmit, the CPU sets this bit in the descriptor to indicate that this is the last descriptor of a packet to be transmitted. For Receive, this bit being set indicates that a valid EOP was received on the LLink side.
30	EngBusy	Read	0	Engine Busy: When set, indicates that the respective channel is busy with a DMA operation. Generally, software should not write any DMA registers while this bit is set. Reading of registers is allowed.
31	Reserved	Read		Reserved - Read as zero.

DMA Control Register

Offset: 0x40

The DMA Control register controls the DMA operation. The following figure illustrates the DMA Control register, and the table that follows describes the DMA Control register bits.

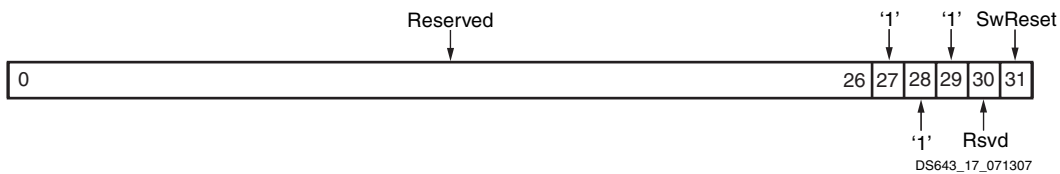


Figure 34: DMA Control Register

Table 67: DMA Control Register

Bit(s)	Name	Core Access	Reset Value	Description
0:26	Reserved		0	Reserved - read as zero
27	Reserved		1	Reserved - read as one
28	Reserved		1	Reserved - read as one
29	Reserved		1	Reserved - read as one
30	Reserved		0	Reserved - read as zero
31	SwReset	Read/Write	0	Software Reset Writing a 1 to this field forces the DMA engine to shutdown and reset itself. After setting this bit, software must poll it until the bit is cleared by the DMA. This indicates that the reset process is done and the pipeline has been flushed 1= Reset DMA - Resets both Rx and Tx Channels 0= Normal Operation (default)

Processor Local Bus Version 4.6 PIM

The Processor Local Bus version 4.6 (PLB v4.6) PIM provides the interconnect from the PLB v4.6 bus to the MPMC.

This sections has the following topics:

- [PLB v4.6 PIM Features](#)
- [PLB v4.6 PIM Overview](#)
- [Supported NPI Transfer Types](#)
- [Supported PLB Master and Bus Widths](#)
- [Configuring PLB v4.6 for Point-To-Point or Shared Bus](#)
- [Configuring PLB v4.6 PIM SUBTYPES](#)
- [Supported Transactions by SUBTYPE](#)

PLB v4.6 PIM Features

The PLB v4.6 PIM supports the following features:

- IBM CoreConnect 32-, 64-, and 128-bit PLB compatibility conforming to PLB v4.6 with Xilinx simplifications. See "[Reference Documents](#)," [page 193](#) for links to more information.
- Access by 32-, 64-, and 128-bit masters. The PLB data and address signals are labelled with big-endian bit and byte ordering as described in [Figure 9, page 80](#).
- Single, data-beat read and write data transfers
- Fixed-burst read and write data transfers
- 4- and 8-word, cacheline read and write transfers
- PLB Point-to-Point (P2P) configuration
- Supports 1:1 and 2:1 (MPMC:PIM) synchronous clock ratios (automatically detected during reset)
- `PIM<Port_Num>_PLB_SAVAlid` on PLB read transfers to minimize bubbles between reads
- 32-bit address offset (the optional address offset is added to the PLB transaction address to compute the physical memory address to be accessed)

PLB v4.6 PIM Overview

PLB v4.6 PIM provides the interconnect from the PLB v4.6 bus to the MPMC.

Over NPI, PLB v4.6 PIM translates a PLB transaction into one or more NPI transactions. These NPI transactions can be byte, half-word, word, 4- and 8-word, cacheline transactions. The NPI transactions type can be 16 word bursts or 32 word bursts also, depending on the value of `C_SPLB<Port_Num>_NATIVE_DWIDTH`.

If the generic `C_SPLB<Port_Num>_NATIVE_DWIDTH` is set to 64, the PIM requests 32 word NPI bursts from MPMC; however, if `C_SPLB<Port_Num>_NATIVE_DWIDTH` is set to 32, the PIM will request 16 word NPI bursts from MPMC. The PLB v4.6 PIM translates PLB transactions into these discrete NPI transactions. Because PLB bursts can have a fixed length ranging from 1 to 16 data beats and can start at different addresses, the PIM must arrange PLB data to fit the available set of NPI transaction types.

The PLB v4.6 PIM is configured by EDK to various subtypes based on the port to which the PIM is connected. The SUBTYPEs are DPLB, IPLB, Single, and PLB:

- The DPLB SUBTYPE is chosen when the PIM is connected to a PowerPC 405 processor DPLB1 port using a point-to-point connection.
- The IPLB SUBTYPE is chosen when the PIM is connected to a PowerPC 405 processor IPLB1 port using a point-to-point connection. When this sub type is chosen, the write FIFO logic in the MPMC data path is also disabled ($C_PI<Port_Num>_WR_FIFO_TYPE = DISABLED$).
- The Single SUBTYPE is chosen when the PIM is connected to a PLB bus where all masters are not burst capable. This is primarily selected in MicroBlaze systems where the PIM is connected to a bus with MicroBlaze IPLB and DPLB ports.
- The PLB SUBTYPE is chosen in all other cases. This PLB PIM supports the full set of PLB transactions.

You can find information regarding the PIM SUBTYPEs in "Configuring PLB v4.6 PIM SUBTYPEs," page 140.

Supported NPI Transfer Types

Based on the setting of $C_SPLB<Port_Num>_NATIVE_DWIDTH$, the PLB v4.6 PIM can generate only certain types of NPI transfers, as shown in the following table.

Table 68: Supported Transactions based on PIM Subtype and Native Width

C_SPLB<Port_Num>_NATIVE_DWIDTH=32						C_SPLB<Port_Num>_NATIVE_DWIDTH=64					
Transaction Type		C_PIM<Port_Num>_SUBTYPE				Transaction Type		C_PIM<Port_Num>_SUBTYPE			
		PLB	DPLB	IPLB	Single			PLB	DPLB ⁽¹⁾	IPLB ⁽¹⁾	Single
		Supported						Supported			
Single	Read	Y	N	N	Y	Single	Read	Y	Y	N	Y
	Write	Y	N	N	Y		Write	Y	Y	N	Y
4-wd Cacheline	Read	Y	N	N	N	4-wd Cacheline	Read	Y	N	Y	N
	Write	Y	N	N	N		Write	Y	N	N	N
8-wd Cacheline	Read	Y	N	N	N	8-wd Cacheline	Read	Y	Y	Y	N
	Write	Y	N	N	N		Write	Y	Y	N	N
16-wd Burst	Read	Y	N	N	N	16-wd Burst	Read	N	N	N	N
	Write	Y	N	N	N		Write	N	N	N	N
32-wd Burst	Read	N	N	N	N	32-wd Burst	Read	Y	N	N	N
	Write	N	N	N	N		Write	Y	N	N	N

1. Point-to-point configuration is required for the DPLB and IPLB subtypes where the $C_SPLB<Port_Num>_NATIVE_DWIDTH=64$.

Supported PLB Master and Bus Widths

The following table shows the supported PLB master and bus widths. A 32-bit `NATIVE_DWIDTH` PIM can only be used with a PLB bus where all masters and slaves on the bus are 32 bits. The 32-bit `NATIVE_DWIDTH` PIM is intended for Spartan-3 based systems.

Table 69: Supported PLB Master and Bus Widths

C_SPLB<Port_Num>_NATIVE_DWIDTH=64			C_SPLB<Port_Num>_NATIVE_DWIDTH=32		
C_SPLB<Port_Num>_DWIDTH	C_SPLB<Port_Num>_SMALLEST_MASTER	Supported	C_SPLB<Port_Num>_DWIDTH	C_SPLB<Port_Num>_SMALLEST_MASTE	Supported
128	128	Y	128	128	N
128	64	Y	128	64	N
128	32	Y	128	32	N
64	64	Y	64	64	N
64	32	Y	64	32	N
32	32	N	32	32	Y

To reduce the number of cycles between read transactions, the PLB v4.6 PIM can accept PLB read transactions when `SPLB<Port_Num>_PLB_SAVValid` is asserted. This allows a subsequent MPMC request to be asserted after the current request has been acknowledged.

Configuring PLB v4.6 for Point-To-Point or Shared Bus

The PLB v4.6 PIM can be configured as either a Point-To-Point (P2P) configuration in which there is only one PLB master communication with the PLB v4.6 PIM or a Shared Bus configuration:

- In P2P configuration, the PLB v4.6 PIM responds to all addresses regardless of the `C_PIM<Port_Num>_HIGHADDR` and `C_PIM<Port_Num>_BASEADDR` parameter values. The PLB address is decoded when the PLB v4.6 PIM is operating in a shared bus configuration only.
- In the Shared Bus configuration, the PLB v4.6 PIM can transfer data from up to 16 masters to MPMC. An extra cycle of latency is incurred when the PLB v4.6 PIM operates on a shared bus because PLB signals to the PLB v4.6 PIM are registered to improve timing.

During memory initialization and calibration, the PIM will assert `SPLB<Port_Num>_Sl_Wait` or `SPLB<Port_Num>_Sl_Rearbitrate` to hold off any PLB transactions until the memory is ready to process transactions.

Configuring PLB v4.6 PIM SUBTYPES

You can configure the PLB v4.6 PIM with various SUBTYPES to optimize it for a set of supported transactions. In most cases the tools will choose the appropriate PLB v4.6 PIM SUBTYPE automatically, based on the connectivity of the system. The PLB v4.6 PIM SUBTYPES are:

- 64-Bit Burst PIM
 - Single word Read and Write transactions
 - 4-word and 8-word, cacheline Read and Write transactions
 - Fixed length burst transactions
- 32-Bit Burst PIM
 - Single word Read and Write transactions
 - 4-word and 8-word, cacheline Read and Write transactions
 - Fixed length burst transactions
- 64-Bit Single PIM
 - Single word Read and Write transactions (reduced size when burst support is not needed)
- 32-Bit Single PIM
 - Single word Read and Write transactions (reduced size when burst support is not needed)
- DPLB
 - Single word Read and Write transactions; and 8-word, cacheline Read and Write transactions
- IPLB
 - 4-word and 8-word, cacheline Read transactions

Caution! When choosing a SUBTYPE, be sure the SUBTYPE supports all the PLB transaction types that will be issued to the PLBv4.6 PIM. When an unsupported transaction is issued on the PLB bus, it is possible to cause the logic in the PLBv4.6 PIM to hang, which would require a system reset to recover.

Supported Transactions by SUBTYPE

The supported transactions for each of the SUBTYPES are listed in the following table.

Table 70: Supported Transactions

SPLB_PLB_SIZE (0:3)	SPLB_PLB_SIZE (0:3)	Description	Read/Write	C_PIM<Port_Num>_SUBTYPE			
				PLB	DPLB	IPLB	Single
SPLB_PLB_SIZE (0:3)	0x0	Single Transactions	Read	Y	Y	N	Y
			Write	Y	Y	N	Y
	0x1	4-wd Cacheline	Read	Y	N	Y	N
			Write	Y	N	Y	N
	0x2	8-wd Cacheline	Read	Y	Y	Y	N
			Write	Y	Y	Y	N
	0x3	16-wd Cacheline	Read	N	N	N	N
			Write	N	N	N	N
	0x4	Reserved	Read	N	N	N	N
			Write	N	N	N	N

Table 70: Supported Transactions (Cont'd)

SPLB_PLB_SIZE (0:3)	0x5	Reserved	Read	N	N	N	N
			Write	N	N	N	N
	0x6	Reserved	Read	N	N	N	N
			Write	N	N	N	N
	0x7	Reserved	Read	N	N	N	N
			Write	N	N	N	N
	0x8	Byte Bursts	Read	N	N	N	N
			Write	N	N	N	N
	0x9	Half Word Bursts	Read	N	N	N	N
			Write	N	N	N	N
	0xA	Word Bursts	Read	Y	N	N	N
			Write	Y	N	N	N
	0xB	Double Word Bursts	Read	Y	N	N	N
			Write	Y	N	N	N
	0xC	Quad Word Bursts	Read	Y	N	N	N
			Write	Y	N	N	N
	0xD	Octal Word Bursts	Read	N	N	N	N
			Write	N	N	N	N
0xE	Reserved	Read	N	N	N	N	
		Write	N	N	N	N	
0xF	Reserved	Read	N	N	N	N	
		Write	N	N	N	N	
SPLB_PLB_TYPE (0:2)	SPLB_PLB_TYPE (0:2)	Description	Read/Write	C_PIM<Port_Num>_SUBTYPE			
				PLB	DPLB	IPLB	Single
	000b	Memory Transfer	Read	Y	Y	Y	Y
			Write	Y	Y	Y	Y
001b-111b	N/A	Read	N	N	N	N	
Indeterminate Burst			Read	N	N	N	N
			Write	N	N	N	N
Burst request of 2-16 databeats			Read	Y	N	N	N
			Write	Y	N	N	N
Burst request of > 16 databeats			Read	N	N	N	N
			Write	N	N	N	N

PowerPC 440 Memory Controller PIM

The PowerPC 440 Memory Controller (PPC440MC) PIM connects MPMC directly to the memory interface port of the PowerPC 440 block in Virtex-5 FXT devices.

Note: The `ppc440mc_ddr2` IP core should be used instead of MPMC whenever possible. The `ppc440mc_ddr2` IP core is optimized as a single port DDR2 memory for the PowerPC440 processor memory interface. The `ppc440mc_ddr2` IP core offers lower latency and higher FMAX than MPMC. MPMC should only be used when SDRAM/DDR memory support or multiple memory ports are needed.

This topic contains the following subsections:

- [PPC440MC Features](#)
- [Supported PPC440MC Interface Configuration](#)
- [PPC440MC Overview](#)
- [PPC440MC Design Implementation](#)
- [PPC440MC Parameter and Port Dependencies](#)
- [PPC440MC Burstwidth and Burstlength by Memory Type/Width Dependencies](#)

PPC440MC Features

- Supports The Virtex-5, V5FXT, memory interface.
- Supports 32-bit and 64-bit NPI interfaces using 32-bit and 64-bit PPC440MC data widths.
- Runs with 1:1 clocking with NPI and MIB.
- Provides parameterized burst sizes of 2, 4, and 8.
- Supports read data latency of 0, 1, and 2 clocks.
- Can operate at a 1:1, 1:2, 1:3, or 1:4 clock ratio with respect to the PowerPC crossbar interconnect clock (`CPMINTERCONNECTCLK`)

Supported PPC440MC Interface Configuration

The PPC440MC PIM requires the following configuration settings in the `MI_CONTROL` register of the PowerPC 440 Block.

Table 71: Required PPC440 Block MI_CONTROL Register Settings

Bit(s)	Name	Required Value Setting	Description
0	enable	1	Turn on the PPC440MC interface.
1	Rowconflictholdenable	0	PPC440MC PIM does not support row/bank management.
2	Bankconflictholdenable	0	PPC440MC PIM does not support row/bank management.
3	Directionconflictholdenable	0	PPC440MC PIM does not support row/bank management.
4:5	Autoholdduration	00	PPC440MC PIM only supports this autohold mode.
6	2:3 Clock Ratio mode	0	PPC440MC PIM only supports integer clock ratios. It does not support a clock ratio of 2:3 with respect to <code>CPMINTERCONNECTCLK</code> .
7	overlaprddwr	0	Overlapped transfers and QDR mode are not supported.
8:9	Burstwidth	Depends on memory type and width	See " PPC440MC Burstwidth and Burstlength by Memory Type/Width Dependencies ," page 144 for supported values. The range of allowable values depend on memory type and width.

Table 71: Required PPC440 Block MI_CONTROL Register Settings (Cont'd)

Bit(s)	Name	Required Value Setting	Description
10:11	Burstlength	Depends on memory type and width	See "PPC440MC Burstwidth and Burstlength by Memory Type/Width Dependencies," page 144 for supported values. The range of allowable values depend on memory type and width.
12:15	Write Data Delay (WDD)	0000	PPC440MC PIM supports this WDD mode only.
16	RMW	0	PPC440MC PIM supports this setting only.

All bits not shown in Table 71, page 142 can normally be set by the users as described in the "Embedded Processor Block" of the *Virtex-5 FPGAs Reference Guide* (a link to this document is provided in "Reference Documents," page 193.)

PPC440MC Overview

The PPC440MC PIM is the interface between the NPI and the PowerPC 440 processor memory interface block, as shown in the following figure.

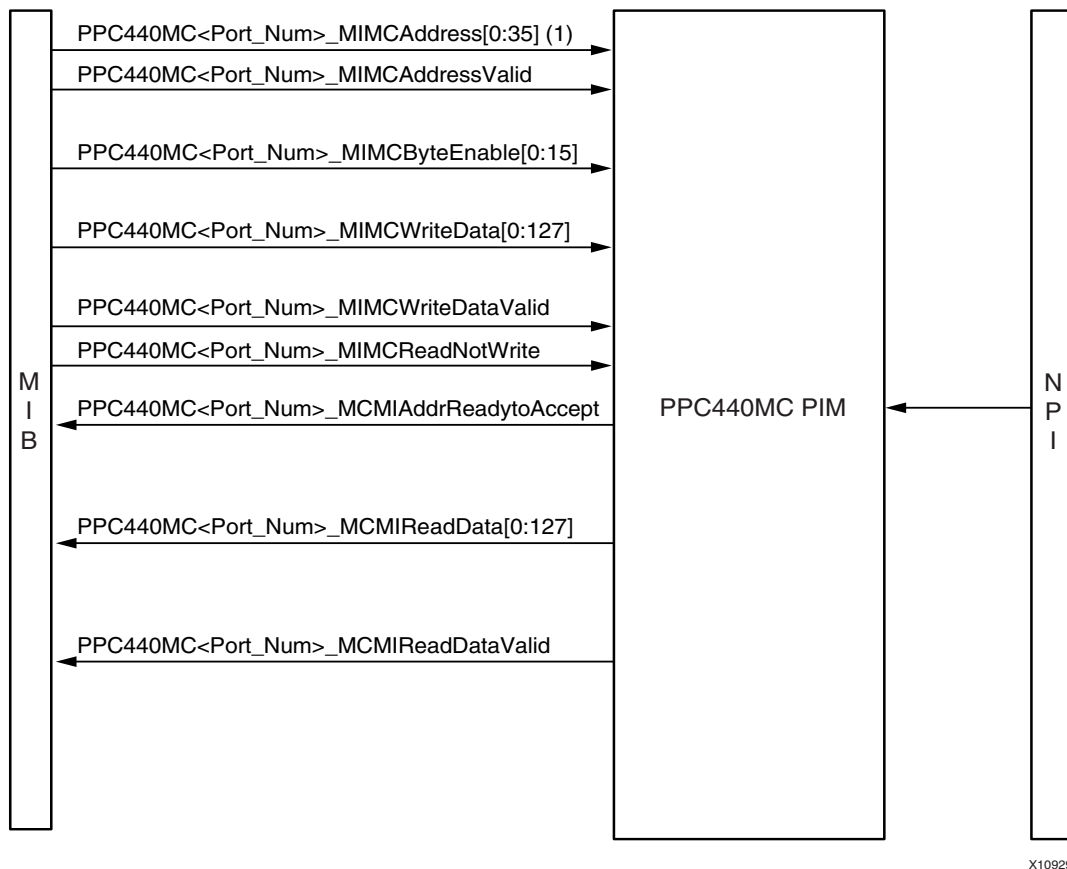


Figure 35: PPC440MC Block Diagram

X10929

The functional units within the PPC440MC PIM are:

- **Addr Path** - generates address, address request (MPMC_PIM_AddrReq), readnotwrite and size (MPMC_PIM_Size) signals. The Addr Path also generates addressreadytoaccept information for PPC440MC.
- **Write Data Path** - generates writedata, writedatavalid (push) and byteenable.
- **Read Data Path** - generates readdata and readdatavalid for PPC440MC. The Read Data Path also generates the RdFIFO_POP and RdModWr.

