



Processor Local Bus (Version 2.3)
On-Chip Peripheral Bus (Version 1.3)
Device Control Register Bus (Version 1.9)
Architectures

Preliminary

Preliminary Edition (November 1997)

This edition provides preliminary user information that applies to the IBM bus architectures used in Core+ASIC development.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the products in this publication, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying product descriptions are error-free.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time.

It is possible that this publication may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the written permission of IBM.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address comments about this publication to:

IBM Corporation
Department H83A
P.O. Box 12195
Research Triangle Park, NC 27709

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

©Copyright International Business Machines Corporation 1997. All rights reserved.

Printed in the United States of America.

4 3 2 1

Notice to U.S. Government Users—Documentation Related to Restricted Rights—Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Patents and Trademarks

IBM may have patents or pending patent applications covering the subject matter in this publication. The furnishing of this publication does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904, United States of America.

The following terms are trademarks of IBM Corporation:

IBM

PowerPC

PowerPC Architecture

PowerPC Embedded Controllers

Other terms which are trademarks are the property of their respective owners.

Contents

About This Book	xix
Chapter 1. Overview.....	1-1
Processor Local Bus	1-2
On-Chip Peripheral Bus	1-2
Device Control Register Bus	1-3
Terms And Definitions	1-4
Chapter 2. Processor Local Bus.....	2-1
PLB Overview.....	2-1
PLB Features.....	2-2
High Performance.....	2-2
System Design Flexibility.....	2-2
PLB Implementation	2-3
PLB Transfer Protocol	2-4
Overlapped PLB Transfers	2-5
PLB Signals.....	2-6
Signal Naming Conventions	2-6
PLB System Signals.....	2-10
Sys_plbClk (System PLB Clock).....	2-10
Sys_plbReset (System PLB Reset).....	2-10
PLB Arbitration Signals.....	2-11
Mn_request (Bus Request).....	2-11
Mn_priority(0:1) (Request Priority).....	2-11
Mn_busLock, PLB_busLock (Bus Arbitration Lock)	2-12
PLB_PAVValid (PLB Primary Address Valid)	2-12
PLB_SAVValid (Secondary Address Valid)	2-15
SI_wait (Wait for Address Acknowledge).....	2-16
SI_addrAck, PLB_MnAddrAck (Address Acknowledge).....	2-17
SI_rearbitrate, PLB_MnRearbitrate (Rearbitrate PLB).....	2-17
Mn_abort, PLB_abort (Abort Request)	2-18
PLB Status Signals.....	2-18
PLB_pendReq (PLB Pending Bus Request)	2-18

PLB_pendPri(0:1) (Pending Request Priority)	2-18
PLB_reqPri(0:1) (Current Request Priority)	2-19
PLB_masterID(0:3) (PLB Master Identification).....	2-19
PLB Transfer Qualifier Signals	2-19
Mn_RNW, PLB_RNW (Read/NotWrite)	2-19
Mn_BE(0:3), PLB_BE(0:3) (Byte Enables)	2-20
Mn_size(0:3), PLB_size(0:3) (Transfer Size).....	2-22
Mn_type(0:2), PLB_type(0:2) (Transfer Type)	2-23
Mn_compress, PLB_compress (Compressed Data Transfer)	2-24
Mn_guarded, PLB_guarded (Guarded Memory Access).....	2-24
Mn_ordered, PLB_ordered (Ordered Transfer)	2-25
Mn_lockErr, PLB_lockErr (Lock Error Status)	2-25
Mn_ABus(0:31), PLB_ABus(0:31) (Address Bus)	2-26
PLB Write Data Bus Signals	2-26
Mn_wrDBus(0:31), PLB_wrDBus(0:31) (Write Data Bus)	2-27
SI_wrDack, PLB_MnWrDack (Write Data Acknowledge)	2-28
SI_wrComp, (Data Write Complete)	2-28
Mn_wrBurst, PLB_wrBurst (Write Burst)	2-28
SI_wrBTerm, PLB_MnWrBTerm (Write Burst Terminate)	2-29
PLB_wrPrim (Write Secondary to Primary Indicator).....	2-29
PLB Read Data Bus Signals.....	2-30
SI_rdDBus(0:31), PLB_MnRdDBus(0:31) (Read Data Bus)	2-30
SI_rdWdAddr(0:3), PLB_MnRdWdAddr(0:3) (Read Word Address)	2-31
SI_rdDack, PLB_MnRdDack (Read Data Acknowledge)	2-31
SI_rdComp, (Data Read Complete)	2-32
Mn_rdBurst, PLB_rdBurst (Read Burst)	2-32
SI_rdBTerm, PLB_MnRdBTerm (Read Burst Terminate)	2-33
PLB_rdPrim (Read Secondary to Primary Indicator)	2-33
Additional Slave Output Signals	2-34
SI_MBusy(0:n), PLB_MBusy(0:n) (Master Busy).....	2-34
SI_MErr(0:n), PLB_MErr(0:n) (Master Error)	2-34
PLB Interfaces	2-36
PLB Master Interface	2-37
PLB Slave Interface	2-38
PLB Arbiter Interface	2-39
PLB Operations	2-40
PLB Non-Address Pipelining	2-40

Read Transfers	2-41
Write Transfers	2-42
Transfer Abort	2-43
Back-to-Back Read Transfers	2-44
Back-to-Back Write Transfers.....	2-45
Back-to-Back Read - Write - Read - Write Transfers.....	2-46
Four-word Line Read Transfers.....	2-47
Four-word Line Write Transfers	2-48
Four-word Line Read Followed By Four-word Line Write Transfers.....	2-49
Sequential Burst Read Transfer Terminated by Master	2-50
Sequential Burst Read Transfer Terminated By Slave.....	2-51
Sequential Burst Write Transfer Terminated by Master.....	2-52
Sequential Burst Write Transfer Terminated By Slave	2-53
Fixed Length Burst Transfer - Notes	2-54
Fixed Length Burst Read Transfer	2-56
Fixed Length Burst Write Transfer.....	2-57
Back-to-Back Burst Read - Burst Write Transfers	2-58
Locked Transfer.....	2-59
Slave Requested Re-arbitration With Bus Unlocked	2-60
Slave Requested Re-arbitration With Bus Locked	2-61
Bus Time-Out Transfer	2-62
Bus Transfer Time-Out Notes.....	2-63
PLB Address Pipelining	2-64
Pipelined Back-to-Back Read Transfers.....	2-65
Pipelined Back to Back Read Transfers - Delayed AAck	2-66
Pipelined Back-to-Back Write Transfers.....	2-67
Pipelined Back-to-Back Write Transfers - Delayed AAck	2-68
Pipelined Back-to-Back Read and Write Transfers	2-69
Pipelined Back-to-Back Read Burst Transfers	2-70
Pipelined Back-to-Back Write Burst Transfers.....	2-71
PLB Bandwidth and Latency	2-72
PLB Master Latency Timer	2-72
PLB/DMA Operations	2-73
PLB/DMA Interface.....	2-73
PLB/DMA Signals	2-74
PLB/DMA Side-Band Signals	2-74
SI_rdDataXfer (Slave Read Data Transfer)	2-74
SI_wrDataXfer (Slave n Write Data Transfer).....	2-75