PPC440MC Design Implementation

The PPC440MC PIM is available in the Virtex-5 FXT family FPGAs only.

PPC440MC Parameter and Port Dependencies

Dependencies exist between the PPC440MC PIM core design parameters and the I/O signals. In addition, when certain features are parameterized out of the design, the related logic will no longer be a part of the design. The unused input signals and related output signals are set to a specified value.

PPC440MC Burstwidth and Burstlength by Memory Type/Width Dependencies

The following table shows the required setting for Burstwidth and the allowable settings for Burstlength in relation to memory type and width.

Table 72: Burstwidth and Burstlength by Memory Type and Width

Memory Type	Memory Width	Required Setting for Burstwidth Field of PPC440 MI_Control Register (1)	Allowable Settings for Burstlength Field of PPC440 MI_Control Register
SDRAM	8	32	4, 8
SDRAM	16	32	4, 8
SDRAM	32	32	4, 8
SDRAM	64	64	2, 4, 8
DDR/DDR2	8	32	4, 8
DDR/DDR2	16	32	4, 8
DDR/DDR2	32	64	2, 4, 8
DDR/DDR2	64	64	4, 8

1. The MPMC data path architecture does not support a native 128 bit NPI interface for the PPC440MC PIM; therefore, the maximum data path width between MPMC and the PowerPC 440 block MC interface is limited to 64 bits. This might limit the performance gain seen between 32-bit and 64-bit DDR/DDR2. The `ppc440mc_ddr2` IP core does not have this limitation and supports a full 128-bit data path with a 64-bit DDR2 memory.

Video Frame Buffer Controller PIM

The Video Frame Buffer Controller (VFBC) allows a user IP to read and write data in two dimensional (2D) sets regardless of the size or the organization of external memory transactions. The VFBC can be used in video applications where hardware control of 2D data is needed to achieve real time operation.

Typical video applications are: motion estimation, video scaling, on-screen displays, and video capture used in video surveillance, video conferencing and video broadcast.

The following subsections describe:

- [VFBC Features](#)
- [VFBC Overview](#)
- [VFBC Command Interface](#)
- [VFBC Write Data Interface](#)
- [VFBC Read Data Interface](#)
- [VFBC Transfer Examples](#)
- [VFBC Synthesis Considerations](#)
- [VFBC Timing Constraints](#)

VFBC Features

- 2D data transfers (32,640 bytes x 16,777,216 lines maximum and two 32-bit words minimum.)
- Asynchronous FIFO command interface.
- Separate asynchronous FIFO write and read data interfaces.
- Configurable 32- or 64-bit NPI data width.
- Independently configurable write and read data widths of 8-, 16-, 32-, or 64-bit.
- Configurable FIFO depths.
- Configurable almost full and almost empty flags.
- Independent write, read, and command FIFO resets.
- Flushable data FIFOs.

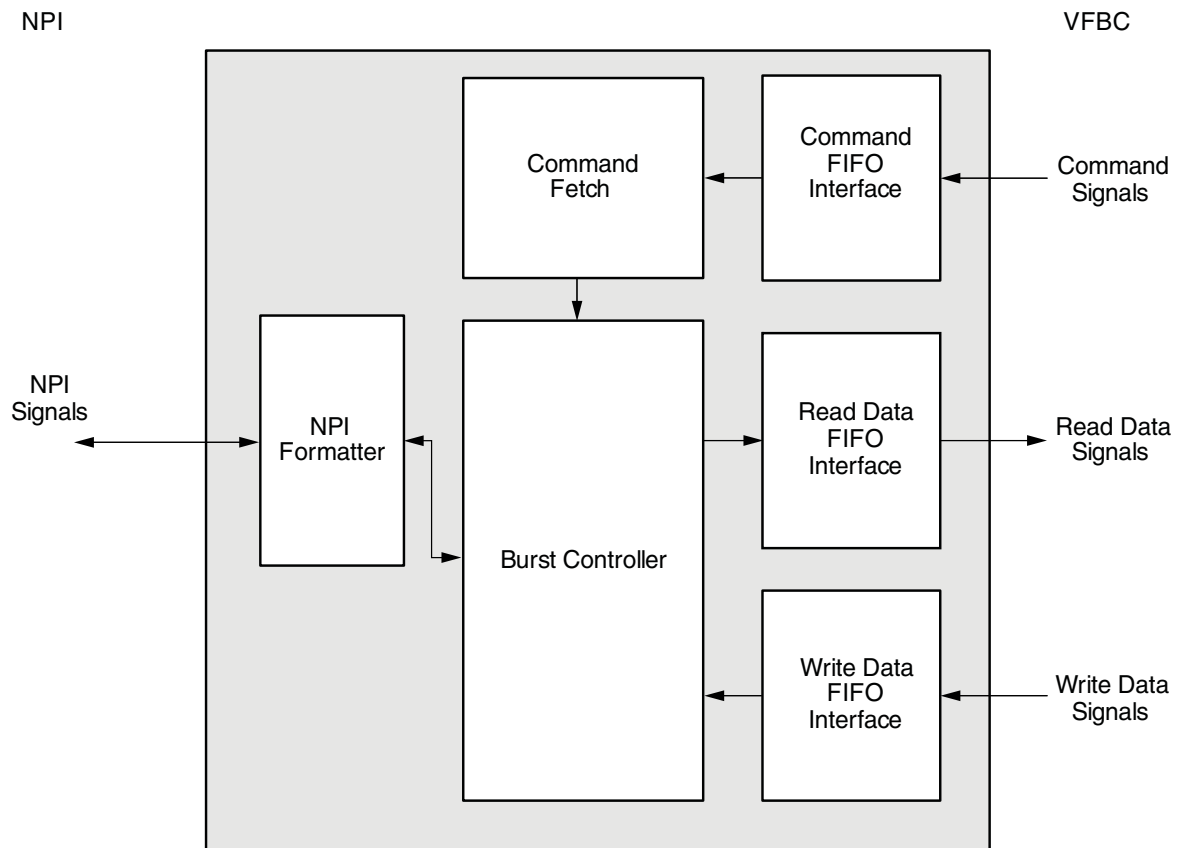
VFBC Overview

The VFBC is a connection layer between video clients and the MPMC. Because video systems are inherently heterogeneous (given the wide variety of video formats and transmission), the VFBC provides key features to address the typical video system.

The VFBC also includes separate Asynchronous FIFO interfaces for command input, write data input, and read data output. This is useful to decouple the video IP from the memory clock domain.

The following diagram shows the VFBC interfaces. The interfaces are discussed in more detail below. Refer to "[VFBC PIM I/O Signals](#)," [page 25](#) for more information on the individual signals for each VFBC interface.

The following figure shows a block diagram of the VFBC interface.



X10910

Figure 36: VFBC High-Level Block Diagram

Data transfers to and from the VFBC data FIFOs are controlled by the command interface. Commands are written into the command interface FIFO in 4-word packets. This four word packet controls the direction (read or write) and the 2D size of the transfer, and also includes the following information: start address of the 2D transfer, X-size in bytes, Y-size in lines, and the width (Stride) of the video frame.

The VFBC data and address signals are labeled with little-endian bit/byte ordering as described in [Figure 8, page 78](#). The Least Significant Bit (LSB) of a 32-bit word is bit zero.

VFBC Command Interface

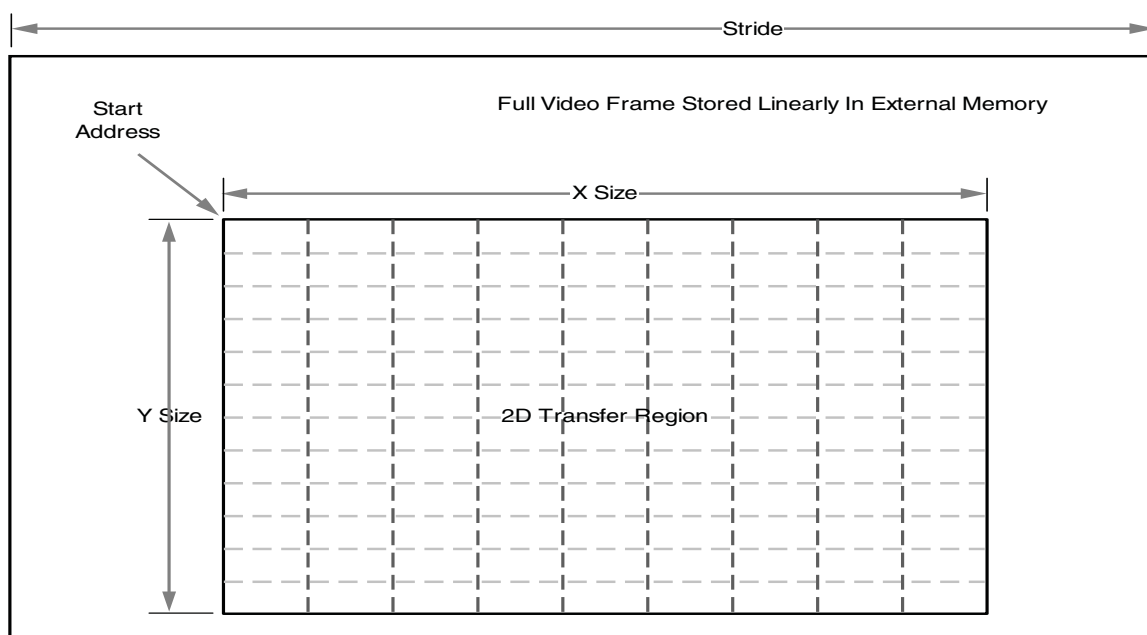
The command interface is implemented as an asynchronous FIFO. Commands are written into the command interface FIFO in 4-word packets. Each command packet word is pushed onto the command FIFO during the clock cycle the `VFBC<Port_Num>_Cmd_Write` signal is active. It is not necessary to write the command words during consecutive clock cycles; VFBC acts on the commands after the last command word is written. The command packets can be written at the same time as data transfers to the data interface FIFOs. [Table 73, page 147](#) shows the command packet data structure.

Table 73: Command Packet Data Structure

Command Packet							
Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X Size ⁽¹⁾	31 Write_NotRead	30:0 Start Address ⁽¹⁾	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride ⁽¹⁾

1. The X Size, Start Address and Stride must be aligned to a 32-word boundary. These values must be a multiple of 128 bytes and require that bits [6:0] be '0'.

The VFBC divides each 2D transfer into 32-word transfers for the MPMC. The following diagram shows a video frame stored linearly in external memory. The frame contains a rectangular region of interest to be transferred by the VFBC.



X10911

Figure 37: 2D Transfers

The first word (Command Word 0, bits 14:0) includes the *X Size* of the transfer, which is the number of consecutive linear bytes of the transaction per line. The second word (Command Word 1, bits 31:0) includes the direction of the transfer and the start address. Bit 31 is, or *Write_NotRead*, denotes a write transaction if high and a read transaction if low. Bits 30:0 are the physical memory byte start address, which is the start address of the transfer.

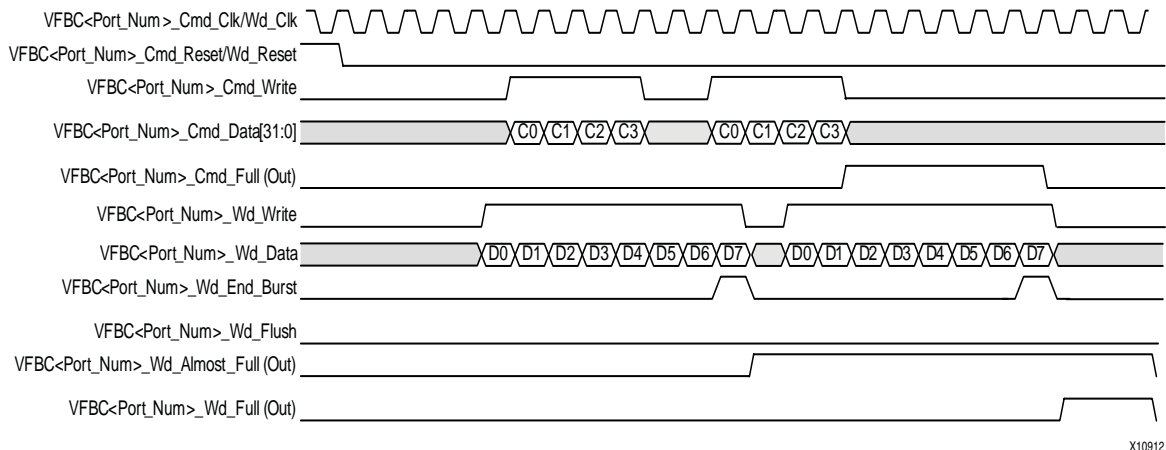
The third word (Command Word 2, bits 23:0) includes the *Y Size* of the transfer, which is the number of lines of the transfer minus one. Figure 37 shows the *Y Size* as 12 lines. The value set in bits 23:0 of the third word must be 0x0000000b for this transfer.

The fourth word (Command Word 3, bits 23:0) includes the *Stride* of the transfer, which is the number of bytes to skip between the start of each line of the transfer. This is the line length (in bytes) of the 2D storage in external memory.

VFBC Write Data Interface

The write data interface is an asynchronous FIFO. The FIFO depth, data width, and the almost full flag are configurable. The data width can be configured as 8-, 16-, 32-, or 64-bits. Data is pushed onto the FIFO during the same clock cycle as when the VFBC<Port_Num>_Wd_Write signal is active.

- The VFBC<Port_Num>_Wd_Flush signal flushes all data currently in the FIFO but keeps the current write command active in the command FIFO. Asserting the FIFO Flush will return the internal read/write FIFO pointers to zero.
- The VFBC<Port_Num>_Wd_Reset signal flushes data in the FIFO and also flushes the write command from the command FIFO.
- The VFBC<Port_Num>_Wd_End_Burst signal is used only when the transfer is not a multiple of the burst size. If the transfer ends on a boundary that is not 32-word aligned, this signal must be asserted high during the last word transferred. The following figure shows a typical VFBC write operation and provides an example of VFBC Write Timing. The actions of the Write data interface are enumerated below the figure.



X10912

Figure 38: VFBC Write Timing

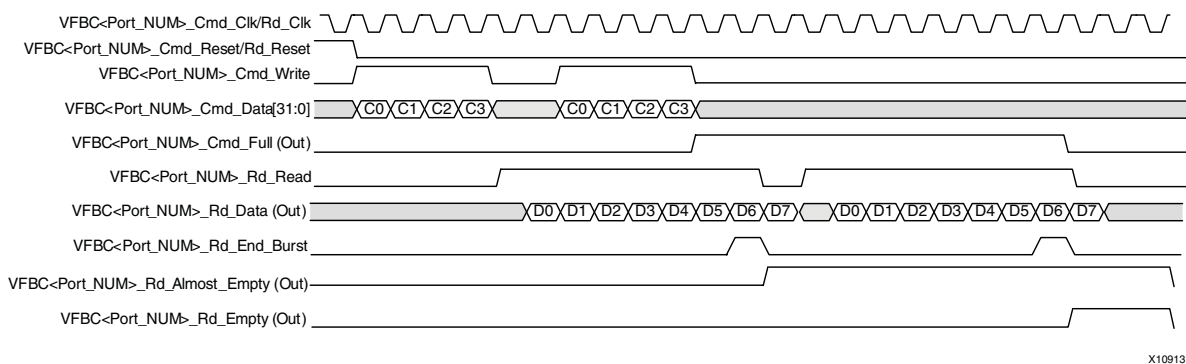
1. Bit 31 in command word C1 must be set to 1 for a write operation.
2. As shown in Figure 38, the command words C0, C1, C2, and C3 are being written during a data write. The command words can be written before, during, or after the data is written.
3. The figure also shows one cycle between VFBC<Port_Num>_Wd_End_Burst and the next burst, although zero to any number of cycles can exist between burst end and the next burst. The number of cycles between bursts is controlled by the VFBC<Port_Num>_Wd_Write signal. This is the FIFO data push signal asserted by the VFBC client.
4. If the VFBC<Port_Num>_Wd_End_Burst signal is used, it must be asserted during the same cycle as the last valid data write and can be on D0-D7 cycles.
5. The VFBC<Port_Num>_Wd_End_Burst signal is optional in this diagram and could be set low always. This signal needs to be used only if the transfer does not end on a 32-word boundary.
6. The VFBC can accept a data transfer on every clock cycle where the VFBC Write FIFO is not Full and that the MPMC memory interface throughput can accommodate the data rate of the VFBC client.

VFBC Read Data Interface

The read data interface is an asynchronous FIFO with a configurable depth, data width, and almost empty flag. The data width can be configured as 8-, 16-, 32-, or 64-bits.

Data is popped off of the FIFO during the same clock cycle as when the VFBC<Port_Num>_Rd_Read signal is active. The VFBC<Port_Num>_Rd_Flush signal flushes the data that is in the FIFO but keeps the current read command active in the command FIFO. Asserting the FIFO Flush will return the internal read/write FIFO pointers to zero. The VFBC<Port_Num>_Rd_Reset signal is used to flush the data in the FIFO and also flush the read command from the command FIFO. The VFBC<Port_Num>_Rd_End_Burst signal is used only when the transfer is not a multiple of the burst size. If the transfer ends on a boundary that is not 32-word aligned, this signal must be asserted high during the last word transferred.

The following figure shows a typical Read operation. Notes on the operation are listed after the figure.



X10913

Figure 39: VFBC Read Timing

Notes:

1. Bit 31 in Command Word C1 must be set to zero for this to be a Read operation.
2. The Command Words C0, C1, C2, and C3 must be written before the data can be read from the data FIFO.
3. The Command Words can be written during a read operation for the next Read operation.
4. Figure 39 shows one cycle between VFBC<Port_Num>_Rd_End_Burst and the next burst; although, zero to any number of cycles can exist between read transactions.
5. If the VFBC<Port_Num>_Rd_End_Burst signal is used, it must be asserted during the same cycle as the last valid data read and can be on D0-D7 cycles.
6. The VFBC<Port_Num>_Rd_End_Burst signal is optional in this diagram and could be always set low. VFBC<Port_Num>_Rd_End_Burst needs to be used only if the transfer does not end on a 32-word boundary.
7. The VFBC can accept a data transfer on every clock cycle given that VFBC Read FIFO is not Empty and that the MPMC memory interface throughput can accommodate the data rate of the VFBC client.

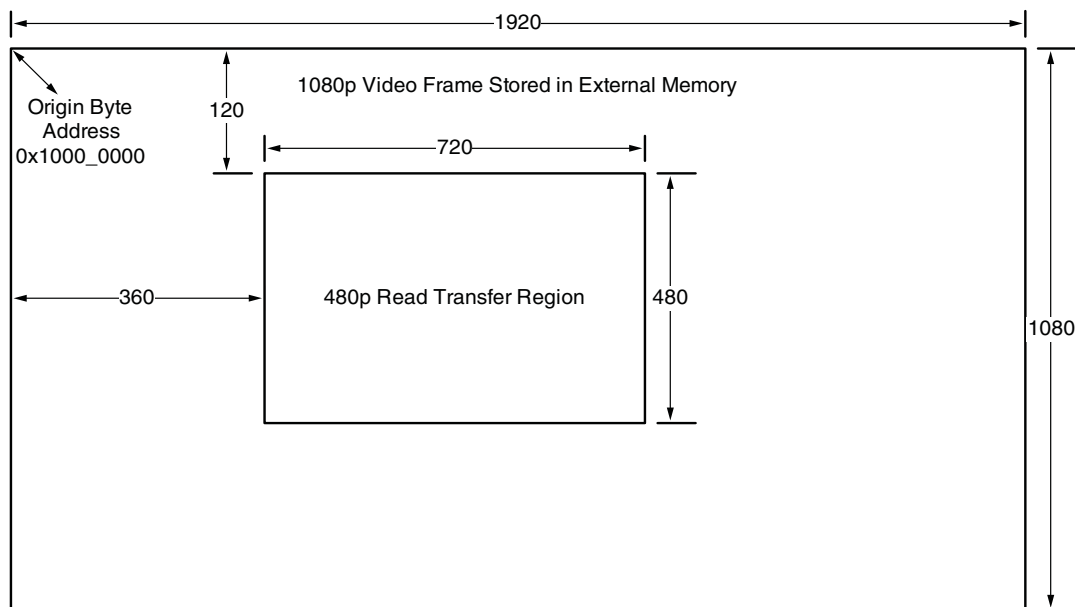
VFBC Transfer Examples

This section provides examples of typical video applications, the VFBC setup used to accomplish these applications, and the resultant transfers.

Frame Mode 1080p to 480p Window

The following figure is an example of a 1080p @ 60fps video source being written to a frame store in external memory. A 480p @ 60fps display is reading a 720x480 window of the 1920x1080 source video from the frame store.

This application would require two VFBC PIMs, one for writing the 1080p source and one for the 480p display. The following figure shows the video frame stored in external memory.



X10924

Figure 40: 1080p Frame to 480p Window

In this example, the 1080p source video frame is stored at address 0x10000000. Assuming that each pixel is stored at 32-bits-per-pixel resolution, each pixel is then 4 bytes. Because the X Size is stored in number of bytes, the X Size is 1920×4 (7680 bytes). This corresponds to the hexadecimal number of 0x1E00.

In this case the X Size and the Stride (also stored in number of bytes) are the same value, 0x1E00. The Y Size is stored as the number of lines minus 1, $1080 - 1$ (1079) or the hexadecimal number 0x437.

The following table shows the command packet for the 1080p source video. This packet could be written to the VFBC during each source video blank interval, for example:

Table 74: 1080p Source Video Command Packet Data Structure

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X Size	31 Write_NotRead	30:0 Start Address	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
0x0000_1E00		0x9000_0000		0x0000_0437		0x0000_1E00	

The 480p display in this example expects to read a 720x480 portion of the 1080p source from the external frame store starting from the 360th pixel on the 120th line. The X Size for the display is 720*4 (2880 bytes). This corresponds to the hexadecimal number 0xB40. The Stride remains the same as the 1080p source command, 0x1E00 (or 1920*4) because the video is stored as a 1080p frame in external memory. The Y Size is 480 – 1 (479) or the hexadecimal number 0x1DF.

The Start Address for the 480p video display includes the line and pixel offset information and is calculated by adding (120*1920 + 360)*4 to the origin or base address of the 1080p frame store. This corresponds to a final start address of 0x100E_15A0.

The following table shows the command packet for the 480p video display hardware.

Note: The Write_NotRead bit in Command Word 1 is now zero to denote a read transfer.

Table 75: 480p Video Display Command Packet Data Structure

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X Size	31 Write_NotRead	30:0 Start Address	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
0x0000_0B40		0x100E_15A0		0x0000_01DF		0x0000_1E00	

Several video functions can be performed by changing the Start Address within Command Word 1 during each video blank interval. For example:

- To cycle through multiple frame stores in external memory.
- To perform a pan-scan on a 480p display of a rescaled 16:9 source when combined with video scaler.

Line Mode 576p

The following example is of a 576p frame being read from external memory as individual lines. This example shows one transfer which is repeated for each line in the video. Each 576p video frame includes 576 line transfers. Each transfer has a different Start Address.

At the beginning of each line during the horizontal blank interval, the VFBC command interface and read interface must be reset for at least two clock cycles. Following the reset, the read command is written to the VFBC command interface. The VFBC read interface becomes non-empty several cycles following the command and data can be popped off of the read interface FIFO.

The following figure shows a transfer for a single 576p line.

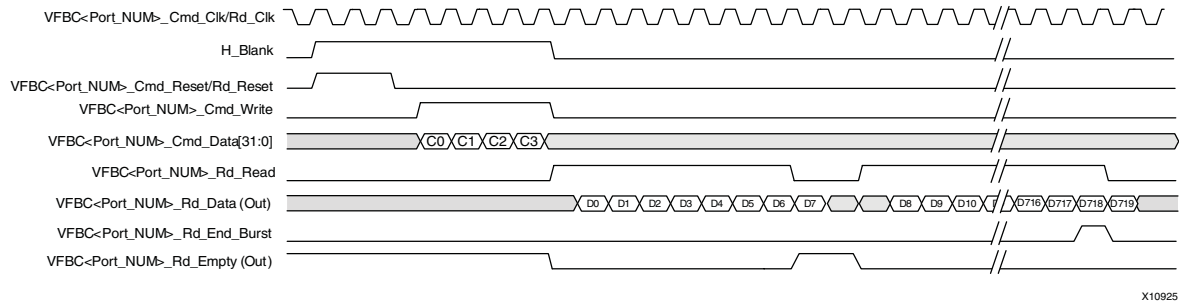


Figure 41: Line Mode 576p Transfer

Note: The VFBC<Port_Num>_Rd_End_Burst signal must be asserted on the last word transfer because 720 is not a multiple of the VFBC burst size of 32 words. In this example the assertion occurs on the 719th pixel read.

The following table shows the command words written to the command interface during the horizontal blank interval for the transfer of the first 576p line. The X Size is 720*4 bytes. The Y Size must be zero (denoting a single line transfer) and the Stride is ignored and can be any value. This example shows the Stride set to zero. The next line transfer has a Start Address of 0x1000_00B40. Each subsequent line transfer Start Address increments by 0xB40 during each horizontal blank interval.

Table 76: 576p Line Command Packet

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X Size	31 Write_NotRead	30:0 Start Address	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
0x0000_0B40		0x1000_000		0x0000_0000		0x0000_0000	

Simple Interlacing and De-interlacing (Field-Jam) Example

You can use the VFBC PIM for simple video processing such as interlacing or de-interlacing. The following tables show the VFBC commands to write a 480i source into a 480p frame store. As with the 1080p to 480p example, the X Size is set to 720*4 (0xB40). The Y size is set to 240-1 (0xEF) because a 480i field contains 240 lines.

There is a different VFBC command for each top and bottom field. The data is interleaved into the frame store by configuring the Stride to be two line lengths (2880*2, or 0x1680) and offsetting the bottom field Start Address by 720*4 (0xB40).

Table 77: 480i Top Field Command Packet

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X Size	31 Write_NotRead	30:0 Start Address	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
0x0000_0B40		0x9000_000		0x0000_00EF		0x000_1680	

The 480p command is similar to the 480i command packets, except the Stride is now a single 480p line, 720*4 (0xB40), the same as the X Size. The Y Size is set to 480-1 (0x1DF).

The Start Address is the same start address as the first line of the 480i top field (0x1000_0000). The following table shows the 480p command packet.

Table 78: 480p Bottom Field Command Packet

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X Size	31 Write_NotRead	30:0 Start Address	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
0x0000_0B40		0x1000_000		0x0000_001DF		0x0000_0B40	

For more information on VFBC transfers, HDL to interface to a VFBC, and example systems using a VFBC and multiple VFBC PIMs, refer to the *Video Starter Kit* on the Xilinx website. The "Reference Documents," page 193 contains a link to that site.

VFBC Synthesis Considerations

The following table shows the maximum frequency for the VFBC interface clocks, VFBC<Port_Num>_Cmd_Clk, VFBC<Port_Num>_Wd_Clk and VFBC<Port_Num>_Rd_Clk.

Table 79: Maximum VFBC Clock Frequencies by FPGA Family

FPGA Family	Clock FMAX	Notes
Spartan-3A DSP	133 MHz	With FIFO depths of 1024 32-bit words or fewer.
Virtex-4	167 MHz	
Virtex-5	200 Mhz	

The VFBC interface clocks can have a higher or lower frequency than the MPMC_Clk0. Each VFBC interface is asynchronous from the MPMC_Clk0 to support typical video clocks such as 27MHz or 74.25MHz. Higher frequencies than those listed in Table 79 might be achievable but results are dependent on device, utilization, and VFBC configuration.

VFBC Timing Constraints

MPMC provides a Tcl script that generates the timing constraints within a UCF file automatically for the VFBC PIM. For the timing constraints to be set correctly, the clock frequency of MPMC_Clk0 must be specified in the MHS file. The MHS file must have the CLK_FREQ value set for all input clock ports. The following code snippet is an example of an MHS file PORT declaration showing the direction as Input (I) and the CLK_FREQ set:

```
PORT display_clk_pin = display_clk, DIR = I, SIGIS = CLK, CLK_FREQ = 27000000,
BUFFER_TYPE = IBUFG
```

If the clock frequency is not set, the automatically generated VFBC timing constraints will assume the frequencies listed in Table 79 for the given device family.

Native Port Interface PIM

This NPI section covers the following topics:

- [NPI PIM Features](#)
- [Connecting a Custom PIM to an NPI PIM](#)
- [NPI Design Restrictions and Recommendations](#)
- [NPI Clock Requirements](#)
- [Configuring the NPI PIM](#)
- [NPI Timing Diagrams](#)

NPI PIM Features

The Native Port Interface (NPI) PIM:

- Allows you to extend the capabilities of MPMC to meet your own design needs.
- Offers a simple interface to memory that can easily be adapted to nearly any protocol.
- Provides address, data, and control signals to enable read and write requests for memory.
- Allows simultaneous push and pull of data from the port FIFOs.
- Has a configurable data width of 32 or 64 bits.
- When using 32-bit NPI, MPMC supports the following transfer sizes: byte, half-word, word, 4-word cacheline, 8-word cacheline, 16-word bursts, 32-word bursts, and 64-word bursts when using block RAM FIFOs. See ["Restrictions on 64-Word Burst Transfers," page 156](#) for more information.
- When using 64-bit NPI, MPMC supports the following transfer sizes: byte, half-word, word, double-word, 4-word cacheline, 8-word cacheline, 16-word bursts, 32-word bursts, and 64-word bursts.
- Runs only at a 1:1 clock ratio to the MPMC memory clock (`PORT_MPMC_Clk0`); a 1:2 clock ratio is *not* supported.

System design parameters are specified in the ["Design Parameters," page 1](#) and NPI Port signals are listed in ["PIM I/O Signals," page 18](#), respectively.

It is recommended that you review the ["Using MPMC in Standalone Systems," page 45](#) before designing a custom PIM. For more information about using the NPI PIM, refer to Answer Record #24912. A link to the Answer Record is located in the ["Reference Documents," page 193](#).

Connecting a Custom PIM to an NPI PIM

The parameters that will help you connect your custom PIM to the NPI PIM are:

- `C_PIM<Port_Num>_DATA_WIDTH` is set to either 32 or 64. This parameter specifies the width of `PIM<Port_Num>_WrFIFO_Data` and `PIM<Port_Num>_RdFIFO_Data` ports. The NPI data and address signals are labeled with little-endian bit/byte ordering as described in [Figure 8, page 78](#).
- `C_PIM<Port_Num>_BASETYPE` must be set to NPI (4).

NPI Design Restrictions and Recommendations

The following design restrictions in the NPI PIM are described in the subsequent sections:

- [Restrictions on Byte, Half-Word, Word, and Double-Word Write Transfers](#)
- [Restrictions between PIM<Port_Num>_AddrReq and PIM<Port_Num>_WrFIFO_Push](#)
- [Restrictions on Pipelining of Address Requests](#)
- [Restrictions on SRL FIFOs](#)
- [Restrictions on block RAM \(BRAM\) FIFOs](#)
- [Restrictions on 64-Word Burst Transfers](#)
- [Recommendation for Improving Write Latency](#)

Restrictions on Byte, Half-Word, Word, and Double-Word Write Transfers

The address phase, write data phase, and read data phase are independent unless MPMC is configured with the following settings:

- C_PIM<Port_Num>_DATA_WIDTH is set to 32 and C_MEM_DATA_WIDTH is set to 32 or 64, and using DDR or DDR2 memory.
- C_PIM<Port_Num>_DATA_WIDTH is set to 32 and C_MEM_DATA_WIDTH is set to 64 and using SDRAM memory.
- C_PIM<Port_Num>_DATA_WIDTH is set to 64 and C_MEM_DATA_WIDTH is set to 64, and using DDR or DDR2 memory.

If one of these cases exists, the following restrictions must be adhered to:

- The PIM<Port_Num>_WrFIFO_Push must occur a minimum of one cycle after the PIM<Port_Num>_AddrAck when requesting either a byte, half-word, word or double-word write transfer.
- Any PIM<Port_Num>_WrFIFO_Push corresponding to previous requests must be asserted before requesting a new byte, half-word, word or double-word write transfer.

Restrictions between PIM<Port_Num>_AddrReq and PIM<Port_Num>_WrFIFO_Push

Due to the definition of PIM<Port_Num>_AddrReq, write data must be pushed into the write FIFOs before it is required by the memory. For safest operation, assert the address request after all data has been pushed into the write FIFOs, as shown in ["64-bit NPI Timing Diagrams," page 158](#). See the ["Restrictions on Byte, Half-Word, Word, and Double-Word Write Transfers," page 155](#) for exceptions.

Restrictions on Pipelining of Address Requests

A FIFO that is controlling the PIM<Port_Num>_RdFIFO_RdWdAddr signal will store up to 4 read address requests only, regardless of transfer size; consequently, an NPI master is not allowed to queue more than 4 read requests without popping the corresponding data out of the Read FIFO.

Restrictions on SRL FIFOs

The user logic must prevent write FIFO overflows for NPI writes, facilitated by the use of the PIM<Port_Num>_WrFIFO_AlmostFull signal. MPMC prevents Read FIFO overflows when using the SRL FIFOs by automatically throttling the PIM<Port_Num>_AddrAck signal.

Restrictions on block RAM (BRAM) FIFOs

The implementation of BRAM FIFOs does not contain a valid `PIM<Port_Num>_WrFIFO_AlmostFull` signal, nor automatic throttling of `PIM<Port_Num>_AddrAck` during Reads. These optimizations were performed to improve the maximum MPMC clock frequency and result in restrictions for both NPI Reads and Writes as follows:

- For Reads, MPMC with BRAM FIFOs does not check how much data is in the Read FIFO before asserting the address acknowledge; consequently, an NPI master is not allowed to queue Read requests that total more than 1024 bytes of data without popping the corresponding data out of the FIFOs. The Read FIFOs will hold up to 1024 bytes of data, corresponding to four 64-word transfers. A further 64-word read request without popping earlier requests will overflow the read FIFO past 1024 bytes of data.
- For Writes, the NPI master is not allowed to queue up more than 1024 bytes of data in the Write FIFOs and the `PIM<Port_Num>_WrFIFO_AlmostFull` signal cannot be used. Consequently, there are options for ensuring the FIFOs will not overflow:
 - The first option is to push 1024 or less bytes of data into the Write FIFO, perform all address requests associated with this data, and then wait for the `PIM<Port_Num>_WrFIFO_Empty` to be asserted before pushing more data.
 - The second option is to push one transaction of data into the FIFOs, assert the address request on the last data beat of the transfer, and then wait for the address acknowledge before pushing more data into the FIFO. Waiting for the previous address acknowledge before pushing more data into the FIFO prevents overflows. This is due to the maximum size of MPMC requests, the BRAM Write FIFO size, and the address acknowledge behavior which is described in the next option.
 - A third, more aggressive operation of the Write FIFO occupancy can also be safely preformed. By relying on the current MPMC architecture maximum of two pending transactions for a particular port, a higher Write FIFO occupancy can be obtained. Because only two current transactions can be stored in the address controller, the 3rd assertion of `PIM<Port_Num>_AddrAck` indicates that the data associated with first Write transaction has been completely popped from the Write FIFO. This provides space for new data in the write FIFO that is the size of the first transaction. Using this method, you can safely estimate the maximum current occupancy of the Write FIFO, and throttle additional Write FIFO pushes safely below the 1024-byte limit.