DMA_endOp (DMA End of Operation)	2-75
DMA_dataXfer (DMA Data Transfer).....	2-75
DMA_flyByBurst (DMA Flyby Burst)	2-75
SI_dmaBurstTerm (Slave n DMA Burst Terminate).....	2-76
DMA Transfer Signals	2-76
INT_dmaReq(0:3) (Internal DMA transfer request)	2-76
DMA_intAck(0:3) (Internal DMA transfer acknowledge)	2-76
EXT_dmaReq(0:3) (External DMA transfer request)	2-76
DMA_extAck(0:3) (External DMA transfer acknowledge).....	2-76
DMA Transfer Operations.....	2-77
DMA Flyby Transfers (Mn_type = 0b001).....	2-77
DMA Buffered External Peripheral Transfers (Mn_type = 0b010)	2-77
DMA Buffered OPB Peripheral Transfer (Mn_type = 0b011).....	2-78
PLB Slave Buffered Memory to Memory Transfers (Mn_type = 0b100)	2-78
PLB Slave Buffered Peripheral To/From Memory Transfers (Mn_type = 0b101)	2-79
DMA Buffered Memory Transfers (Mn_type = 0b110).....	2-80
PLB Slave Buffered Memory to Memory Transfers With Sideband Signals	
(Mn_type = 0b111).....	2-80
PLB/DMA Example Timing Diagrams.....	2-81
DMA Flyby Memory Read Transfer	2-82
DMA Flyby Memory Write Transfer	2-83
DMA Flyby Burst Memory Read Transfer.....	2-84
DMA Buffered Memory to Peripheral Transfer	2-85
DMA Buffered Peripheral to Memory Transfer	2-86
DMA PLB Slave Buffered Memory To Memory Transfer	2-87
DMA PLB Slave Buffered Peripheral To Memory Transfer	2-88
DMA PLB Slave Buffered Memory To Peripheral Transfer	2-89
Chapter 3. On-Chip Peripheral Bus	3-1
OPB Overview	3-1
Physical Implementation.....	3-2
OPB Signals	3-5
Signal Naming Conventions	3-5
Arbitration Signals	3-7
Mn_request (Master Bus Request).....	3-8
OPB_busLock, Mn_busLock(OPB Bus Arbitration Lock)	3-8
OPB_MnGrant (OPB Master Bus Grant).....	3-8

OPB_timeout (OPB Timeout Error)	3-9
OPB_retry, Sln_retry(OPB Bus Cycle Retry).....	3-9
Bus Signals.....	3-10
OPB_ABus(0:31), Mn_ABus(0:31) (OPB Address Bus).....	3-10
OPB_DBus(0:31), Mn_DBus(0:31), Sln_DBus(0:31)(OPB Data Bus).....	3-10
Data Transfer Control Signals	3-10
OPB_select, Mn_select (OPB Select)	3-10
OPB_RNW, Mn_RNW (OPB Read Not Write)	3-11
OPB_fwXfer, Mn_fwXfer, OPB_hwXfer, Mn_hwXfer (OPB Transfer Size) .	3-11
OPB_seqAddr, Mn_seqAddr (OPB Sequential Address)	3-11
Mn_DBusEn, Sln_DBusEn (Master Data Bus Enable).....	3-12
OPB_xferAck, Sln_xferAck (OPB Transfer Acknowledge)	3-12
OPB_fwAck,Sln_fwAck,OPB_hwAck,Sln_hwAck(OPBTransferSizeAcknowledge)	3-12
OPB_errAck, Sln_errAck (OPB Error Acknowledge).....	3-13
OPB_toutSup, Sln_toutSup (Slave Time-out Suppress)	3-13
Optional DMA Peripheral Support Signals	3-13
Sln_dmaReq (Slave DMA Request)	3-13
DMA_SlnAck (DMA Slave Acknowledge).....	3-13
OPB Interfaces	3-14
OPB Master Interface	3-15
OPB Slave Interface	3-16
OPB Arbiter Interface	3-17
Optional DMA Interface	3-18
OPB Operations	3-19
OPB Bus ArbitrationProtocol	3-20
OPB Basic Bus Arbitration.....	3-20
OPB Bus Arbitration - Continuous Bus Request	3-21
OPB Bus Arbitration - BusLock Signal	3-22
OPB Multiple Bus Master Arbitration	3-23
OPB Bus Master Priority.....	3-24
OPB Bus Parking.....	3-25
Data Transfer Protocol	3-26
OPB Basic Data Transfer	3-26
Overlapped Bus Arbitration	3-29
Continuous Bus Request.....	3-31
Bus Lock Operation	3-32

Sequential Address Signal Operation.....	3-34
Slave Re-try Operation	3-35
Bus TimeOut Error.....	3-37
OPB Bus Timeout Error Condition	3-38
OPB Timeout Error Suppression	3-39
Dynamic Bus Sizing.....	3-40
Data Alignment	3-42
Data Transfer with Dynamic Bus Sizing	3-42
Optional DMA Peripheral Cycle.....	3-52
Chapter 4. Device Control Register Bus	4-1
DCR Overview.....	4-1
DCR Implementation	4-3
DCR Signals	4-5
CPU_dcrWrite (CPU DCR Write)	4-5
CPU_dcrRead (CPU DCR Read)	4-5
CPU_dcrABus(0:9) (CPU DCR Address Bus).....	4-5
CPU_dcrDBusOut(0:31) (CPU DCR Data Bus Out).....	4-6
DCR_cpuAck (DCR CPU Acknowledge)	4-6
DCR_cpuDBusIn(0:31) (DCR CPU Data Bus In)	4-6
DCR Interfaces	4-7
DCR Operations	4-8
Recommended Bus Loading	4-9
Recommended Implementation.....	4-10
CPU Same Speed as DCR Slave.....	4-11
CPU Same Speed as DCR Slave with Longer DCR Access.....	4-12
CPU Two Times Faster than DCR Slave.....	4-13
CPU Slower than DCR Slave	4-14
DCR Implementation in 401 Core.....	4-15
CPU Same Speed as DCR Slave.....	4-15
CPU Same Speed as DCR Slave With Longer DCR Access	4-17
CPU Two Times Faster Than DCR Slave	4-18
CPU Slower than DCR Slave	4-19
Chapter 5. Timing Guidelines.....	5-1
PLB Timing Guidelines	5-1

PLB Master Timing Guidelines	5-2
PLB Arbiter Timing Guidelines	5-2
PLB Slave Timing Guidelines	5-4
OPB Timing Guidelines	5-5
DCR Timing Guidelines	5-7
Chapter 6. Signal Summary	6-1
Index	X-1