Restrictions on 64-Word Burst Transfers

When using 32-bit NPI and SRL FIFOs, 64-word burst transfers are not supported because the data path FIFOs might not be deep enough. If 64-word burst transfers are required with 32-bit NPI, then BRAM FIFOs must be used. All custom 32-bit NPI PIMs should be carefully documented if they require 64-word burst transfers. The following table summarizes the 64-word burst support:

Table 80: 64-word NPI Burst Support

NPI Width	FIFO Support	Support Status
64	BRAM	Yes
64	SRL	Yes
32	BRAM	Yes
32	SRL	No

Recommendation for Improving Write Latency

In NPI write burst transactions, the general recommendation is to perform the address request after pushing in all the write data. The Write Data FIFO is not protected against underrun so this is the safest method of operation since it ensures all the data is present in the Write Data FIFO before it is moved out to memory.

If reduced write latency is desired, the user can take advantage of the behavior of the MPMC control logic to know when it is safe to make an earlier address request even though the write data for the transaction has not all been pushed in. The user can analyze the throughput by which data is written into the FIFO and the throughput by which the FIFO is drained out to memory to find a safe time to generate the early address request.

For example:

1. Assume the NPI width is 64 and the memory is a 32-bit DDR device. If the user design pushes in the Write data on every NPI clock cycle, the address request can be generated immediately after the first Write data beat is pushed in. This works because the memory data path cannot drain the FIFO faster than it is filled.
2. Assume the NPI width is 64 and the memory is a 64-bit DDR device. In this case, the memory data path can drain the Write Data FIFO at twice the maximum rate at which it can be filled. If the user design pushes in the Write data on every NPI clock cycle, the address request can be generated after half of the write data is pushed in. This ensures the Write Data FIFO does not underrun.

NPI Clock Requirements

The NPI PIM must run at the same frequency as MPMC. Support for any other frequency must be implemented in the custom PIM.

Configuring the NPI PIM

The NPI PIM is configured through the MPMC interface. Review the ["IP Configuration Graphical User Interface," page 189](#) for details on how to configure a PIM.

NPI Timing Diagrams

The following timing diagrams illustrate the functionality of the port interfaces. In the actual design signal names are prefixed with `PIM<Port_Num>_`, but in this section this prefix has been omitted for readability. Only a small sampling of possible timing diagrams are shown here. For example:

- 64-word burst transfers are not shown. These are very similar to the 32-word burst transfers, with the exception that there are more data beats.
- 32-bit NPI and 64-bit NPI are very similar. Differences are in the permitted address alignment and the number of data beats required to complete a particular transfer.

64-bit NPI Timing Diagrams

The following 64-bit NPI Timing Diagrams are illustrated subsequently:

- [Double-Word Write](#)
- [Double-Word Read](#)
- [8-Word, Cacheline Write](#)
- [8-Word, Cacheline Read](#)
- [32-Word, Burst Write](#)
- [32-Word, Burst Read](#)
- [8-Word, Cacheline Write with Almost Full Flag Asserted](#)
- [8-Word, Cacheline Read with Back-to-Back Transfers](#)
- [32-Word, Burst Read with Read FIFO Flush Asserted](#)

Double-Word Write

This figure shows the following:

- A 64-bit NPI.
- A double-word write transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a double-word boundary.
- The RdModWr must be asserted because the value of C_MEM_DATA_WIDTH is unknown.
- There is a Write Transfer Special Case (WrFIFO_Push asserted after AddrAck).

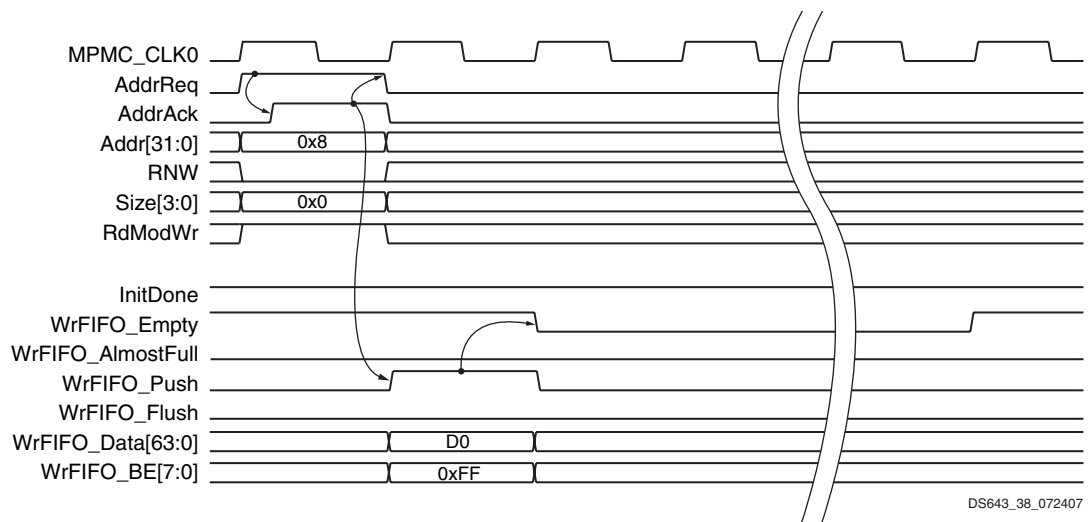
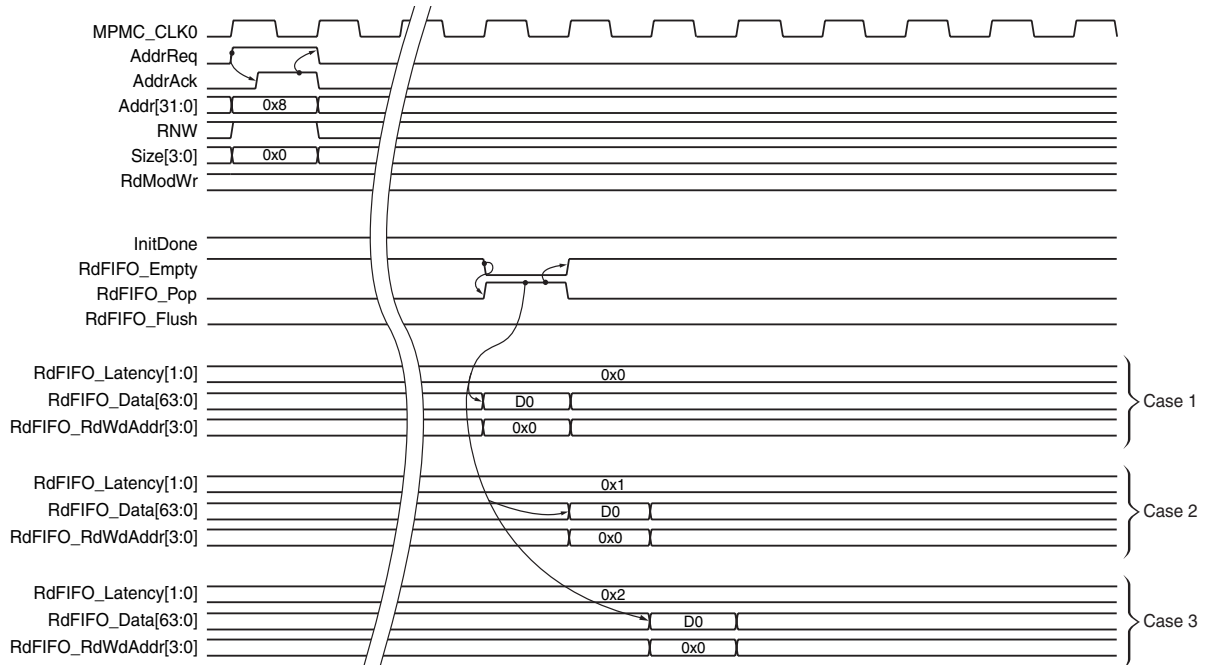


Figure 42: 64-bit NPI Double-word Write

Double-Word Read

This figure shows the following:

- A 64-bit NPI.
- A double-word read transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a double-word boundary.
- There are three cases of possible RdFIFO_Latency values.



DS643_39_072407

Figure 43: 64-bit NPI Double-word Read

8-Word, Cacheline Write

This figure shows the following:

- A 64-bit NPI.
- An 8-word, cacheline write transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on an 8-word boundary.
- The RdModWr does not need to be asserted because WrFIFO_BE is 0xFF during WrFIFO_Push, and because the 8-word transfer is larger than maximum value of 4 * C_MEM_DATA_WIDTH. RdModWr is discussed in "Error Correction Code," page 85.
- Write Transfer Safe Mode (AddrReq asserted on same cycle as last WrFIFO_push.)

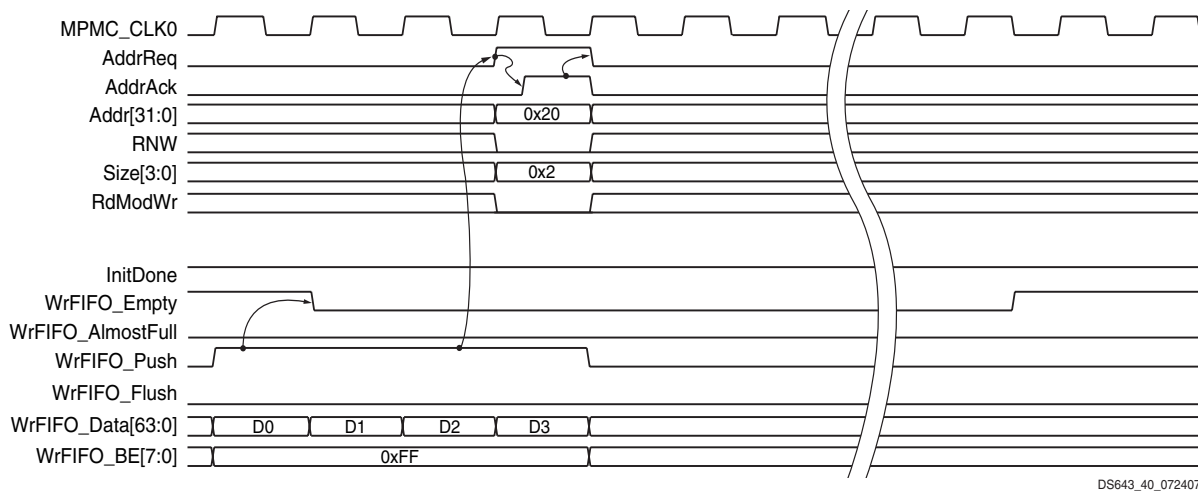


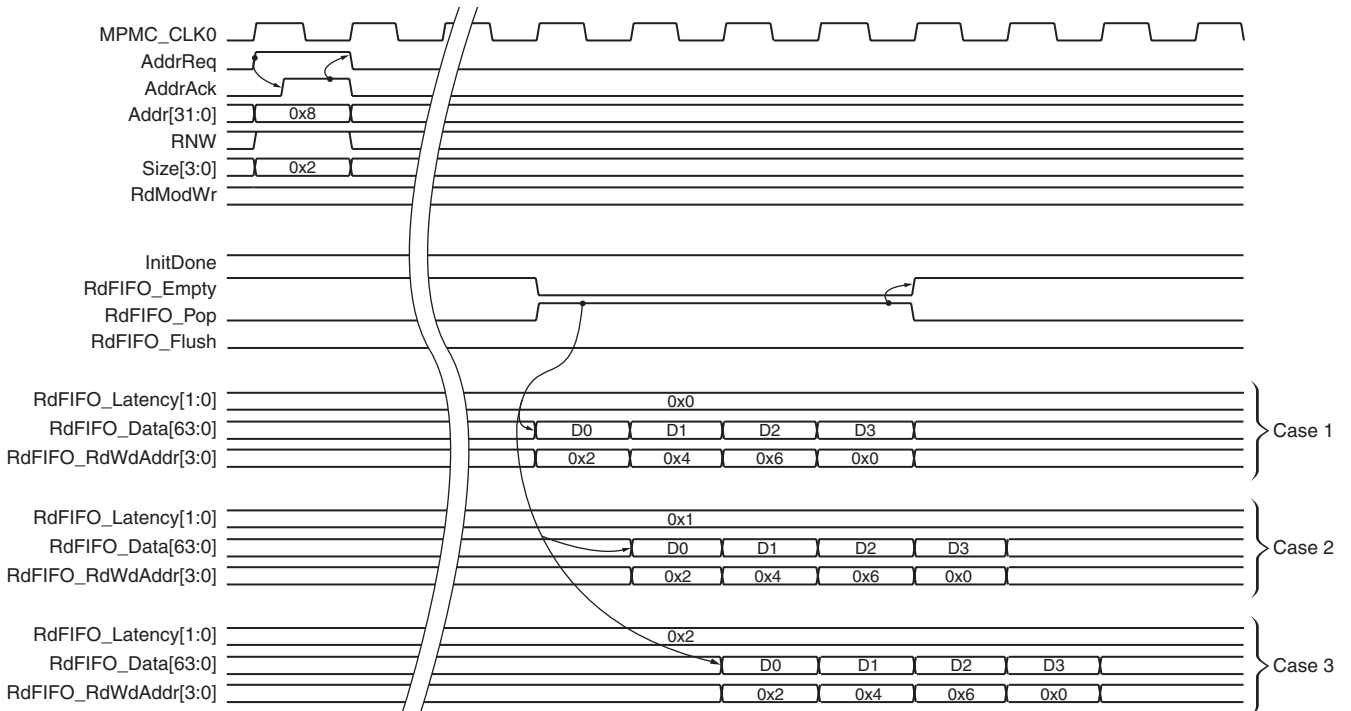
Figure 44: 64-bit NPI 8-word Cacheline Write

DS643_40_072407

8-Word, Cacheline Read

This figure shows the following:

- A 64-bit NPI.
- An 8-word cacheline read transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a double-word boundary.
- The RdFIFO_RdWdAddr indicates that data is returned target-word first.
- There are three cases of possible RdFIFO_Latency values.



X11045

Figure 45: 64-bit NPI 8-word Cacheline Read

32-Word, Burst Write

This figure shows the following:

- A 64-bit NPI.
- A 32-word, burst write transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a 32-word boundary.
- The RdModWr does not need to be asserted because WrFIFO_BE is 0xFF during WrFIFO_Push, and because a 32-word transfer is larger than maximum value of 4 * C_MEM_DATA_WIDTH.
- The Write Transfer Safe Mode is used (AddrReq is asserted on same cycle as the last WrFIFO_push.)

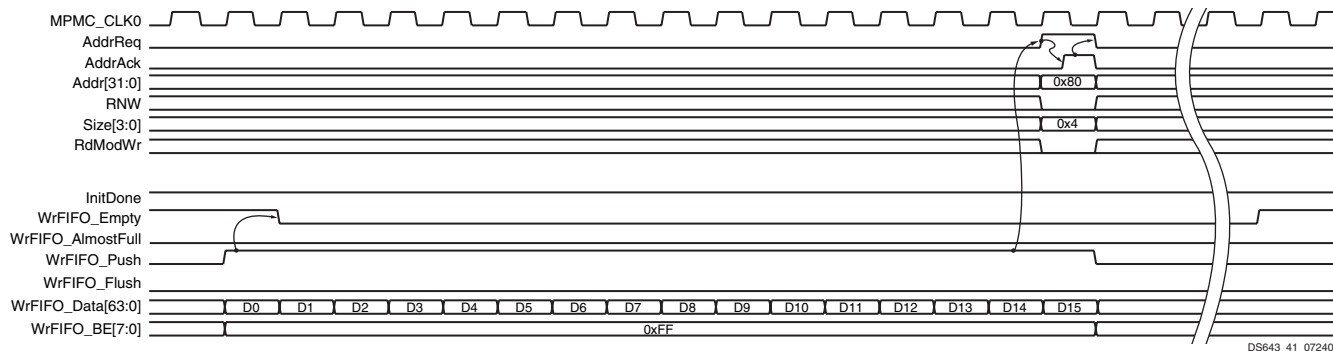


Figure 46: 64-bit NPI 32-word Burst Write

DS643_41_072407

32-Word, Burst Read

This figure shows the following:

- A 64-bit NPI.
- A 32-word, burst read transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a double-word boundary.
- There are three cases of possible RdFIFO_Latency values.

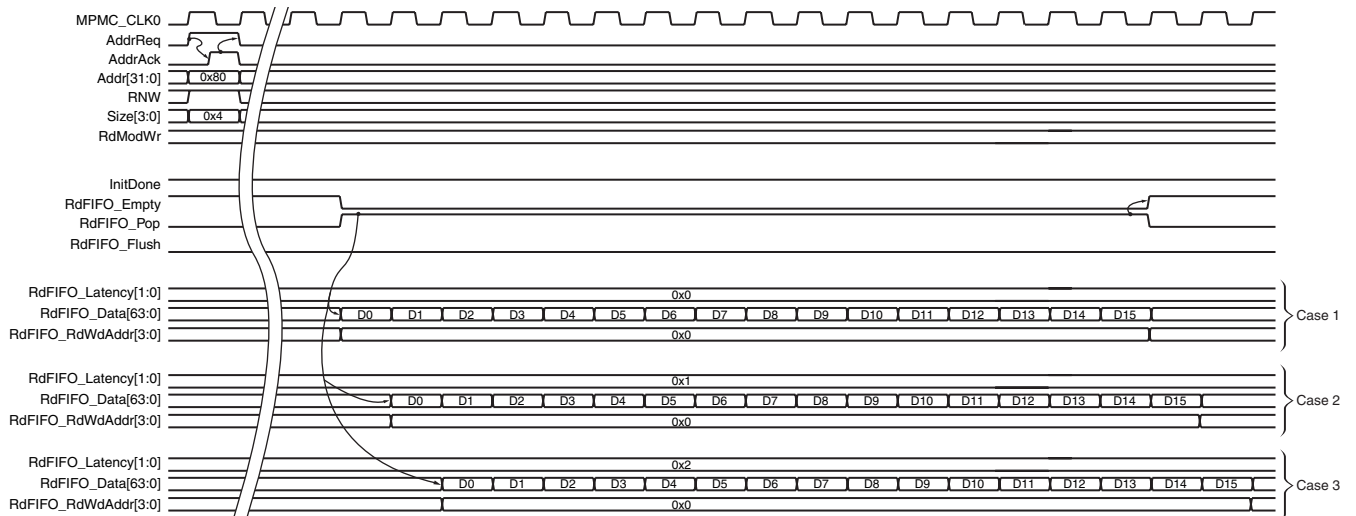


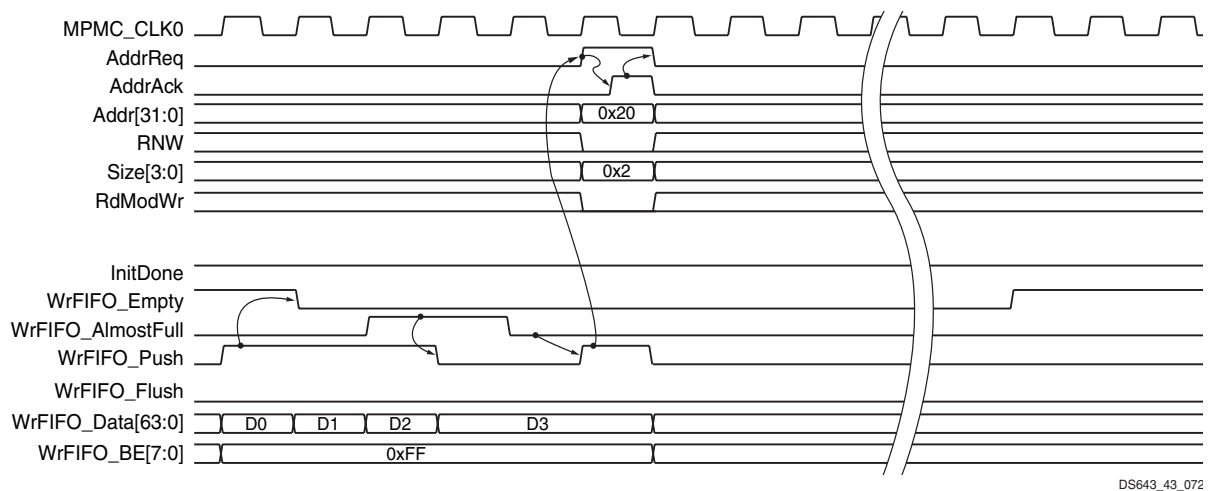
Figure 47: 64-bit NPI 32-word Burst Read

DS643_42_072407

8-Word, Cacheline Write with Almost Full Flag Asserted

This figure shows the following:

- A 64-bit NPI.
- An 8-word, cacheline write transfer.
- The `WrFIFO_AlmostFull` is asserted on same cycle as the third `WrFIFO_Push`. The fourth `WrFIFO_Push` and `AddrReq` are delayed until after `WrFIFO_AlmostFull` is deasserted.
- The address is acknowledged in the same cycle as it is requested.
- The address is on an 8-word boundary.
- The `RdModWr` does not need to be asserted because `WrFIFO_BE` is `0xFF` during `WrFIFO_Push`, and because an 8-word transfer is larger than maximum value of $4 * C_MEM_DATA_WIDTH$.
- The Write Transfer Safe Mode is used (`AddrReq` is asserted on same cycle as the last `WrFIFO_push`.)



DS643_43_072

Figure 48: 64-bit NPI 8-Word Cache-line Write with Almost Full Flag Asserted

8-Word, Cacheline Read with Back-to-Back Transfers

This figure shows the following:

- A 64-bit NPI.
- There are two back-to-back, 8-word burst, read transfers.
- First address acknowledged same cycle as requested.
- Second address is acknowledged cycle after request.
- There is no gap between address requests.
- The addresses are on double-word boundaries.
- The `RdFIFO_RdWdAddr` indicates that data is returned target-word first.
- There is a one cycle gap between read data for first request and second request. This could be more or less cycles depending on arbitration and pipeline settings.
- There are three cases of possible `RdFIFO_Latency` values.

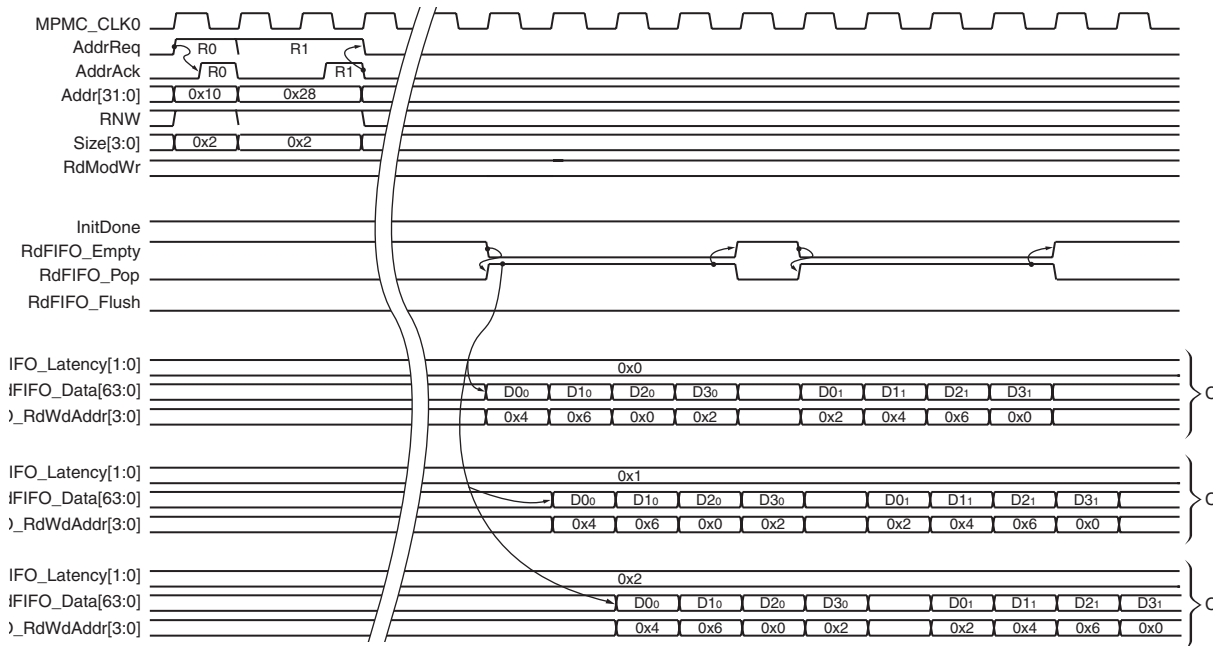
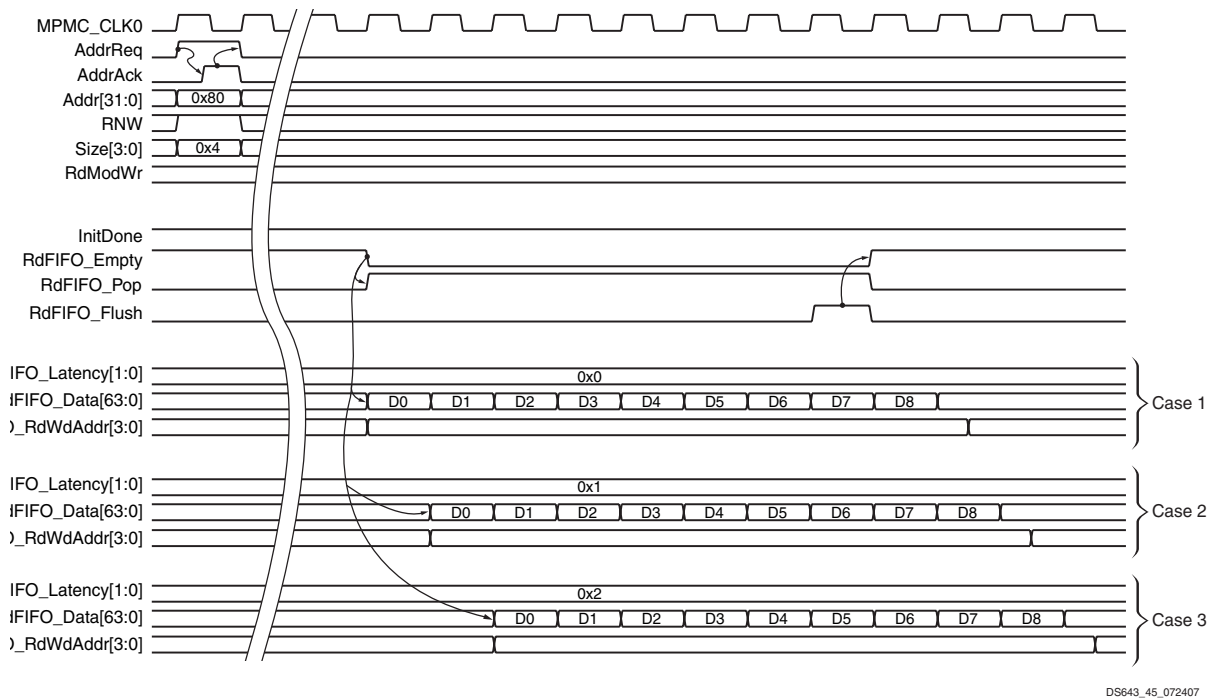


Figure 49: 64-Bit, NPI 8-Word Cacheline Read with Back-to-Back Transfers

32-Word, Burst Read with Read FIFO Flush Asserted

This figure shows the following:

- A 64-bit NPI.
- An 18-word, burst read transfer (a 32-word burst read transfer that is terminated by RdFIFO_Flush.)
- The address is acknowledged in the same cycle as it is requested.
- The address is on a 32-word boundary.
- There are three cases of possible RdFIFO_Latency values.



DS643_45_072407

Figure 50: 64-Bit NPI 32-word Burst Read with Read FIFO Flush Asserted

32-Bit NPI Timing Diagrams

The following 32-bit NPI timing diagrams are illustrated:

- "Word Write"
- "Word Read"
- "8-Word, Cacheline Write"
- "8-Word, Cacheline Read"

Word Write

This figure shows the following:

- A 32-bit NPI.
- A word write transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a word boundary.
- The RdModWr must be asserted because the value of C_MEM_DATA_WIDTH is unknown.
- The Write Transfer Special Case is used (WrFIFO_Push is asserted after AddrAck.)

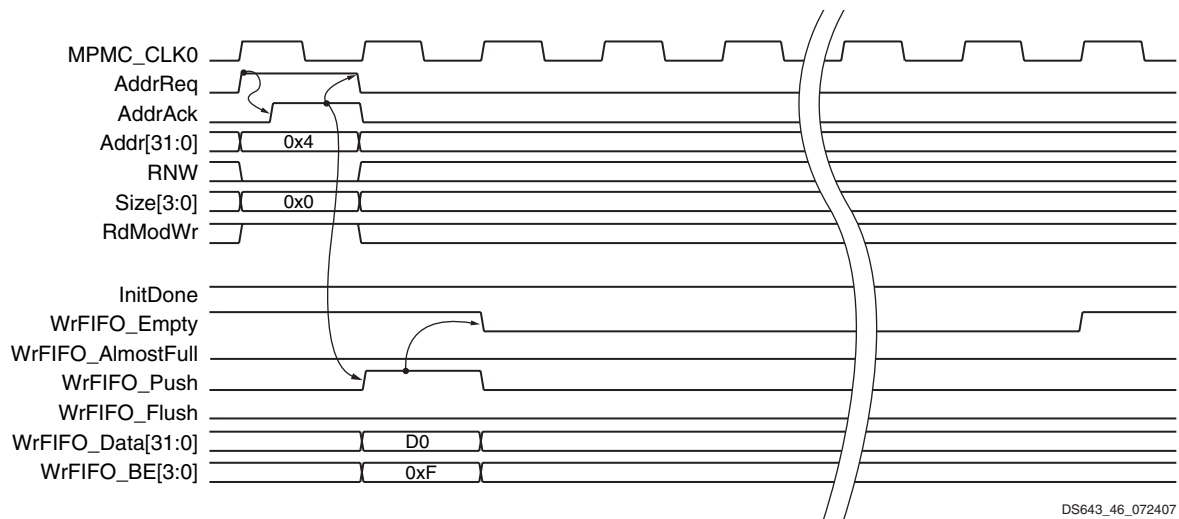
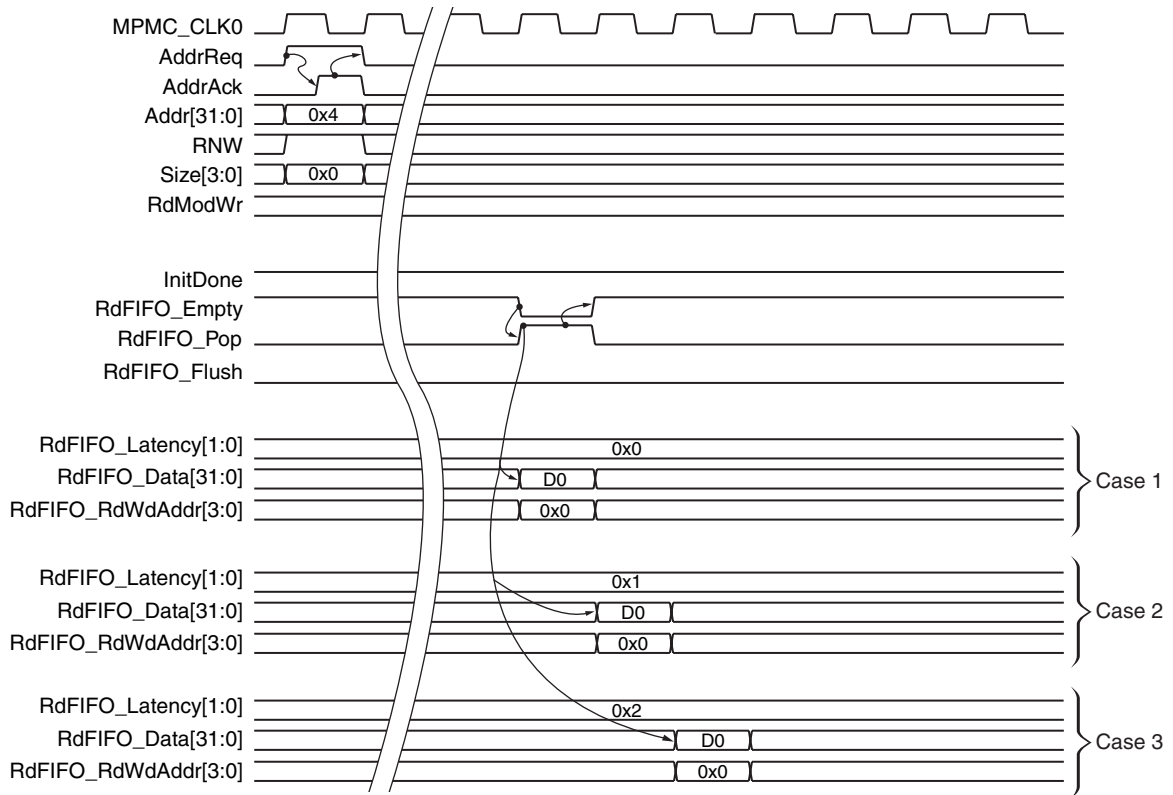


Figure 51: 32-Bit NPI Word Write

Word Read

This figure shows the following:

- A 32-bit NPI.
- A word read transfer.
- The address acknowledged same cycle as requested.
- The address is on a word boundary.
- There are three cases of possible `RdFIFO_Latency` values.



DS643_47_072407

Figure 52: 32-Bit NPI Word Read

8-Word, Cacheline Write

This figure shows the following:

- A 32-bit NPI.
- An 8-word, cacheline write transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on an 8-word boundary.
- The RdModWr does not need to be asserted because WrFIFO_BE is 0xF during WrFIFO_Push, and because the 8-word transfer is larger than the maximum value of $4 * C_MEM_DATA_WIDTH$.
- The Write Transfer Safe Mode is use (AddrReq is asserted on the same cycle as the last WrFIFO_push).

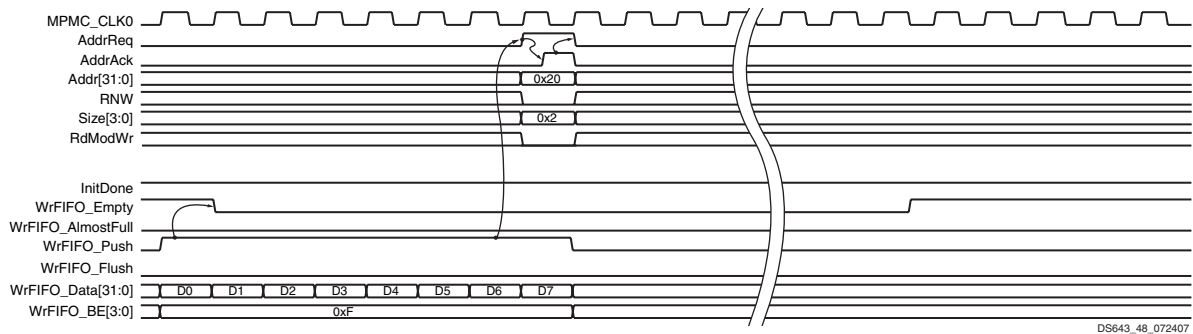
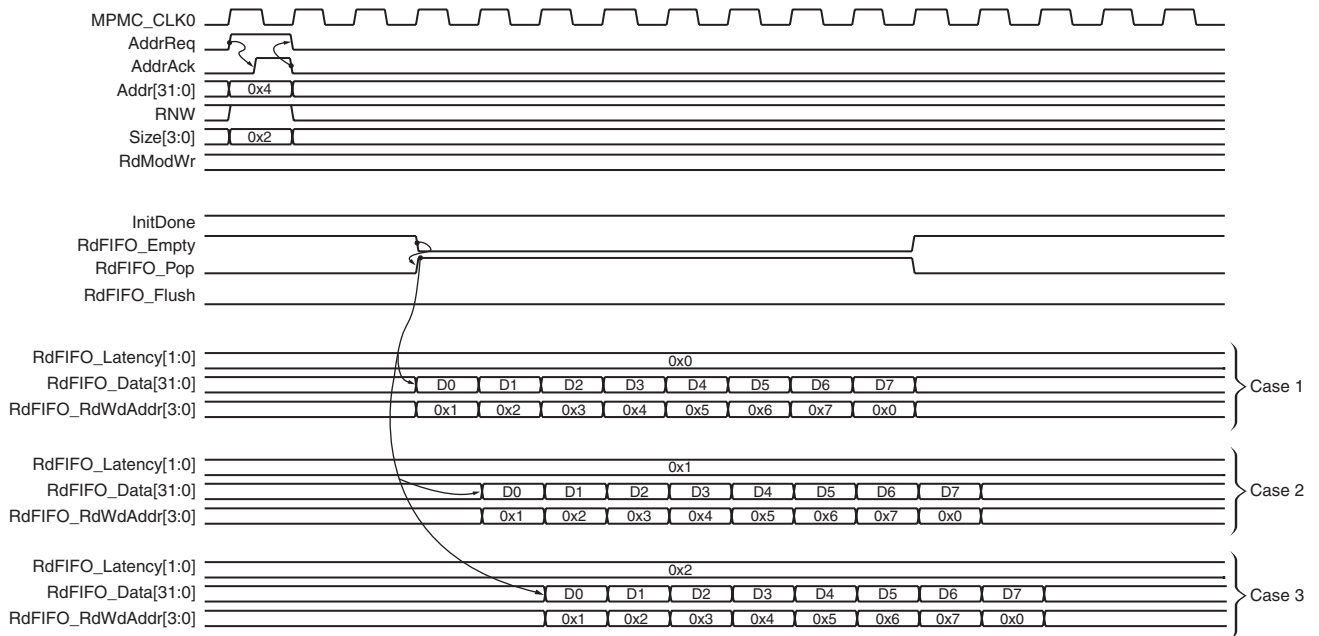


Figure 53: 32-Bit NPI 8-word Cacheline Write

8-Word, Cacheline Read

This figure shows the following:

- A 32-bit NPI.
- An 8-word cacheline read transfer.
- The address is acknowledged in the same cycle as it is requested.
- The address is on a word boundary.
- The RdFIFO_RdWdAddr indicates that data is returned target-word first.
- There are three cases of possible RdFIFO_Latency values.