Figures

Figure 1-1.	Core+ASIC System Architecture	1-1
Figure 2-1.	PLB Interconnect Diagram	2-3
Figure 2-2.	PLB Address and Data Cycles	2-4
Figure 2-3.	Overlapped PLB Transfers	2-5
Figure 2-4.	Master Interface	2-37
Figure 2-5.	PLB Slave Interface	2-38
Figure 2-6.	PLB Arbiter Interface	2-39
Figure 2-7.	Read Transfers	2-41
Figure 2-8.	Write Transfers	2-42
Figure 2-9.	Transfer Abort	2-43
Figure 2-10.	Back-to-Back Read Transfers	2-44
Figure 2-11.	Back-to-Back Write Transfers	2-45
Figure 2-12.	Back-to-Back Read - Write - Read - Write	2-46
Figure 2-13.	Four Word Line Read	2-47
Figure 2-14.	Four Word Line Write	2-48
Figure 2-15.	Four Word Line Read followed by Four Word Line Write	2-49
Figure 2-16.	Burst Read Transfer Terminated By Master)	2-50
Figure 2-17.	Burst Read Transfer Terminated By Slave	2-51
Figure 2-18.	Burst Write Transfer Terminated by Master	2-52
Figure 2-19.	Burst Write Transfer Terminated By Slave	2-53
Figure 2-20.	Fixed Length Burst Read Transfer	2-56
Figure 2-21.	Fixed Length Burst Write Transfer	2-57
Figure 2-22.	Back-to-Back Burst Read - Burst Write Transfers (Wait = 0, Hold = 0) ...	2-58
Figure 2-23.	Locked Transfer	2-59
Figure 2-24.	Slave Requested Re-arbitration With Bus Un-locked	2-60
Figure 2-25.	Slave Requested Re-arbitration With Bus Locked	2-61
Figure 2-26.	Bus Time-Out Transfer	2-62
Figure 2-27.	Pipelined Back-to-Back Read Transfers	2-65
Figure 2-28.	Pipelined Back-to-Back Read Transfers - Delayed AACK	2-66
Figure 2-29.	Pipelined Back-to-Back Write Transfers	2-67
Figure 2-30.	Pipelined Back-to-Back Write Transfers - Delayed AACK	2-68
Figure 2-31.	Pipelined Back-to-Back Read and Write Transfers	2-69
Figure 2-32.	Pipelined Back-to-Back Read Burst Transfers	2-70
Figure 2-33.	Pipelined Back-to-Back Write Burst Transfers	2-71
Figure 2-34.	PLB/DMA Interface	2-73

Figure 2-35.	DMA Flyby Memory Read Transfer	2-82
Figure 2-36.	DMA Flyby Memory Write Transfer	2-83
Figure 2-37.	DMA Flyby Burst Memory Read Transfer	2-84
Figure 2-38.	DMA Buffered Memory to Peripheral Transfer	2-85
Figure 2-39.	DMA Buffered Peripheral to Memory Transfer	2-86
Figure 2-40.	DMA PLB Slave Buffered Memory to Memory Transfer	2-87
Figure 2-41.	DMA PLB Slave Buffered Peripheral to Memory Transfer	2-88
Figure 2-42.	DMA PLB Slave Buffered Memory to Peripheral Transfer	2-89
Figure 3-1.	Physical Implementation of the OPB Bus	3-2
Figure 3-2.	OPB Master Interface	3-15
Figure 3-3.	OPB Slave Interface	3-16
Figure 3-4.	OPB Arbiter Interface	3-17
Figure 3-5.	Optional DMA Interface	3-18
Figure 3-6.	OPB Basic Bus Arbitration	3-20
Figure 3-7.	OPB Bus Arbitration - Continuous Bus Request	3-21
Figure 3-8.	OPB Bus Arbitration - BusLock Signal	3-22
Figure 3-9.	OPB Multiple Bus Request Arbitration	3-23
Figure 3-10.	Reduced Latency Arbitration using Bus Parking	3-25
Figure 3-11.	OPB Basic Data Transfer	3-26
Figure 3-12.	Fullword - Fullword Read and Write Operation 1	3-27
Figure 3-13.	Fullword - Fullword Read and Write Operation 2	3-28
Figure 3-14.	OPB Data Transfer	3-29
Figure 3-15.	Continuous Bus Request	3-31
Figure 3-16.	Bus Lock Data Transfer Cycle	3-32
Figure 3-17.	Bus Lock Signal Penalty Case	3-33
Figure 3-18.	Sequential Address Signal Operation	3-34
Figure 3-19.	Retry Signal Operation	3-36
Figure 3-20.	Bus Timeout Error Condition	3-38
Figure 3-21.	Timeout Error Suppression	3-39
Figure 3-22.	Attachment Of Bus Devices Of Varying Width	3-42
Figure 3-23.	Fullword - Halfword Read and Write Operation	3-43
Figure 3-24.	Fullword - Byte Read Operation	3-44
Figure 3-25.	Overlapped Arbitration	3-45
Figure 3-26.	Dynamic Bus Sizing with BusLock Signal	3-47
Figure 3-27.	Dynamic Bus Sizing Without Interruption	3-49
Figure 3-28.	Fullword - Byte, Read and Write Operation	3-50
Figure 3-29.	Halfword - Byte, Read and Write Operation	3-51
Figure 3-30.	Optional DMA Peripheral Cycle	3-52
Figure 4-1.	DCR Block Diagram	4-2

Figure 4-2.	DCR Bus Implementation	4-3
Figure 4-3.	CPU Interface	4-7
Figure 4-4.	CPU Same Speed as DCR Slave	4-11
Figure 4-5.	CPU Same Speed as DCR Slave with Longer DCR Access	4-12
Figure 4-6.	CPU Two Times Faster than DCR Slave	4-13
Figure 4-7.	CPU Slower than DCR Slave	4-14
Figure 4-8.	CPU Same Speed As DCR Slave	4-16
Figure 4-9.	CPU Same Speed As DCR Slave With Longer DCR Access	4-17
Figure 4-10.	CPU 2X Faster Than DCR Slave	4-18
Figure 4-11.	CPU Slower Than DCR Slave	4-19

Tables

Table 2-1.	Summary of PLB Signals	2-7
Table 2-2.	Mn_priority(0:1) Request Priority Level	2-12
Table 2-3.	PLB_PAVValid Assertion	2-13
Table 2-4.	PLB_SAVValid Assertion	2-15
Table 2-5.	PLB Master Identification	2-19
Table 2-6.	Byte Enable Signals	2-20
Table 2-7.	Byte Enable Signals During Burst Transfers	2-21
Table 2-8.	PLB Transfer Size Signals	2-22
Table 2-9.	PLB Transfer Type Signals	2-23
Table 2-10.	PLB Address Bus Signal Bits	2-26
Table 2-11.	PLB Read Word Address Signals	2-31
Table 2-12.	Fixed Length Burst Transfer	2-54
Table 2-13.	Summary of DMA Signals	2-74
Table 3-1.	Master Output Connection	3-3
Table 3-2.	Slave Output Connection	3-4
Table 3-3.	Arbiter Output Connection	3-4
Table 3-4.	Summary of OPB Signals	3-6
Table 3-5.	OPB_fwXfer and OPB_hwXfer Encoding	3-11
Table 3-6.	OPB_fwAck and OPB_hwAck Encoding	3-12
Table 3-7.	Summary of Dynamic Bus Sizing	3-40
Table 4-1.	Summary of DCR Signals	4-5
Table 5-1.	PLB Master Signal Timing Guidelines	5-2
Table 5-2.	PLB Arbiter Signal Timing Guidelines	5-2
Table 5-3.	PLB Slave Signal Timing Guidelines	5-4
Table 5-4.	OPB Timing Guidelines	5-5
Table 5-5.	DCR Timing Guidelines	5-7
Table 6-1.	Signal Summary Table	6-1

About This Book

This book begins with an overview of the IBM Core+ASIC system architecture followed by detailed information on high performance and flexible bus structure used in Core+ASIC development.