X11007

Figure 54: 32-Bit NPI 8-word Cacheline Read

Performance, Timing, and Resource Utilization

The following sections describe:

- [MPMC Operational Frequencies](#)
- [MPMC Optimization](#)
- [MPMC Latency and Throughput](#)
- [Resource Utilization](#)

MPMC Operational Frequencies

This section contains:

- [Operating Frequency Range by Device](#)
- [MIG PHY Supported FMAX](#)

Operating Frequency Range by Device

Note: The following table lists the target operating frequency range for MPMC for a variety of FPGA device families and speed grades. These values provide guideline memory clock frequencies for MPMC.

Table 81: Target Operating Frequency Ranges by FPGA Family

FPGA Family	Target FMAX Range (MHz)
Spartan-3 Generation, -4 Speed Grade	125-133
Spartan-3 Generation, -5 Speed Grade	133-166
Virtex-4, -10 Speed Grade	165-185
Virtex-4, -11 Speed Grade	185-205
Virtex-4, -12 Speed Grade	200-225
Virtex-5, -1 Speed Grade	175-200
Virtex-5, -2 Speed Grade	185-220
Virtex-5, -3 Speed Grade	210-250

The values shown in [Table 81](#) are based on the following condition assumptions:

- Default parameters settings for MPMC
- 2:1 PIM clock ratios
- No use of floorplanning
- Default implementation tool options

Note: The target FMAX is influenced by the exact system and is provided for guidance. It is not a guaranteed value across all systems. The FMAX timing for the MPMC Performance Monitors has not been characterized and could be significantly lower. The FMAX timing for SDRAM has not been characterized and is likely to be limited by the physical system.

MIG PHY Supported FMAX

The Spartan-3, and Virtex-4, and Virtex-5 MIG PHYs also have maximum supported frequencies that should not be exceeded even if the MPMC is capable of meeting timing at higher clock frequencies.

Spartan-(3/3A/3AN/3E/3ADSP) devices can have data width limitations depending on part and package size. Verify data width compatibility through the MIG GUI or through the “Supported Devices” section of the *MIG User Guide*, ug086.pdf, in Section III, “Spartan- (3/3A/3AN/3E/3ADSP) FPGA to Memory Interfaces”. The ["Reference Documents," page 193](#) contains a link to this resource.

The following table summarizes the maximum supported frequencies of the MIG PHYs used in MPMC and also provides the name of the Application Note that describes the MIG algorithm used in the specified FPGA device family.

Table 82: MIG PHY Maximum Supported Frequencies by FPGA Family

FPGA Family	Speed Grade	Memory Type	Maximum Supported PHY Frequency	Application Note with MIG Algorithm	Notes
Spartan-3	-4	DDR/DDR2	133 MHz	XAPP454/768c	
Spartan-3	-5	DDR/DDR2 (Component)	166 MHz	XAPP454/768c	166 MHz limited to 8/16/32 bit designs on left or right sides of the parts.
Spartan-3	-5	DDR/DDR2 (DIMM)	133 MHz	XAPP454/768c	
Virtex-4	-10	DDR	175 MHz	XAPP701/702	MPMC uses Direct Clocking algorithm for both DDR and DDR2 on Virtex-4.
Virtex-4	-11	DDR	180 MHz	XAPP701/702	
Virtex-4	-12	DDR	185 MHz	XAPP701/702	
Virtex-4	-10	DDR2	220 MHz	XAPP701/702	
Virtex-4	-11	DDR2	230 MHz	XAPP701/702	
Virtex-4	-12	DDR2	240 MHz	XAPP701/702	
Virtex-5	-1	DDR2	266 MHz	XAPP858	
Virtex-5	-2	DDR2	300 MHz	XAPP858	
Virtex-5	-3	DDR2	333 MHz	XAPP858	
Virtex-5	-1	DDR	200 MHz	XAPP851	
Virtex-5	-2	DDR	200 MHz	XAPP851	
Virtex-5	-3	DDR	200 MHz	XAPP851	

MPMC Optimization

This section provides information about optimizing and tuning MPMC timing, performance, and size. It is recommended that you begin with creating a stable working MPMC system using default parameters settings and system configurations when possible. For example, start with an MPMC system created by the BSB tool in XPS. After establishing a working system, the following information can be used to fine tune the MPMC and experiment with different settings.

MPMC Performance Optimization

The following list identifies possible ways to improve MPMC performance:

1. Increase clock frequency as much as possible. Higher clock frequency increases throughput and can lower latency. It is useful to have the flexibility in the board to set the clock frequency to match the FMAX of the MPMC. For example, if a system has a fixed 100 MHz oscillator, the memory might have to run at 100 or 200 MHz. If the FMAX of the MPMC is 145 MHz, having the ability to install a 145 MHz oscillator is ideal for maximizing MPMC performance.
2. Latency can be reduced by disabling pipeline stages. Disabling pipeline stages has the trade-off that it might degrade timing. In some cases you could be running at a memory clock frequency well below the FMAX of the MPMC in which case disabling pipelines would not cause timing failures. It is recommended that you first get your system working in the default case where all pipeline stages are on and then experiment with disabling as many pipeline stages as possible while timing is still met. For example, the following parameters can be used to reduce latency in the MPMC core:

```
PARAMETER C_WR_DATAPATH_TML_PIPELINE = 0
PARAMETER C_ARB_PIPELINE = 0
PARAMETER C_PI<Port_Num>_ADDRACK_PIPELINE = 0
PARAMETER C_PI<Port_Num>_RD_FIFO_MEM_PIPELINE = 0
PARAMETER C_PI<Port_Num>_RD_FIFO_APP_PIPELINE = 0
PARAMETER C_PI<Port_Num>_WR_FIFO_MEM_PIPELINE = 0
PARAMETER C_PI<Port_Num>_WR_FIFO_APP_PIPELINE = 0
```

Note: Some FIFO pipeline parameters must be the same value across all ports. See "[MPMC Per-port Parameters](#)," [page 6](#) for more information

Additionally, some PIMs have optional pipeline stages that can be disabled:

XCL

```
PARAMETER C_XCL<Port_Num>_PIPE_STAGES = 0
PARAMETER C_XCL<Port_Num>_B_PIPE_STAGES = 0
```

MIB/PPC440MC

```
PARAMETER C_PPC440MC<Port_Num>_PIPE_STAGES = 0
```

3. Choosing memory with a lower CAS latency at the target operating clock frequency will reduce latency.
4. Experiment with CUSTOM arbitration. By changing the arbitration priorities to favor critical ports, higher throughput and potentially lower latency can be achieved for the critical ports. This is especially the case in a heavily loaded system with many active ports.
5. Use custom logic connected to an NPI interface rather than the using the PLB interface. The raw NPI interface offers better direct performance to the memory controller. However, attaching custom logic to the NPI interface may require more design and verification effort. When using NPI, larger bursts sizes offer higher throughput and memory utilization than smaller burst sizes.
6. In designs where the PowerPC 405 IPLB1 and DPLB1 ports are connected point-to-point to two separate MPMC ports, these two PLB ports could be run 1:1 clock ratio to the memory clock.

By running 1:1 instead of 1:2 clock ratio, the throughput and latency can be improved by the higher interface clock rate. The MPMC PLB PIM automatically instantiates specially optimized logic when it is directly connected point-to-point to a PowerPC 405 processor IPLB1 or DPLB1 port. This optimized logic can operate at a higher clock frequency than a normal PLB bus-based connection.

7. In Virtex-5 FXT designs using the PPC440MC interface, the `ppc440mc_ddr2` IP core should be used instead of MPMC whenever possible. The `ppc440mc_ddr2` IP core is optimized as a single port DDR2 memory for the PowerPC440 memory interface. The `ppc440mc_ddr2` IP core offers row/bank management, lower latency, and higher FMAX than MPMC. MPMC should only be used when SDRAM/DDR memory support or multiple memory ports are needed. The performance advantage of `ppc440mc_ddr2` is especially high with 64-bit DDR2 memories. In this case `ppc440mc_ddr2` offers a native 128 bit data path connection to the PowerPC440 block whereas the MPMC with PPC440MC PIM offers a maximum 64-bit data path to the PowerPC440 block.
8. In Virtex-5 FX designs using the PPC440MC interface with MPMC, the performance of DMA or burst based transactions is better with a larger burst length setting (Parameter `C_PPC440MC<Port_Num>_BURST_LENGTH`).
9. The MPMC PLB v4.6 PIM translates non-cache burst transfers into NPI transactions of 16 word bursts (`C_SPLB<Port_Num>_NATIVE_DWIDTH = 32`) or 32 word bursts (`C_SPLB<Port_Num>_NATIVE_DWIDTH = 64`). Non-cache PLB burst performance is maximized when the start address and burst length are aligned to the corresponding NPI transactions. Shorter length PLB burst transfers or transfers that cross NPI transaction boundaries may result in extra memory cycles where data is masked off or discarded. For example if a DMA engine is connected to the PLB port of MPMC, allocating the DMA engine's memory space in the software application to align to NPI transaction boundaries will enable more efficient data transfers and improve system performance.

MPMC Size Optimization

The following list identifies possible ways to reduce MPMC size:

1. Reducing the number of ports has the biggest effect on reducing core size. For example instead of two separate PLB ports to connect two PLB masters to MPMC, the two PLB masters can be connected to a PLB bus arbiter and attached to a single MPMC port. This will likely reduce system size, but might affect performance.
2. Reducing down to a single port MPMC further decreases size because data path switching logic can be removed and the arbitration logic can be fully removed for added logic savings. Therefore the MPMC size reduction is greater from two ports to one port than from three ports to two ports.
3. Match PHY SDR data path width to NPI data width. The SDR data path width of the PHY is 1x the memory width for SDRAM and 2x the memory width for DDR/DDR2. The NPI width of some PIMs can be selected between 32 and 64 bits. Designing the system so that the PIM NPI width and the PHY SDR data path widths match will optimize the flow of data and reduce logic utilization.

For example, if a system has a 16-bit DDR and uses PLB PIMs, the PLB PIM should be configured to have an NPI width of 32 bits to minimize MPMC size (Parameter `C_SPLB<Port_Num>_NATIVE_DWIDTH = 32`). If the default NPI width of 64 bits is used instead, wider data path FIFOs and data path switches will be generated causing more logic to be used.

VFBC, NPI, and PLB v4.6 PIMs all have user configurable NPI data widths. The SDMA PIM is fixed to a 64-bit NPI data width and the XCL PIM is fixed to a 32-bit NPI width. Refer to the "[Personality Interface Modules](#)," page 102 for more information

4. In PowerPC 405 processor systems, the IPLB0 and DPLB0 ports can be combined using a PLB arbiter to connect to a single MPMC port (see "[Standard PowerPC CoreConnect Use Case](#)," page 42 for an example.) This will reduce MPMC size compared to a two port MPMC with separate PLB PIMs connected to IPLB1 and DPLB1 with system performance as the trade-off.

5. In MicroBlaze systems, cached memory access travels over IXCL and DXCL ports to two separate MPMC XCL ports. By default, uncached access to memory travels over the PLB port to a third MPMC PLB port. This approach uses three MPMC ports per MicroBlaze. To save logic resources, the MicroBlaze could be configured with parameter `C_ICACHE_ALWAYS_USED = 1` and `C_DCACHE_ALWAYS_USED = 1` so that uncached memory access flows over the XCL ports. This allows the third MPMC port to be removed to reduce system size.

The MicroBlaze IXCL and DXCL ports can be connected to a dual XCL PIM which uses only one MPMC port for two XCL connections. Using the dual XCL PIM significantly reduces MPMC size.

6. Using a fixed priority arbiter (Parameter `C_ARB0_ALGO = FIXED`) will save some logic over a round robin arbiter and is a significant savings over a custom arbitration setting.
7. Using STATIC PHY over MIG PHY also reduces logic utilization because the static PHY is more simple in structure; however, the static PHY is also less robust than the MIG PHY and is therefore *not* recommended. See "[Configurable Physical Interface](#)," page 54 for more information.
8. Disabling optional pipeline stages in MPMC reduces flip-flop utilization but also degrades timing.
9. Minimize the number of ports or set ports to be read-only or write-only when possible.

Note: Ports with `C_PIM<Port_Num>_SUBTYPE` set to "IXCL" and "IPLB" are automatically configured to be read-only.

Timing Optimization

The following list identifies possible ways to improve timing of MPMC systems:

1. Generally, the size optimizations detailed in "[MPMC Size Optimization](#)," page 175 reduce MPMC complexity and will improve timing (except removal of pipeline stages.)
2. Better matching of data path widths between PHY and PIMs also improves timing because it minimizes MUXes used for width matching logic. For example, a 16-bit DDR memory has an internal 32-bit PHY data path. This would better match with a 32-bit NPI interface than a 64-bit NPI interface.
3. Experiment with different values for `C_RD_DATAPATH_TML_MAX_FANOUT`. All other pipeline stages are turned on by default. Keeping the pipeline stages enabled improves timing and FMAX.
4. Minimize the use of DCMs to synthesize the MPMC memory and PIM clocks. If possible the board should provide a clock source that directly inputs the desired MPMC memory clock frequency. Using on-chip clock synthesis and cascading of DCMs can increase clock jitter which degrades the timing budget of the overall system.
5. MPMC timing can be greatly affected by block RAM placement. The use of block RAM floorplanning can greatly improve MPMC timing results. If the number of available block RAMs in a design is low, freeing up block RAM can improve timing. This is especially important on Virtex-5 designs. Choosing the memory interface DQ pinout so they are placed near columns of block RAMs might also improve timing.
6. If possible, use fixed arbitration which simplifies the arbitration logic.
7. Move `C_WR_TRAINING_PORT` to a different port that is less timing critical (for example, using a DXCL port instead of an SDMA port for the write training pattern).
8. Run EDK in **xplorer** mode. This might significantly increase the tool run times but might also improve place and route results.

MPMC Latency and Throughput

This section provides MPMC latency and throughput estimations under a variety of MPMC configurations and memory speeds. The values are based on using default MPMC configurations (unless otherwise noted). Actual system performance can be affected by the timing of the memory part, PHY calibration settings, clock speeds, clock ratios, and the exact behavior of PIMs and devices connected to the MPMC.

Latency values are provided for the initial latency of the first transaction as measured from the transaction request at the PIM to the first data beat transferred.

Throughput values describe the theoretical maximum continuous rate of sustained data transfer using pipelined back-to-back burst transactions from an ideal device connected to MPMC. Throughput values might not take into account the fraction of total available memory bandwidth taken up by refresh operations. Refresh will slightly reduce the available bandwidth of the system.

The latency and throughput values were measured under the following conditions:

1. Spartan-3 generation measurements were taken with a Spartan-3A device using MT46V32M8-75 DDR memory or MT47H32M8-3 DDR2 memory.
2. Virtex-4 and Virtex-5 measurements were taken with a Virtex-5 device using MT46V32M8-75 DDR memory or MT47H32M8-3 DDR2 memory. Virtex-4 performance is assumed to be similar to Virtex-5 performance.
3. Nonregistered memory is used.

Note: These values are provided for guidance and are not a guarantee of performance under all conditions. Simulation and/or hardware instrumentation (such as using Chipscope™ Pro) should be used to verify performance targets in the full system.

NPI PIM Latency and Throughput

The latency and throughput at the NPI level reflects the performance of the MPMC core because NPI is the native interface of MPMC. The following table provides the NPI estimations by port, pipeline setting, memory interface, NPI width (in bits), and NPI burst type as well as the initial transaction latency, and maximum data throughput.