The IBM Core+ASIC system bus architecture features:

- A high performance 32-bit on-chip processor local bus to interface between processor cores and integrated bus controllers.
- A flexible 32-bit on-chip peripheral bus with dynamic bus sizing for easy connection of on-chip peripheral devices.
- A simple device control register bus to move data between the processor's general purpose registers and slave logic's device control registers.

Who Should Use This Book

This book is for hardware, software, and application developers who need to understand Core+ASIC development. The audience should understand embedded system design, operating systems, and the principles of computer organization.

How This Book is Organized

This book is organized as follows:

- Chapter 1, “Overview”
- Chapter 2, “Processor Local Bus”
- Chapter 3, “On-Chip Peripheral Bus”
- Chapter 4, “Device Control Register Bus”

To help readers find material in these chapters, the book contains:

- Contents on page v
- Figures on page xiii
- Tables on page xvii
- Index on page X-1

Document Conventions

The following is a list of notational conventions frequently used in this book.

$\overline{\text{Active_Low}}$	An overbar indicates an active-low signal.
0x1f	Hexadecimal numbers
0b1001	Binary numbers
CR _{FLD}	The field in the condition register pointed to by a field of an instruction.
24 _S	The sign bit is replicated (sign-extended) 24 times.
xx	Bit positions which are don't-cares.
←	Assignment
^	AND logical operator
¬	NOT logical operator
∨	OR logical operator
⊕	Exclusive-OR (XOR) logical operator
+	Twos complement addition
−	Twos complement subtraction, unary minus
×	Multiplication
÷	Division yielding a quotient
%	Remainder of an integer division; (33 % 32) = 1.

<code> </code>	Concatenation
<code>=, ≠</code>	Equal, not equal relations
<code><, ></code>	Signed comparison relations
<code>≤, ≥</code>	Unsigned comparison relations
<code>if...then...else...</code>	Conditional execution; if <i>condition</i> then <i>a</i> else <i>b</i> , where <i>a</i> and <i>b</i> represent one or more pseudocode statements. Indenting indicates the ranges of <i>a</i> and <i>b</i> . If <i>b</i> is null, the else does not appear.
<code>do</code>	Do loop. “to” and “by” clauses specify incrementing an iteration variable; “while” and “until” clauses specify terminating conditions. Indenting indicates the range of the loop.
<code>leave</code>	Leave innermost do loop or do loop specified in a leave statement.
<code>n</code>	A decimal number
<code>x'n'</code>	A hexadecimal number
<code>b'n'</code>	A binary number
<code>FLD</code>	An instruction field
<code>FLD_b</code>	A bit in an instruction field
<code>FLD_{b:b}</code>	A range of bits in an instruction field
<code>FLD_{b,b,...}</code>	A list of bits, by number or name, in a named field
<code>REG_b</code>	A bit in a named register
<code>REG_{b:b}</code>	A range of bits in a named register
<code>REG_{b,b,...}</code>	A list of bits, by number or name, in a named register
<code>REG[FLD]</code>	A field in a named register
<code>REG[FLD, FLD ...]</code>	A list of fields in a named register
<code>GPR(r)</code>	General Purpose Register <i>r</i> , where $0 \leq r \leq 31$.
<code>(GPR(r))</code>	The contents of General Purpose Register <i>r</i> , where $0 \leq r \leq 31$.
<code>DCR(DCRN)</code>	A DCR specified by the DCRF field in a mfdcr or mtdcr instruction
<code>SPR(SPRN)</code>	An SPR specified by the SPRF field in a mfspr or mtspr instruction
<code>RA, RB, ...</code>	GPRs
<code>(Rx)</code>	The contents of a GPR, where <i>x</i> is A, B, S, or T
<code>(RA 0)</code>	The contents of the register RA or 0, if the RA field is 0.
<code>C_{0:3}</code>	A four-bit object used to store condition results in compare instructions.

n_b	The bit or bit value b is replicated n times.
xx	Bit positions which are don't-cares.
CEIL(x)	Least integer $\geq x$.
EXTS(x)	The result of extending x on the left with sign bits.
PC	Program counter.
RESERVE	Reservation bit; indicates whether a process has reserved a block of storage.
CIA	Current instruction address; the 32-bit address of the instruction being described by a sequence of pseudocode. This address may be used to set the next instruction address (NIA). Does not correspond to any architected register.
NIA	Next instruction address; the 32-bit address of the next instruction to be executed. In pseudocode, a successful branch is indicated by assigning a value to NIA. For instructions that do not branch, the NIA is CIA +4.
MS(addr, n)	The number of bytes represented by n at the location in main storage represented by addr.
EA	Effective address; the 32-bit address, derived by applying indexing or indirect addressing rules to the specified operand, that specifies a location in main storage.
ROTL((RS), n)	Rotate left; the contents of RS are shifted left the number of bits specified by n .
MASK(MB,ME)	Mask having 1's in positions MB through ME (wrapping if MB > ME) and 0's elsewhere.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an effective address.

Instruction Formats

Instructions are four bytes long. Bits of instructions are numbered 0:31, with 0 being the most significant bit (MSB). Instruction addresses are always word-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. The remaining instruction bits contain additional fields. All instruction fields belong to one of the following categories:

- **Defined**

These instruction fields contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

These fields contain operands, such as general purpose register selectors and immediate values, that may vary from execution to execution. The instruction format diagrams specify the operands in variable fields.

- Reserved

Bits in a reserved field should be set to 0. In the instruction format diagrams, reserved fields are shaded.

If any bit in a defined field does not contain the expected value, the instruction is illegal and an illegal instruction exception occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid and its result is architecturally undefined.

Instruction Fields

AA (30)	Absolute address bit.
	<ul style="list-style-type: none"> 0 The immediate field represents an address relative to the current instruction address (CIA). The effective address (EA) of the branch target is either the sum of the LI field sign-extended to 32 bits and the branch instruction address, or the sum of the BD field sign-extended to 32 bits and the branch instruction address. 1 The immediate field represents an absolute address. The EA of the branch target is either the LI field or the BD field, sign-extended to 32 bits.
BA (11:15)	Specifies a bit in the condition register (CR) used as a source of a CR-Logical instruction.
BB (16:20)	Specifies a bit in the CR used as a source of a CR-Logical instruction.
BD (16:29)	An immediate field specifying a 14-bit signed twos complement branch displacement. This field is concatenated on the right with 0b00 and sign-extended to 32 bits.
BF (6:8)	Specifies a field in the CR used as a target in a compare or mcrf instruction.
BFA (11:13)	Specifies a field in the CR used as a source in a mcrf instruction.
BI (11:15)	Specifies a bit in the CR used as a source for the condition of a conditional branch instruction.
BO (6:10)	Specifies options for conditional branch instructions.
BT (6:10)	Specifies a bit in the CR used as a target as the result of a CR-Logical instruction.
D (16:31)	Specifies a 16-bit signed two's-complement integer displacement for load/store instructions.

DCRF (11:20)	Specifies a device control register (DCR).
FXM (12:19)	Field mask used to identify CR fields to be updated by the mtcrf instruction.
IM (16:31)	An immediate field used to specify a 16-bit value (either signed integer or unsigned).
LI (6:29)	An immediate field specifying a 24-bit signed twos complement branch displacement; this field is concatenated on the right with b'00' and sign-extended to 32 bits.
LK (31)	Link bit. 0 Do not update the link register (LR). 1 Update the LR with the address of the next instruction.
MB (21:25)	Mask begin. Used in rotate-and-mask instructions to specify the beginning bit of a mask.
ME (26:30)	Mask end. Used in rotate-and-mask instructions to specify the ending bit of a mask.
NB (16:20)	Specifies the number of bytes to move in an immediate string load or store.
OPCD (0:5)	Primary opcode. Primary opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The OPCODE field name does not appear in instruction descriptions.
OE (21)	Enables setting the OV and SO fields in the fixed-point exception register (XER) for extended arithmetic.
RA (11:15)	A GPR used as a source or target.
RB (16:20)	A GPR used as a source.
Rc (31)	Record bit. 0 Do not set the CR. 1 Set the CR to reflect the result of an operation.
	RS (6:10) A GPR used as a source.
RT (6:10)	A GPR used as a target.
SH (16:20)	An immediate field specifying a shift amount.
SPRF (11:20)	Specifies a special purpose register (SPR).
TO (6:10)	Specifies the conditions on which to trap, as specified in the descriptions of the tw and twi instructions.

- XO (21:30) Extended opcode for instructions without an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.
- XO (22:30) Extended opcode for instructions with an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.

Chapter 1. Overview

This chapter provides an overview of the IBM bus architecture beginning with a system block diagram, and ending with a section on terms and definitions.

The bus architecture provides for high performance and functional integration of advanced features continuously optimized for standard and customized applications.

Figure 1-1 provides a block diagram of the Core+ASIC system architecture.

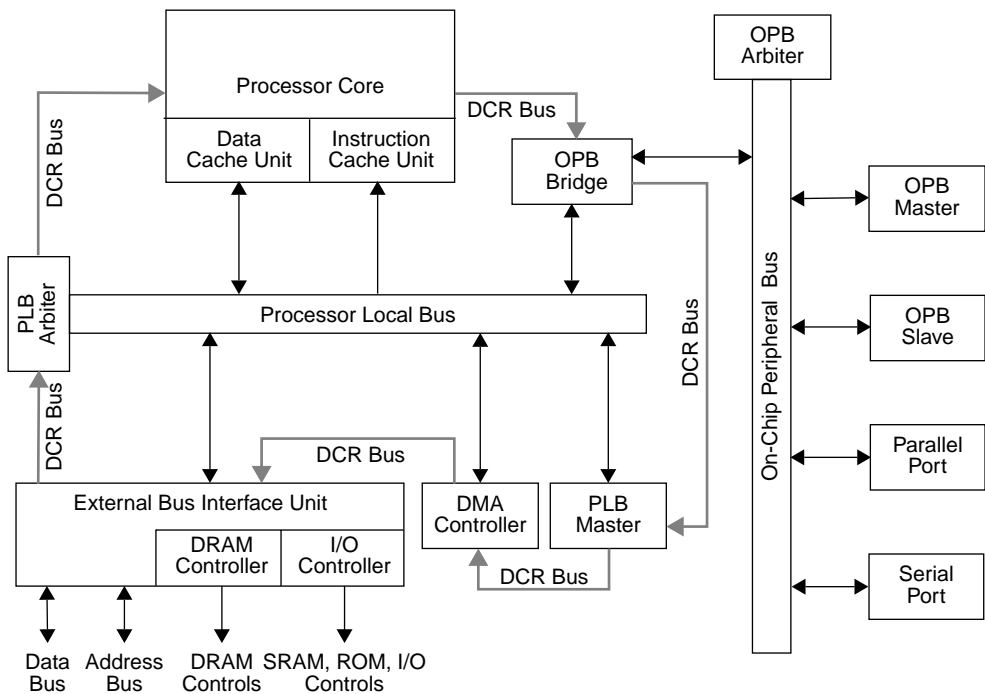


Figure 1-1. Core+ASIC System Architecture

A direct memory access (DMA) controller and an additional processor local bus master is attached to the processor local bus (PLB). PLB slaves in this example consist of an external bus interface unit (EBIU) and an on-chip peripheral bus (OPB) bridge unit. Peripherals such as the serial and parallel port, and others shown here as generic OPB master and slave devices, are connected to the OPB, which is linked to the PLB through the OPB bridge. OPB peripherals may also comprise DMA peripherals. Direct data transfer between OPB peripherals and external resources is supported under DMA control.

This Core+ASIC system architecture allows data transfers among OPB peripherals to occur independently from, and concurrent with, data transfers between the processor and memory, or among other PLB devices.

What follows is a brief overview of the IBM Core+ASIC system bus architecture.

1.1 Processor Local Bus

The processor local bus (PLB) is a high performance 32-bit on-chip bus which provides a standard interface between the processor cores and integrated bus controllers so that a library of processor cores and bus controllers can be developed for use in the Core+ASIC development. The PLB supports read and write data transfers between master and slave devices equipped with a PLB bus interface and connected through PLB signals.

Each PLB master is attached to the PLB through separate address, read data, and write data buses and a plurality of transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address, read data, and write data buses and a plurality of transfer control and status signals for each data bus.

Access to the PLB is granted through a central arbitration mechanism that allows masters to compete for bus ownership. This arbitration mechanism is flexible enough to provide for the implementation of various priority schemes. Additionally, an arbitration locking mechanism is provided to support master-driven atomic operations.

See Chapter 2, "Processor Local Bus," on page 2-1 for details.

1.2 On-Chip Peripheral Bus

The on-chip peripheral bus (OPB) enables easy connection of on-chip peripheral devices. The OPB is a fully synchronous bus which functions independently at a separate level of bus hierarchy. It is, however, not intended to connect directly to the processor core. The processor core can access the peripheral on this bus through the OPB bridge unit.

See Chapter 3, "On-Chip Peripheral Bus," on page 3-1 for details.

1.3 Device Control Register Bus

The device control register (DCR) bus moves data between the CPU's general purpose registers (GPRs) and DCR slave logic's device control registers (DCRs). The DCR bus is a fully synchronous bus. Therefore, it is assumed that in Core+ASIC environment where the CPU and the DCR slave logic are running at different clock speeds, the slower clock's rising edge always corresponds to a faster clock's rising edge.

The DCR bus is typically implemented as a distributed mux across the chip such that each sub-unit not only has a path to place its own DCRs on the CPU's DCR read path, but also has a path which bypasses its DCRs and places another unit's DCRs on the CPU's DCR read path.