Table 83: NPI Latency and Throughput

Number of Ports	Pipeline settings	Memory Interface	NPI Width (Bits)	MPMC NPI Burst Type	Initial Transaction Latency (Memory Clocks)	Maximum Total Data Throughput (MBytes/sec)
Spartan-3 Generation Reads						
1-8	Default	DDR1 @ 100 MHz 32 bits	64	16 Word Burst	24	533
1-8	Default	DDR1 @ 100 MHz 32 bits	64	32 Word Burst	24	640
1-8	Default	DDR1 @ 100 MHz 32 bits	64	64 Word Burst	24	711
1-8	Default	DDR2 @ 133 MHz 16 bits	32	16 Word Burst	25	406
1-8	Default	DDR2 @ 133 MHz 32 bits	64	16 Word Burst	25	656
1-8	Default	DDR2 @ 133 MHz 32 bits	64	32 Word Burst	25	813
1-8	Default	DDR2 @ 133 MHz 32 bits	64	64 Word Burst	25	923
1-8	Default	DDR1 @ 83 MHz 16 bits	32	16 Word Burst	24	267
1-8	All Pipelines Off	DDR1 @ 83 MHz 16 bits	32	16 Word Burst	20	267
Virtex-4/Virtex-5 Reads						
1-8	Default	DDR2 @ 200 MHz 32 bits	64	16 Word Burst	23	853
1-8	Default	DDR2 @ 200 MHz 32 bits	64	32 Word Burst	23	1113
1-8	Default	DDR2 @ 200 MHz 32 bits	64	64 Word Burst	23	1313
1	Default	DDR2 @ 200 MHz 64 bits	64	16 Word Burst	23	800
2-8	Default	DDR2 @ 200 MHz 64 bits	64	16 Word Burst	23	1067
1-8	Default	DDR2 @ 200 MHz 64 bits	64	32 Word Burst	23	1113
2	Default	DDR2 @ 200 MHz 64 bits	64	32 Word Burst	23	1600
3-8	Default	DDR2 @ 200 MHz 64 bits	64	32 Word Burst	23	1707
1	Default	DDR2 @ 200 MHz 64 bits	64	64 Word Burst	23	1313
2	Default	DDR2 @ 200 MHz 64 bits	64	64 Word Burst	23	1600
3-8	Default	DDR2 @ 200 MHz 64 bits	64	64 Word Burst	23	2226
1-8	Default	DDR1 @ 100 MHz 32 bits	64	16 Word Burst	20	533
1-8	Default	DDR1 @ 100 MHz 32 bits	64	32 Word Burst	20	640
1-8	Default	DDR1 @ 100 MHz 32 bits	64	64 Word Burst	20	711
1-8	All Pipelines Off	DDR1 @ 100 MHz 32 bits	64	16 Word Burst	16	533
1-8	All Pipelines Off	DDR1 @ 100 MHz 32 bits	64	32 Word Burst	16	640
1-8	All Pipelines Off	DDR1 @ 100 MHz 32 bits	64	64 Word Burst	16	711

Table 83: NPI Latency and Throughput (Cont'd)

Number of Ports	Pipeline settings	Memory Interface	NPI Width (Bits)	MPMC NPI Burst Type	Initial Transaction Latency (Memory Clocks)	Maximum Total Data Throughput (MBytes/sec)
Spartan-3 Generation Writes						
1-8	Default	DDR1 @ 100 MHz 32 bits	64	16 Word Burst	N/A*	400
1-8	Default	DDR1 @ 100 MHz 32 bits	64	32 Word Burst	N/A*	533
1-8	Default	DDR1 @ 100 MHz 32 bits	64	64 Word Burst	N/A*	640
1-8	Default	DDR2 @ 133 MHz 16 bits	32	16 Word Burst	N/A*	328
1-8	Default	DDR2 @ 133 MHz 32 bits	64	16 Word Burst	N/A*	474
1-8	Default	DDR2 @ 133 MHz 32 bits	64	32 Word Burst	N/A*	656
1-8	Default	DDR2 @ 133 MHz 32 bits	64	64 Word Burst	N/A*	813
1-8	Default	DDR1 @ 83 MHz 16 bits	32	16 Word Burst	N/A*	222
1-8	All Pipelines Off	DDR1 @ 83 MHz 16 bits	32	16 Word Burst	N/A*	222

* Latency on writes is not characterized because MPMC allows write data to be pushed in before or after the address request.

Virtex-4/Virtex-5 Writes						
1-8	Default	DDR2 @ 200 MHz 32 bits	64	16 Word Burst	N/A*	610
1-8	Default	DDR2 @ 200 MHz 32 bits	64	32 Word Burst	N/A*	883
1-8	Default	DDR2 @ 200 MHz 32 bits	64	64 Word Burst	N/A*	1138
1-8	Default	DDR2 @ 200 MHz 64 bits	64	16 Word Burst	N/A*	753
1-8	Default	DDR2 @ 200 MHz 64 bits	64	32 Word Burst	N/A*	1219
1	Default	DDR2 @ 200 MHz 64 bits	64	64 Word Burst	N/A*	1600
2-8	Default	DDR2 @ 200 MHz 64 bits	64	64 Word Burst	N/A*	1766
1-8	Default	DDR1 @ 100 MHz 32 bits	64	16 Word Burst	N/A*	400
1-8	Default	DDR1 @ 100 MHz 32 bits	64	32 Word Burst	N/A*	533
1-8	Default	DDR1 @ 100 MHz 32 bits	64	64 Word Burst	N/A*	640
1-8	All Pipelines Off	DDR1 @ 100 MHz 32 bits	64	16 Word Burst	N/A*	400
1-8	All Pipelines Off	DDR1 @ 100 MHz 32 bits	64	32 Word Burst	N/A*	533
1-8	All Pipelines Off	DDR1 @ 100 MHz 32 bits	64	64 Word Burst	N/A*	640

Notes on NPI Throughput

1. NPI throughput increases with burst size so the 64 word bursts offer the highest maximum bandwidth, but might increase the delay on other ports.
2. The throughput of the NPI interface is limited to a 64 bit wide data path running at the memory clock speed. Because the memory can have up to a 64-bit DDR interface (128 bit wide SDR data path), it is possible that a single NPI port might not be able to access the full available bandwidth of the memory. In such situations, multiple NPI ports could be needed to fully utilize the memory in systems requiring highest throughput.
3. The MPMC control logic does not support row or bank management. After each NPI transaction, the row and bank that was accessed is closed with a precharge.

PLB PIM Latency and Throughput

The following table provides latency and throughput of each PLB PIM port with performance information for various MPMC and PLB PIM configurations.

Table 84: PLB PIM Latency and Throughput

Pipeline Settings	Memory Interface	NPI Width	PLB Burst Type	Memory to PLB Clock Ratio	Initial Transaction Latency (PLB Clocks)	Maximum Total Data Throughput of a PLB Port (MBytes/sec)
Spartan-3 Generation Reads						
Default	DDR1 @100 MHz 32 bits	64	16 Doublewords	1:1	29	483
Default	DDR2 @133 MHz 16 bits	32	16 Words	2:1	17	177
Default	DDR2 @133 MHz 32 bits	64	16 Doublewords	2:1	17	353
Default	DDR1 @83 MHz 16 bits	32	16 Words	1:1	29	201
All Pipelines Off	DDR1 @83 MHz 16 bits	32	16 Words	1:1	25	210
Spartan-3 Generation Writes						
Default	DDR1 @100 MHz 32 bits	64	16 Doublewords	1:1	4	254
Default	DDR2 @133 MHz 16 bits	32	16 Words	2:1	4	113
Default	DDR2 @133 MHz 32 bits	64	16 Doublewords	2:1	4	225
Default	DDR1 @83 MHz 16 bits	32	16 Words	1:1	4	106
All Pipelines Off	DDR1 @83 MHz 16 bits	32	16 Words	1:1	4	106
Virtex-4/Virtex-5 Reads						
Default	DDR2 @200 MHz 32 bits	64	16 Doublewords	2:1	16	530
Default	DDR2 @200 MHz 64 bits	64	16 Doublewords	2:1	16	530
Default	DDR1 @100 MHz 32 bits	64	16 Doublewords	1:1	25	484
All Pipelines Off	DDR1 @100 MHz 32 bits	64	16 Doublewords	1:1	21	507
Virtex-4/Virtex-5 Writes						
Default	DDR2 @200 MHz 32 bits	64	16 Doublewords	2:1	4	338
Default	DDR2 @200 MHz 64 bits	64	16 Doublewords	2:1	4	381
Default	DDR1 @100 MHz 32 bits	64	16 Doublewords	1:1	4	304
All Pipelines Off	DDR1 @100 MHz 32 bits	64	16 Doublewords	1:1	4	304

Notes on PLB PIM Throughput

The PLB PIM latency is measured from `PAValid` of the first PLB transaction to when the first `PLB_wrDack` or `PLB_rdDack` is asserted. The throughput of the PLB PIM is measured by using a PLB master to generate a continuous stream of 16-word or 16-double word bursts over a Point-to-Point PLB connection. These bursts are address-aligned to the size of the burst and include the generation of secondary address requests.

Table 84 describes the maximum throughput of a PLB PIM. The maximum throughput of each PLB PIM is usually less than the available throughput of the MPMC core. Though each PIM is usually not capable of fully utilizing the available bandwidth, the remaining MPMC Core/NPI bandwidth is available for other ports to use.

For example, a six port MPMC with 64-bit DDR2 memory at 200 MHz is capable of 2226 MBytes/sec of theoretical read throughput. In that configuration a single PLB PIM is capable of 427 MBytes/sec of throughput. Therefore:

- Four PLB ports, each running at maximum throughput, could achieve a combined system throughput of up to 1708 MBytes/sec (limited by the PLB PIMs).
- Five PLB ports, each running at maximum throughput, could achieve a combined system throughput of up to 2135 MBytes/sec (limited by the PLB PIMs).
- Six PLB ports, each running at maximum throughput, could achieve a combined system throughput of up to 2226 MBytes/sec (limited by the MPMC Core).

The PLB PIM throughput is most efficient when the PLB burst address range fits within MPMC 16-word (32 bit NPI) and 32-word (64 bit NPI) burst boundaries. [Table 84, page 180](#) assumes the bursts are aligned to the burst size.

XCL PIM Latency and Throughput

The following table describes the latency and throughput for the XCL PIM port. The table provides performance information for common MPMC and XCL PIM configurations. The XCL PIM latency is measured from the first push of data into the access FIFO until the first data is available on the read data FIFO. The throughput transactions were measured by transferring 1 kilobyte of data in simulation.

The time between the first transaction and the last data item is used to divide the number of bytes transferred to calculate the read throughput. The time measured between the first transaction and the last transaction has been sent over the access FIFO is used for the write throughput calculation.

Table 85: XCL PIM Latency and Throughput

Pipeline Settings	Memory Interface	Linesize	Port Subtype	Memory to XCL Clock Ratio	Initial Transaction Latency (XCL Clocks)	Maximum Total Data Throughput of XCL Port (MBytes/sec)
Spartan-3 Generation Reads						
Default	DDR2 @ 133 MHz 32 bits	16	XCL	2:1	17	130
Default	DDR2 @ 133 MHz 16 bits	16	XCL	2:1	17	131
Default	DDR1 @ 83 MHz 16 bits	8	XCL	1:1	29	72
Default	DDR1 @ 83 MHz 16 bits	4	DXCL	1:1	29	41
None	DDR1 @ 83 MHz 16 bits	4	DXCL	1:1	23	51
Spartan-3 Generation Writes						
Default	DDR2 @ 133 MHz 32 bits	16	XCL	2:1	N/A	193
Default	DDR2 @ 133 MHz 16 bits	16	XCL	2:1	N/A	193
Default	DDR1 @ 83 MHz 16 bits	8	XCL	1:1	N/A	156
Default	DDR1 @ 83 MHz 16 bits	1	DXCL	1:1	N/A	35
None	DDR1 @ 83 MHz 16 bits	1	DXCL	1:1	N/A	35
Virtex-4/Virtex-5 Reads						
Default	DDR2 @ 200 MHz 32 bits	16	XCL	2:1	16	202
Default	DDR2 @ 200 MHz 16 bits	16	XCL	2:1	16	202
Default	DDR2 @ 200 MHz 16 bits	8	XCL	2:1	16	138
Default	DDR1 @ 100 MHz 16 bits	4	DXCL	1:1	26	54
None	DDR1 @ 100 MHz 16 bits	4	DXCL	1:1	20	69
Virtex-4/Virtex-5 Writes						
Default	DDR2 @ 200 MHz 32 bits	16	XCL	2:1	N/A	289
Default	DDR2 @ 200 MHz 16 bits	16	XCL	2:1	N/A	289
Default	DDR2 @ 200 MHz 16 bits	8	XCL	2:1	N/A	234
Default	DDR1 @ 100 MHz 16 bits	1	DXCL	1:1	N/A	42
None	DDR1 @ 100 MHz 16 bits	1	DXCL	1:1	N/A	42

VFBC PIM Latency and Throughput

The VFBC PIM uses the NPI interface of the MPMC, therefore the latency and throughput of the VFBC PIM is similar to the NPI PIM. The maximum throughput of the VFBC PIM is 95.2% of the NPI PIM throughput (see [Table 83, page 178](#)). The VFBC uses 32-word bursts and 64-bit NPI interface only; consequently, only those configurations from [Table 83](#) are valid for the VFBC numbers.

The VFBC PIM adds an additional 68 MPMC_C1k cycles for the first 32-word burst of each VFBC command. Each subsequent burst does not have an additional latency cost. The minimum number of cycles between each 32-word burst transaction is 0 MPMC_C1k cycles. This number will increase if the VFBC or MPMC FIFOs are not ready. For Write transactions, data is written into the NPI interface starting at 8 MPMC_C1k cycles before the NPI address request is asserted. (A complete Write transaction takes 16 MPMC_C1k cycles only for 32-word bursts.)

Resource Utilization

The following subsections detail the MPMC resource utilization:

- [Resource \(block RAM\) Utilization](#)
- [Resource \(Slice, LUT, and Flip-Flop\) Utilization](#)

Resource (block RAM) Utilization

By default, MPMC uses block RAM based FIFOs in each of its ports to buffer read and write data from the external memory to improve memory efficiency, reduce LUT utilization, and to improve timing. MPMC also provides a per-port parameter called `C_PI<Port_Num>_RD_FIFO_TYPE` and `C_PI<Port_Num>_WR_FIFO_TYPE` which can be used to disable FIFOs for (read-only or write-only ports) or to choose the use of SRL FIFOs instead of block RAM FIFOs.

MPMC block RAM utilization is outlined as follows:

- 1 block RAM for control state machine.
- 1 block RAM for arbitration if CUSTOM arbitration algorithm is used (`C_ARB0_ALGO = CUSTOM`). For FIXED or ROUND_ROBIN arbitration no block RAM is used. If only 1 port is used no block RAM is used.
- If ECC is enabled, 1 block RAM is used for 8-, 16-, and 32-bit DDR/DDR2 memories, and two block RAMs for 64-bit DDR/DDR2 memories.
- 1 block RAM for every port where a Performance Monitor is enabled.

The following table shows the block RAM usage for each Read and Write port:

Table 86: Read and Write Port block RAM Usage

Read Ports	0 block RAMs if write-only port (<code>C_PI<Port_Num>_RD_FIFO_TYPE = DISABLED</code>)
	1 block RAM for 32 bit NPI ⁽¹⁾ port width and 8/16 bit DDR/DDR2 memory
	1 block RAM for 32 bit NPI port width and 8/16/32 bit SDRAM memory
	2 block RAMs for 64 bit NPI port width or 32 bit DDR/DDR2 memory or 64 bit SDRAM memory
	On Virtex-5, there is an optimization where 1 block RAM is used for 64 bit NPI with 32 bit DDR and DDR2 or 64 bit SDRAM memory
	4 block RAMs for 64 bit DDR and DDR2 memory

Table 86: Read and Write Port block RAM Usage (Cont'd)

Write Ports	0 block RAMs if read-only port (C_PI<Port_Num>_WR_FIFO_TYPE = DISABLED) or for IXCL/IPLB PIM subtypes.
	1 block RAM for 32 bit NPI port width and 8/16 bit DDR/DDR2 memory
	1 block RAM for 32 bit NPI port width and 8/16/32 bit SDRAM memory
	2 block RAMs for 64 bit NPI port width or 32 bit DDR/DDR2 memory or 64 bit SDRAM memory
	On Virtex-5, there is an optimization where 1 block RAM is used for 64 bit NPI with 32 bit DDR/DDR2 or 64 bit SDRAM memory
	4 block RAMs for 64 bit DDR/DDR2 memory

1. NPI Width is defined as: SDMA = 64 bits, XCL = 32 bits PLB PIM = value of parameter C_SPLB<Port_Num>_NATIVE_DWIDTH (the default is 64 and this value must be 64 for IPLB and DPLB SUBTYPES), NPI = value of parameter C_PIM<Port_Num>_DATA_WIDTH (default is 64).

To conserve block RAMs, the MPMC can be configured to use SRL FIFOs instead of block RAM FIFOs. With a 64-bit NPI:

- Each read SRL FIFO uses approximately 288 LUTs per port.
- Each write SRL FIFO uses approximately 256 LUTs per port.

The use of SRL FIFOs might negatively impact timing due to the slower speed of SRL FIFOs compared to block RAMs (BRAMs).

Resource (Slice, LUT, and Flip-Flop) Utilization

This section provides an estimate for Slice, Lookup Table (LUT) and Flip-Flop (FF) utilization of MPMC. The values provided are for resource estimation and are not guaranteed. These estimates assume default MPMC settings and default implementation tool options. Actual FPGA resource utilization depends on exact MPMC configuration parameters and implementation tool optimizations including cross boundary logic optimization, logic sharing, register re-timing, global system optimization, logic trimming, timing targets, and implementation tool settings.

MPMC Core (Slice, LUT, and FF) Resource Utilization

The following table provides MPMC Core LUT and FF resource utilization. This includes the PHY, arbiter, control logic, and data path (using block RAM FIFOs) up to the internal NPI interfaces. PIM resource utilizations are provided separately in the tables that follow. An estimate of Slice utilization is also provided for reference. The reported Slice utilization numbers correspond to default slice packing effort for a medium full FPGA. The Slice utilization might vary in actual systems due to factors such as the fullness of the FPGA device, Slice packing effort, and the amount Slice merging with other logic in the system.

Table 87: LUT and Flip Flop Utilization Estimates⁽¹⁾

FPGA Family	Memory Width	Memory Type	Base Single Port MPMC Core Size			Each Additional Port Size		
			LUT Utilization	FF Utilization	Slice Utilization	LUT Utilization	FF Utilization	Slice Utilization
Spartan-3 Generation	16 ⁽²⁾	DDR/DDR2	340-390	850-970	620-720	180-200	220-250	210-240
Spartan-3 Generation	32	DDR/DDR2	450-510	1160-1330	840-970	170-200	250-290	220-260
Virtex-4	32	DDR/DDR2	1210-1390	1290-1480	1250-1440	250-290	310-360	280-320
Virtex-4	64	DDR/DDR2	2070-2380	2080-2390	2080-2390	240-280	350-410	290-340

Table 87: LUT and Flip Flop Utilization Estimates⁽¹⁾

FPGA Family	Memory Width	Memory Type	Base Single Port MPMC Core Size			Each Additional Port Size		
			LUT Utilization	FF Utilization	Slice Utilization	LUT Utilization	FF Utilization	Slice Utilization
Virtex-5	32	DDR/DDR2	960-1100	1410-1620	640-740	170-190	280-320	120-140
Virtex-5	64	DDR/DDR2	1770-2030	2050-2360	1030-1200	200-230	340-390	140-170

1. The size values provided assume that, on average, half the ports have 32 bit NPI interfaces.
2. If all ports have 32 bit NPI interfaces the data path size would be further reduced because a 32 bit NPI has the same SDR data width as a 16 bit DDR/DDR2 memory.