See Chapter 4, "Device Control Register Bus," on page 4-1 for details.

1.4 Terms And Definitions

Terms frequently used in this document:

- **Bus Transfer**

A bus transfer is a movement of data from one bus device to another bus device. A bus transfer is defined by the type of data being transferred on the bus (e.g. address transfer, read data transfer, etc.).

- **Bus Transaction**

A bus transaction is a complete exchange between two bus devices. Bus transactions are divided into two parts: supplying the address and receiving or supplying the data.

- **Address Cycle**

During the address cycle the master requests the bus and supplies the address and transfer qualifiers to initiate a bus transaction.

- **Data Cycle**

During the data cycle the master receives or supplies data to complete a bus transaction.

- **Beat**

During a beat exactly one piece of data is exchanged between a master and a slave. A beat may extend across one or more clock cycles.

- **Clock cycle**

A PLB clock period. It starts with the rising edge of the PLB clock.

- **Primary Request**

A primary request is one for which a bus transfer is started while the requested data bus is idle.

- **Secondary Request**

A secondary request is one for which a bus transfer is started before a current bus transfer in the same direction is completed.

- **Primary Address**

The primary address is the address associated with a primary request.

- **Secondary Address**

The secondary address is the address associated with a secondary request.

Chapter 2. Processor Local Bus

The processor local bus (PLB) is designed to interface directly with the processor cores. The primary goal of the PLB is to provide a standard interface between the processor cores and integrated bus controllers such that a library of processor cores and bus controllers can be developed for use in the Core+ASIC development. This document is organized as follows:

- PLB Overview
- PLB Signals
- PLB Interfaces
- PLB Operations
- PLB/DMA Operations

2.1 PLB Overview

The PLB is a high performance 32-bit on-chip bus used in highly integrated Core+ASIC systems. The PLB supports read and write data transfers between master and slave devices equipped with a PLB bus interface and connected through PLB signals.

Each PLB master is attached to the PLB through separate address, read data, and write data buses and a plurality of transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address, read data, and write data buses and a plurality of transfer control and status signals for each data bus.

Access to the PLB is granted through a central arbitration mechanism that allows masters to compete for bus ownership. This arbitration mechanism is flexible enough to provide for the implementation of various priority schemes. Additionally, an arbitration locking mechanism is provided to support master-driven atomic operations.

The PLB is a fully-synchronous bus. Timing for all PLB signals is provided by a single clock source which is shared by all masters and slaves attached to the PLB.

2.1.1 PLB Features

The PLB addresses the high performance and design flexibility needs of highly integrated Core+ASIC systems.

2.1.1.1 High Performance

PLB features in this category include:

- Overlapping of read and write transfers allows two data transfers per clock cycle for maximum bus utilization.
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth.
- Address pipelining reduces overall bus latency by allowing the latency associated with a new request to be overlapped with an ongoing data transfer in the same direction.
- Late master request abort capability reduces latency associated with aborted requests.
- Hidden (overlapped) bus request/grant protocol reduces arbitration latency.
- Fully synchronous bus.

2.1.1.2 System Design Flexibility

PLB features in this category include:

- Bus architecture supports sixteen masters and any number of slave devices.
- Four levels of request priority for each master allow PLB implementations with various arbitration schemes.
- Bus arbitration-locking mechanism allows for master-driven atomic operations.
- Byte-enable capability allows for unaligned halfword transfers and 3-byte transfers.
- Support for 16-, 32-, and 64-byte line data transfers.
- Read word address capability allows slave devices to fetch line data in any order (that is, target-word-first or sequential).
- Sequential burst protocol allows byte, halfword, and word burst data transfers in either direction.
- Guarded and unguarded memory transfers allow a slave device to enable or disable the prefetching of instructions or data.
- DMA buffered, flyby, peripheral to memory, memory to peripheral, and DMA memory to memory operations are also supported.

2.1.2 PLB Implementation

The PLB implementation consists of a PLB macro to which all masters and slaves are attached. The logic within the PLB macro consists of a central bus arbiter and the necessary bus control and gating logic.

The PLB architecture supports up to sixteen master devices. However, PLB macro implementations supporting less than sixteen masters are allowed. The PLB architecture also supports any number of slave devices. However, it should be noted that the number of masters and slaves attached to a PLB macro in a particular system will directly affect the performance of the PLB macro in that system.

Figure 2-1 shows an example of the PLB connections for a system with three masters and three slaves.

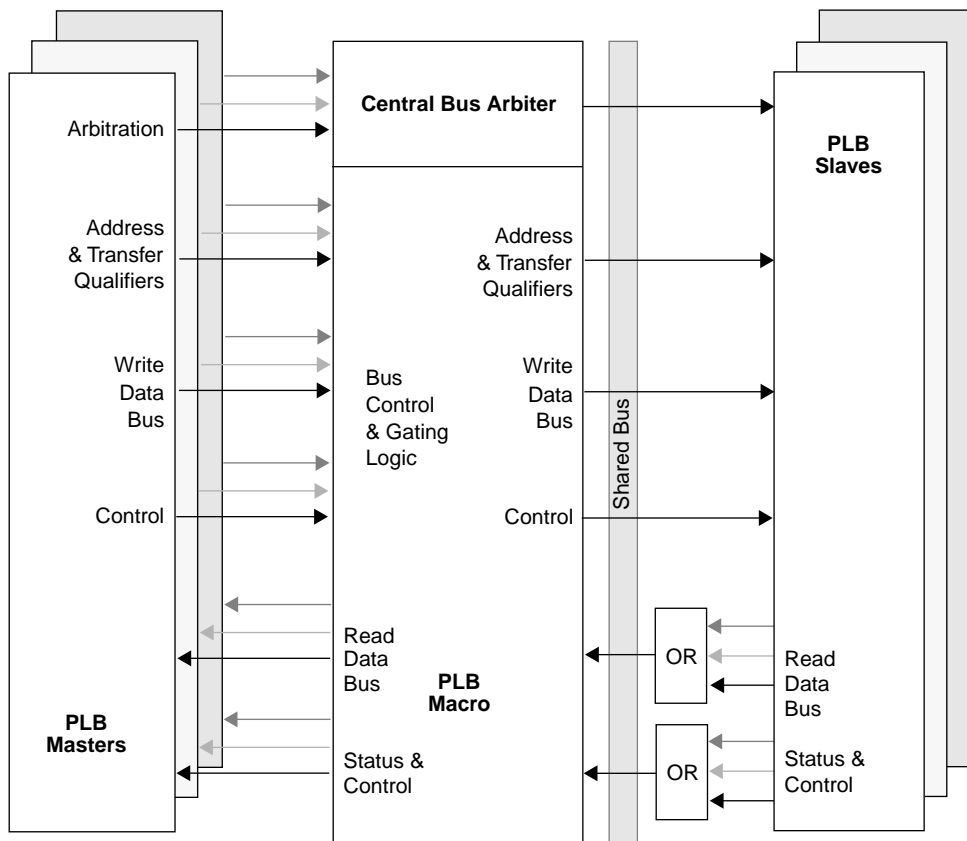


Figure 2-1. PLB Interconnect Diagram

2.1.3 PLB Transfer Protocol

A PLB bus transaction as shown in Figure 2-2 is grouped under an address cycle and a data cycle.

The address cycle has three phases: request, transfer, and address acknowledge. A PLB bus transaction begins when a master drives its address and transfer qualifier signals and requests ownership of the bus during the request phase of the address cycle. Once bus ownership has been granted by the PLB arbiter, the master's address and transfer qualifiers are presented to the slave devices during the transfer phase.

During normal operation, the address cycle is terminated by a slave latching the master's address and transfer qualifiers during the address acknowledge phase.

Each data beat in the data cycle has two phases: transfer and data acknowledge. During the transfer phase the master will drive the write data bus for a write transfer or sample the read data bus for a read transfer. Data acknowledge signals are required during the data acknowledge phase for each data beat in a data cycle.

Note: For a single-beat transfer, the data acknowledge signals also indicate the end of the data transfer. For line or burst transfers, the data acknowledge signals apply to each individual beat and indicate the end of the data cycle only after the final beat.

Figure 2-2 demonstrates PLB address and data cycles.

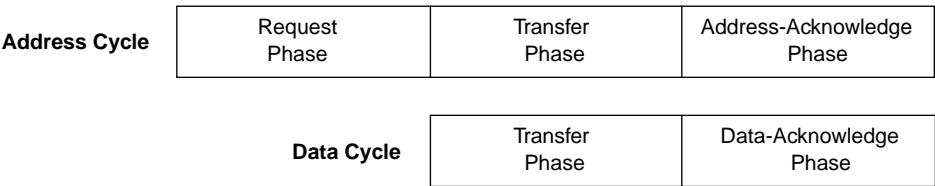


Figure 2-2. PLB Address and Data Cycles

2.1.4 Overlapped PLB Transfers

Figure 2-3 shows an example of overlapped PLB transfers. PLB address, read data, and write data buses are decoupled from one another allowing for address cycles to be overlapped with read or write data cycles, and for read data cycles to be overlapped with write data cycles. The PLB split-bus transaction capability allows the address and data buses to have different masters at the same time.

PLB address pipelining capability allows a new bus transfer to begin before the current transfer has been completed. Address pipelining reduces overall bus latency on the PLB by allowing the latency associated with a new transfer request to be overlapped with an ongoing data transfer in the same direction.

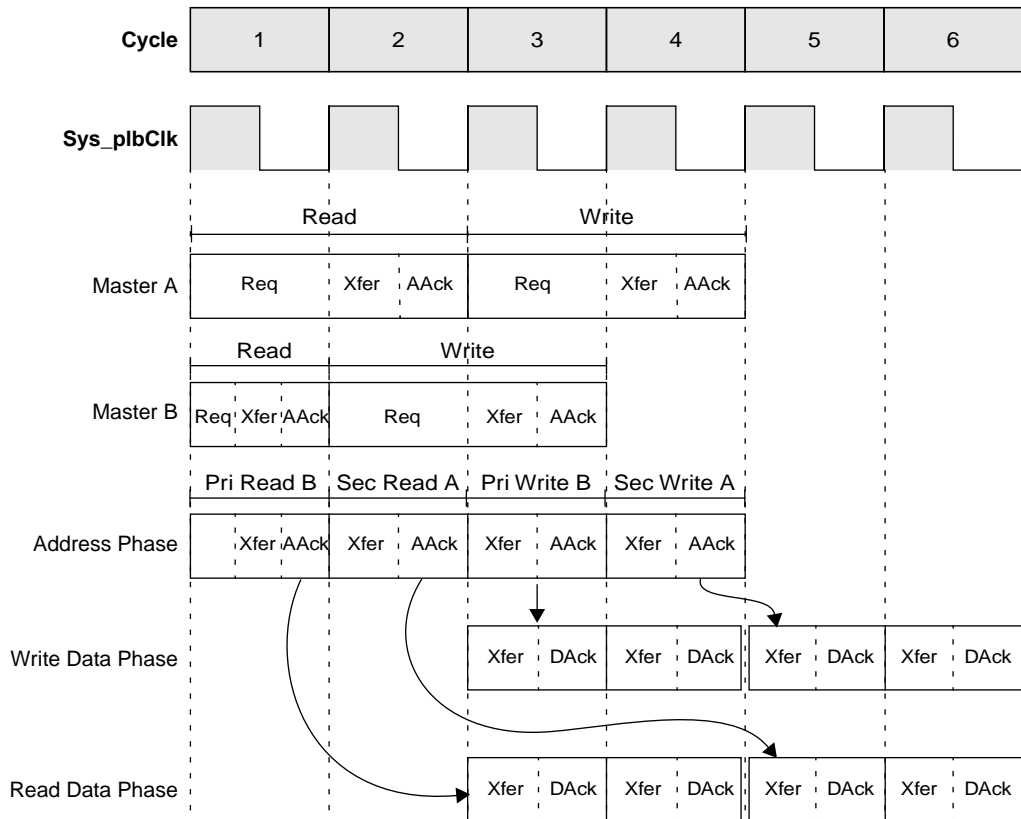


Figure 2-3. Overlapped PLB Transfers

Note: A master may begin to request ownership of the PLB in parallel with the address cycle and/or data cycle of another master's bus transfer. Overlapped read and write data transfers and split-bus transactions allow the PLB to operate at a very high bandwidth.

2.2 PLB Signals

PLB signals can be grouped under the following categories:

- PLB System Signals
- PLB Arbitration Signals
- PLB Status Signals
- PLB Transfer Qualifier Signals
- PLB Write Data Bus Signals
- PLB Read Data Bus Signals
- Additional Slave Output Signals

2.2.1 Signal Naming Conventions

The PLB implementation consists of a PLB macro to which all masters and slaves are attached. The logic within the PLB macro consists of a central bus arbiter and the necessary bus control and gating logic. Slaves are attached to the PLB macro on a shared bus and use the following naming convention:

- Signals which are outputs of the PLB macro and inputs to the slave devices are prefixed with “PLB_”. There will only be one output of the PLB macro for each one of these signals and it will be received as an input by each slave attached to the PLB macro. For example, PLB_PAValiD is an output of the PLB macro and is an input to each slave attached to the PLB macro.
- Signals which are outputs of the slaves and inputs to the PLB macro are prefixed with “SI_”. Each slave will have its own output which is then logically or’ed together at the chip level to form a signal input to the PLB macro. The slaves must ensure that these signals are driven to a logic ‘0’ when they are not involved in a transfer on the PLB.

Each master is attached directly to the PLB macro with its own address, read data, and write data bus signals which use the following naming convention:

- Signals which are driven by a master as an input to the PLB macro are pre-fixed with “Mn_”. There may be as many as sixteen masters which all have their own set of PLB input signals. For example, the Mn_request signal, when implemented would result in M0_request, M1_request, thru M15_request.
- Signals which are driven by the PLB macro to a master have a prefix PLB_Mn to indicate that this signal is connected from the PLB macro to a specific master (that is, PLB_MnAddrAck). The PLB macro provides a maximum of sixteen outputs for this signal, one for each master attached on the bus. For example, the PLB_MnAddrAck signal, when implemented would result in PLB_M0AddrAck, PLB_M1AddrAck, thru PLB_M15AddrAck.