XCL PIM LUT and FF Resource Utilization

The following table provides the XCL PIM LUT and FF resource utilization

Table 88: XCL PIM LUT and FF Resource Utilization

FPGA Family	SUBTYPE	LUT Utilization	FF Utilization	Slice Utilization
Spartan-3	IXCL	150	107	115
Spartan-3	DXCL	168	121	129
Virtex-4	IXCL	161	119	134
Virtex-4	DXCL	194	129	152
Virtex-5	IXCL	189	112	101
Virtex-5	DXCL	207	131	118

PLB PIM LUT and FF Utilization

The following tables provide the PLB PIM LUT and FF utilization by FPGA device family, NPI width, and PLB SUBTYPE

Table 89: PLB PIM LUT and FF Resource Utilization

FPGA Family	PLB PIM SUBTYPE	NPI Width	LUT Utilization	FF Utilization	Slice Utilization
Spartan-3	PLB	32	571-631	388-497	437-524
Spartan-3	PLB	64	853-919	589-673	635-721
Spartan-3	Singles	32	222-284	333-399	274-344
Spartan-3	Singles	64	296-371	474-553	346-424
Virtex-4	PLB	32	593-655	388-495	435-532
Virtex-4	PLB	64	893-944	597-672	648-717
Virtex-4	DPLB	64	471-549	490-579	413-512
Virtex-4	IPLB	64	304-340	331-388	297-340
Virtex-4	Singles	32	229-328	332-400	267-333
Virtex-4	Singles	64	371-438	476-559	370-432
Virtex-5	PLB	32	480-577	385-488	303-370

Table 89: PLB PIM LUT and FF Resource Utilization (Cont'd)

Virtex-5	PLB	64	677-850	585-669	478-581
Virtex-5	Singles	32	195-212	330-394	207-224
Virtex-5	Singles	64	286-308	471-547	252-308

SDMA PIM LUT and FF Resource Utilization

The following table provides SDMA PIM Slice, LUT, and FF Resource Utilization for different FPGA devices.

Table 90: SDMA PIM LUT and FF Resource Utilization by FPGA Device

C_SPLB_DWIDTH	C_SPLB_P2P	C_PI_RDDATA_DELAY	C_COMPLETED_ERR_TX	C_COMPLETED_ERR_RX	Flip-Flops	LUTs (4-input)	Slice Utilization
xc3s1400a-4-fg676							
32	1	0	0	0	962	1881	1296
32	0	0	0	0	1036	1878	1323
32	0	1	0	0	1036	1878	1323
32	0	2	0	0	1109	1924	1364
32	0	2	1	0	1110	1925	1366
32	0	2	1	1	1111	1927	1368
64	0	2	1	1	1111	1927	1368
128	0	2	1	1	1111	1927	1367
xc4vfx100-10-ff1152							
32	1	0	0	0	1018	1957	1300
32	0	0	0	0	1066	1961	1332
C_SPLB_DWIDTH	C_SPLB_P2P	C_PI_RDDATA_DELAY	C_COMPLETED_ERR_TX	C_COMPLETED_ERR_RX	Flip-Flops	LUTs (4-input)	Slice Utilization
32	0	1	0	0	1066	1961	1332
32	0	2	0	0	1139	2026	1384
32	0	2	1	0	1141	2028	1386
32	0	2	1	1	1143	2030	1388
64	0	2	1	1	1143	2030	1388
128	0	2	1	1	1143	2030	1388
xc5vlx220-2-ff1760							
32	1	0	0	0	944	1486	738
32	0	0	0	0	1018	1532	869
32	0	1	0	0	1018	1532	869
32	0	2	0	0	1091	1625	792
32	0	2	1	0	1092	1627	844
32	0	2	1	1	1093	1623	813
64	0	2	1	1	1093	1623	773
128	0	2	1	1	1093	1620	795

PPC440MC LUT and FF Resource Utilization

The PPC440MC PIM uses approximately 50 Slices, 120 FFs, and 60 LUTs on Virtex-5 FXT.

VFBC PIM Slice, LUT, and FF Resource Utilization

The following table provides the NPI data width, the allowable types for C_PI0_WR_FIFO_TYPE, C_PI0_RD_FIFO_TYPE, the C_VFBC_RDWD_DATA_WIDTH, Slice utilization, LUT utilization, FF utilization, and block RAM by FPGA architecture.

Table 91: VFBC LUT and FF Resource Utilization

FPGA Architecture	NPI Data Width	C_PI_WR_FIFO_TYPE	C_PI_RD_FIFO_TYPE	C_VFBC_RDWD_DATA_WIDTH	Slice Utilization	LUT Utilization	FF Utilization	block RAM
Virtex-5	32	BRAM	BRAM	8	436	1,167	1,010	3 RAMB36s
	32	BRAM	BRAM	16	517	1,169	1,033	3 RAMB36s
	32	DISABLED	BRAM	32	368	1,045	895	2 RAMB36s
	32	BRAM	DISABLED	32	480	1,028	946	2 RAMB36s
	32	BRAM	BRAM	32	509	1,169	1,031	3 RAMB36s
	32	BRAM	BRAM	64	522	1,165	1,035	5 RAMB36s
	64	BRAM	BRAM	8	551	1,191	1,167	5 RAMB36s
	64	BRAM	BRAM	16	539	1,184	1,165	5 RAMB36s
	64	DISABLED	BRAM	32	468	1,044	961	3 RAMB36s
	64	BRAM	DISABLED	32	425	978	1,035	3 RAMB36s
	64	BRAM	BRAM	32	525	1,187	1,179	5 RAMB36s
Virtex-4	32	BRAM	BRAM	8	1,133	1,869	1,075	3 RAMB16s
	32	BRAM	BRAM	16	1,131	1,867	1,073	3 RAMB16s
	32	DISABLED	BRAM	32	1,035	1,754	944	2 RAMB16s
	32	BRAM	DISABLED	32	1,044	1,721	984	2 RAMB16s
	32	BRAM	BRAM	32	1,129	1,864	1,072	3 RAMB16s
	32	BRAM	BRAM	64	1,112	1,854	975	5 RAMB16s
	64	BRAM	BRAM	8	956	1,486	1,164	5 RAMB16s
	64	BRAM	BRAM	16	954	1,484	1,162	5 RAMB16s
	64	DISABLED	BRAM	32	833	1,371	873	3 RAMB16s
	64	BRAM	DISABLED	32	861	1,300	980	3 RAMB16s
	64	BRAM	BRAM	32	990	1,513	1,114	5 RAMB16s
64	BRAM	BRAM	64	1,001	1,539	1,125	5 RAMB16s	

Table 91: VFBC LUT and FF Resource Utilization (Cont'd)

FPGA Architecture	NPI Data Width	C_PI_WR_FIFO_TYPE	C_PI_RD_FIFO_TYPE	C_VFBC_R DWD_DAT_WIDTH	Slice Utilization	LUT Utilization	FF Utilization	block RAM
Spartan-3 ADSP	32	BRAM	BRAM	8	1,139	1,778	1,246	3 RAMB16BWEs
	32	BRAM	BRAM	16	1,137	1,776	1,244	3 RAMB16BWEs
	32	DISABLED	BRAM	32	1,049	1,664	1,116	2 RAMB16BWEs
	32	BRAM	DISABLED	32	1,049	1,628	1,159	2 RAMB16BWEs
	32	BRAM	BRAM	32	1,137	1,774	1,242	3 RAMB16BWEs
	32	BRAM	BRAM	64	1,148	1,886	1,062	5 RAMB16BWEs
	64	BRAM	BRAM	8	997	1474	1206	5 RAMB16BWEs
	64	BRAM	BRAM	16	988	1524	1105	5 RAMB16BWEs
	64	DISABLED	BRAM	32	859	1384	901	3 RAMB16BWEs
	64	BRAM	DISABLED	32	864	1322	977	3 RAMB16BWEs
	64	BRAM	BRAM	32	1006	1555	1122	5 RAMB16BWEs
	64	BRAM	BRAM	64	1005	1557	1100	5 RAMB16BWEs

IP Configuration Graphical User Interface

The IP Configuration interface offers a convenient and user-friendly way to configure a system. You can invoke the IP Configurator GUI within XPS by double-clicking the MPMC IP in the System Assembly View, or by right-clicking MPMC and selecting Configure IP. The tabs in the interface are:

- [Base Configuration](#)
- [Memory Interface](#)
- [Port Configuration](#)
- [Advanced](#)

To create a functional MPMC, you need to use the following three tabs only:

- **Base Configuration:** select the port type for each active port and remove unused ports between active ports
- **Memory Interface:** select the memory part and configure the memory settings
- **Port Configuration:** set the parameters for each port

The Advanced tab contains advanced features such as per-port addresses, per-port data path configuration, arbitration algorithms, performance, and debugging. This tab is intended for advanced users only.

Base Configuration

MPMC provides up to eight ports for accessing the memory; each port can be configured to a different bus or port type. The Base Configuration tab lets you set:

- Port types of the eight available ports: the dropdown box contains INACTIVE, XCL, PLBv46, SDMA, NPI, PPC440MC, and VFBC
- Common base addresses

The Port Configuration box contains an interface diagram of MPMC; under each port interface is a dropdown selection that lets you choose the port type. "[Memory and Memory Part Parameters](#)," [page 11](#) contains a description of the parameter that specifies port types.

The LeftJustify button is for eliminating the inactive ports between active ports, which is desirable because inactive ports also consume logic. During this justifying process, all parameters related to the ports and external bus connections are shifted left accordingly. Below is an example representing the action that occurs with a left-justification of ports.

	Port0	Port1	Port2	Port3	Port4	Port5	Port6	Port7
Before Left Justifying	XCL	PLBv46	Inactive	Inactive	XCL	Inactive	SDMA	Inactive
After Left Justifying	XCL	PLBv46	XCL	SDMA	Inactive	Inactive	Inactive	Inactive

The Common Addresses box lets you enter common Base Addresses and SDMA Register Base Addresses. Common base addresses are used by every port for address decoding. For common SDMA addresses, each port has its own set of registers which are offset at a fixed size from the common SDMA base address. To configure each individual port address, go to the Address option under the Advanced tab.

Memory Interface

MPMC can work with a wide variety of memory parts from different manufacturers. This tab lets you select the memory part used in conjunction with the MPMC.

MPMC has a list of supported memory devices. To select the device to meet your needs, use the Memory Part Selector area, where available memory parts can be filtered based on the following criteria:

- Type
- Memory manufacturer
- Memory style
- Memory density – capacity of memory
- Memory width – data width of memory

The filtered results appear in the Part No. drop-down list. The Memory Part Selector/Part No. drop-down contains a CUSTOM option also, if the correct part is not available in the built-in memory database. After you select the CUSTOM option, all the memory parameters are editable and you can enter the parameters you require. After you have selected a memory part, its parameters are automatically loaded into the Selected Memory Info area and the Memory/DIMM Settings tab.

Note: If you select a similar memory part first and then select the CUSTOM option, the memory parameters from the first memory part are retained, but are the fields are editable. This can simplify the process of entering a CUSTOM memory part if it is in a format that is similar to one in the database.

Memory/DIMM Settings Tab

The Memory/DIMM Settings tab provides the following areas: Settings, Configuration, and Info.

Under the Settings area, selection and drop-down boxes are available where you can adjust:

- Number of DIMMs
- Memory Data Width
- Memory Clock Period (ps)
- ODT Setting
- Part Information

A check box is available for Reduced Drive Output also.

Based on the selected device, the appropriate parameters are editable.

The Configuration area provides selection options for CE Width, ODT Width, Clock Width, CSn Width, and No. of Ranks. A check box is available for Registered Memory also.

The Info area provides the following Memory Width options: DM, ADDr, Bank Addr, and DQS.

Memory Part Settings Tab

The Memory Part Settings tab has two areas: Part Settings and Memory Timing Settings.

- Part Settings contains options for selecting Data Depth, Data Width, Bank, Row, and Column Bits.
- Memory Timing Settings contains memory part options.

MIG Settings Tab

The MIG Settings tab allows you to launch the MIG Tool from within the MPMC GUI. It also allows you to set some MIG specific parameters. For more information about the integrated MIG tool flow and how to set the MIG specific parameters, refer to "[MIG/MPMC Tool Flow](#)," page 57.

Port Configuration

In the Port Configuration tab you can configure the parameters of each individual port. Tabs are available for Port 0-3 and Port 4-7. The Port tabs are divided into quadrants with a port number represented in each quadrant. Only the parameters related to the current port type (selected in the Base Configuration tab) are active and editable. For a detailed description of parameters, refer to "[Memory and Memory Part Parameters](#)," page 11.

Advanced

The Advanced tab contains the tabs:

- [Address Configuration](#)
- [Data Path](#)
- [Arbitration](#)
- [ECC/Debug](#)
- [Misc](#)

These tabs offer more advanced customization of MPMC.

Address Configuration

In the Address Configuration tab, you can choose to ignore the common base addresses and configure the addresses of each individual port.

The Use Common Port Address check box is checked on by default. If you do not want to use common port address, you can deselect the check box to disable the use of common port addresses and enable the Port Addresses box. You can then change the port addresses of each port.

The rows in the Port Addresses box are:

- Base Address: Base address of the port as specified by `C_PIM<Port_Num>_BASEADDR` which is described in [Table 2, page 6](#).
- High Address: High address of the port as specified by `C_PIM<Port_Num>_HIGHADDR` which is described in [Table 2, page 6](#).
- Offset: Address offset as specified by `C_PIM<Port_Num>_OFFSET` which is described in [Table 2, page 6](#). This allows a multi-processor system to have identical logic memory space inside each processor while mapping each logical memory space to different physical locations inside the memory.
- SDMA Control Port Base Address: logical base address of SDMA channel (if configured as SDMA).
- SDMA Control Port High Address: logical high address of SDMA channel (if configured as SDMA).

The Restore Default Addr button restores the address to common address mode.

Data Path

In the Data Path tab, you can configure the common data path (pipeline) settings and the individual settings for each port.

General Pipeline Settings let you set the general pipeline parameters, and Port-specific Settings let you change the pipeline settings for the individual ports as follows:

- NPI Width: Width of Native Port Interface.
- Read FIFO Config: Implement FIFO using block RAM, SRL, or Wr-Only (Write Only, Read FIFO Disabled).
- Write FIFO Config: Implement FIFO using block RAM, SRL, or Rd-Only (Read Only, Write FIFO Disabled).
- Read Memory Pipeline: Enable pipeline for read access to memory.
- Read Port Pipeline: Enable pipeline for read access to the underlying port.
- Write Memory Pipeline: Enable pipeline for write access to memory.
- Write Port Pipeline: Enable pipeline for write access to port.
- Address Ack Pipeline: Enable pipeline for acknowledgement of address request.

Arbitration

MPMC has a maximum of eight ports that can all access the memory at the same time. Consequently, it is very important to have an arbitration algorithm to determine which port has priority at any given moment. In the Arbitration tab, you can choose what arbitration algorithm to use.

From the Select Arbitration Algorithm box, select one of the three following arbitration algorithms:

- Round Robin: Perform Round Robin arbitration.
- Fixed: Perform Fixed priority arbitration.
- Custom: In this mode, you can customize the number of time slots, as well as the arbitration priorities in each time slot. The arbitration priority in each time slot is encoded as a string to indicate decreasing priorities among ports.
For example, the string "01234567" gives the highest priority to port 0, then decreasing priorities from port 1 through port 7.

ECC/Debug

In the ECC/Debug tab, you can set debug, Error Correction Code (ECC), and Performance Monitor (PM) related options. All debug registers, ECC registers and PMs are accessed using a Control bus interface.

In the General ECC/Debug Settings box, you can enable debug, ECC, and PM. If any one of them is enabled, the Control Bus is activated and you must enter the base address in the Control Base Address box.

After you enable ECC, you can configure the behavior further in the ECC Settings box.

After you enable PM, you can change the general parameters of the Performance Monitor in the Common Performance Monitor Settings box, and the port-specific parameters in Port-specific Performance Monitor Settings box.

Misc

The Misc tab contains parameters that require your attention but do not fall into any of the previous categories.

Reference Documents

- http://www.xilinx.com/ise/embedded/edk_docs.htm
- ISE documentation page: http://www.xilinx.com/ise/logic_design_prod/foundation.htm
- Memory website contains links to the following documents: *Memory Interfaces Data Capture Using Direct Clocking Technique* and *MIG User Guide*, which provides additional MIG information and documentation http://www.xilinx.com/products/design_resources/mem_corner
- *MicroBlaze Processor Reference Guide*
http://www.xilinx.com/ise/embedded_design_prod/index.htm
- Spartan-3 generation FPGAs page:
http://www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/index.htm
- Virtex-4 page: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/index.htm
- Virtex-5 page: http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/index.htm
- *Embedded Processor Block in Virtex-5 FPGAs Reference Guide*
<EDK Install Directory>/doc/usenglish/embedproc_ug200.pdf
- *Specification for PLB v4.6 Bus Protocol with Xilinx Simplifications, SP026*,
<EDK Install Directory>/doc/usenglish/sp026.pdf
- *PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide*:
<EDK Install Directory>/doc/usenglish/mg_ug.pdf
- Application Notes:
 - XAPP768C, *Interfacing Spartan-3 Devices With 166 MHz or 333 Mb/s DDR SDRAM Memories*: <http://www.xilinx.com/support/software/memory/protected/XAPP768c.pdf>
 - XAPP454, *DDR2 SDRAM Memory Interface for Spartan-3 FPGAs*
<http://www.xilinx.com/bvdocs/appnotes/xapp454.pdf>
 - XAPP701, *DDR2 SDRAM Physical Layer Using Direct-Clocking Technique*:
http://www.xilinx.com/support/documentation/application_notes/xapp701.pdf
 - XAPP851DDR SDRAM Controller Using Virtex-5 FPGA Devices:
http://www.xilinx.com/support/documentation/application_notes/xapp851.pdf
 - XAPP858, *High-Performance DDR2 SDRAM Interface in Virtex-5 Devices*:
http://www.xilinx.com/support/documentation/application_notes/xapp858.pdf
- Design resources page:
http://www.xilinx.com/products/design_resources/proc_central/index.htm
- Development boards page: <http://www.xilinx.com/products/devboards/index.htm>
- IBM CoreConnect documentation:
 - *CoreConnect Device Control Register Bus: Architecture Specification, IBMCoreConnect 64-Bit Processor Local Bus: Architecture Specification* and *IBM CoreConnect 64-Bit On-Chip Peripheral Bus: Architectural Specification*: http://www.xilinx.com/products/ipcenter/dr_pcentral_coreconnect.htm
- LocalLink User Interface page:
http://www.xilinx.com/aurora/aurora_protocol_member/sp006.pdf
- *Video Starter Kit*:
http://www.xilinx.com/products/devboards/reference_design/vsk_s3/vsk_s3.htm
- Answer Records (Contains useful design, debug, and implementation):
http://www.xilinx.com/xlnx/xil_ans_browser.jsp