Note: The PLB specification uses “SI” and “Mn” in reference to a slave and master outputs only for the purpose of maintaining clarity, and consistency, throughout the documentation. In actual designs, slave and master outputs must be prefixed by a 3-letter

qualifier identifying the unit. In its current version, the PLB specification allows a maximum of sixteen masters. However, this does not preclude the implementation of PLB macros capable of supporting less than sixteen masters.

Table 2-1 provides a summary of all PLB input/output signals in alphabetical order, the interfaces under which they are grouped, followed by a brief description and page reference for detailed functional description.

Table 2-1. Summary of PLB Signals

Signal Name	Interface	I/O	Description	Page
Mn_request	Master n	I	Master n bus request	2-11
Mn_abort	Master n	I	Master n abort bus request indicator	2-18
Mn_priority(0:1)	Master n	I	Master n bus request priority	2-11
Mn_busLock ¹	Master n	I	Master n bus lock	2-12
Mn_RNW	Master n	I	Master n read/write	2-19
Mn_BE(0:3)	Master n	I	Master n byte enables	2-20
Mn_size(0:3)	Master n	I	Master n transfer size	2-22
Mn_type(0:2)	Master n	I	Master n transfer type	2-23
Mn_compress	Master n	I	Master n compressed data transfer indicator	2-24
Mn_guarded	Master n	I	Master n guarded transfer indicator	2-24
Mn_ordered	Master n	I	Master n synchronize transfer indicator	2-25
Mn_lockErr	Master n	I	Master n lock error indicator	2-25
Mn_ABus(0:31)	Master n	I	Master n address bus	2-26
Mn_wrDBus(0:31)	Master n	I	Master n write data bus	2-27
Mn_wrBurst	Master n	I	Master n burst write transfer indicator	2-28
Mn_rdBurst	Master n	I	Master n burst read transfer indicator	2-32
PLB_MnAddrAck	Master n	O	PLB master n address acknowledge	2-17
PLB_MnRearbitrate	Master n	O	PLB master n bus rearbitrate indicator	2-17
PLB_Mn_Busy	Master n	O	PLB master n slave busy indicator	2-34
PLB_Mn_Err	Master n	O	PLB master n slave error indicator	2-34
PLB_Mn_WrDAck	Master n	O	PLB master n write data acknowledge	2-28

Table 2-1. Summary of PLB Signals (cont.)

Signal Name	Interface	I/O	Description	Page
PLB_Mn_WrBTerm	Master n	O	PLB master n terminate write burst indicator	2-29
PLB_MnRdDBus(0:31)	Master n	O	PLB master n read data bus	2-30
PLB_MnRdWdAddr(0:3)	Master n	O	PLB master n read word address	2-31
PLB_MnRdDAck	Master n	O	PLB master n read data acknowledge	2-31
PLB_MnRdBTerm	Master n	O	PLB master n terminate read burst indicator	2-33
PLB_masterID(0:1)	Arbiter	O	PLB current master identifier	2-19
PLB_PAValid	Arbiter	O	PLB primary address valid indicator	2-12
PLB_SAValid	Arbiter	O	PLB secondary address valid indicator	2-15
PLB_pendReq	Arbiter	O	PLB pending bus request indicator	2-18
PLB_abort	Arbiter	O	PLB abort bus request indicator	2-18
PLB_reqPri(0:1)	Arbiter	O	PLB current request priority	2-19
PLB_pendPri(0:1)	Arbiter	O	PLB pending request priority	2-18
PLB_busLock	Arbiter	O	PLB bus lock	2-12
PLB_RNW	Arbiter	O	PLB read not write	2-19
PLB_BE(0:3)	Arbiter	O	PLB byte enables	2-20
PLB_size(0:3)	Arbiter	O	PLB transfer size	2-22
PLB_type(0:2)	Arbiter	O	PLB transfer type	2-23
PLB_compress	Arbiter	I	PLB compressed data transfer indicator	2-24
PLB_guarded	Arbiter	O	PLB guarded transfer indicator	2-24
PLB_ordered	Arbiter	O	PLB synchronize transfer indicator	2-25
PLB_lockErr	Arbiter	O	PLB lock error indicator	2-25
PLB_ABus(0:31)	Arbiter	O	PLB address bus	2-26
PLB_wrDBus(0:31)	Arbiter	O	PLB write data bus	2-27
PLB_wrBurst	Arbiter	O	PLB burst write transfer indicator	2-28
PLB_rdBurst	Arbiter	O	PLB burst read transfer indicator	2-32

Table 2-1. Summary of PLB Signals (cont.)

Signal Name	Interface	I/O	Description	Page
PLB_wrPrim	Arbiter	O	PLB secondary to primary write request indicator	2-29
PLB_rdPrim	Arbiter	O	PLB secondary to primary read request indicator	2-33
SI_addrAck	Slave	I	Slave address acknowledge	2-17
SI_rearbitrate	Slave	I	Slave rearbitrate bus indicator	2-17
SI_wait	Slave	I	Slave wait indicator	2-16
SI_rdComp	Slave	I	Slave read transfer complete indicator	2-32
SI_rdDAck	Slave	I	Slave read data acknowledge	2-31
SI_rdBTerm	Slave	I	Slave terminate read burst transfer	2-33
SI_rDBus(0:31)	Slave	I	Slave read data bus	2-30
SI_rdWdAddr(0:3)	Slave	I	Slave read word address	2-31
SI_wrComp	Slave	I	Slave write transfer complete indicator	2-28
SI_wrDAck	Slave	I	Slave write data acknowledge	2-28
SI_wrBTerm	Slave	I	Slave terminate write burst transfer	2-29
SI_MBusy(0:3)	Slave	I	Slave busy indicator	2-34
SI_MErr(0:3)	Slave	I	Slave error indicator	2-34
Sys_plbClk	System	I	System C2 clock	2-10
Sys_plbReset	System		System PLB reset	2-10

2.2.2 PLB System Signals

Two PLB system signals have been defined: Sys_plbClk and Sys_plbReset.

2.2.2.1 Sys_plbClk (System PLB Clock)

This signal provides the timing for the PLB and is an input to all PLB masters and slaves, as well as the PLB arbiter. All PLB master, slave, and arbiter output signals are asserted/negated relative to the rising edge of Sys_plbClk and all PLB master, slave, and arbiter input signals are sampled relative to this edge.

Note: The master and slave attached to the PLB are expected to operate at the frequency of the PLB. Thus, any speed matching that is required due to I/O constraints or units that run at different frequencies will be handled in the PLB interfaces of masters and slaves. Processor cores which run at speeds significantly greater than that of the PLB will require synchronization logic to be inserted either within the core or between the core and the PLB.

2.2.2.2 Sys_plbReset (System PLB Reset)

This signal is the PLB arbiter's power-on reset signal. This signal can also be used to bring the PLB to an idle or quiescent state. The PLB idle state is defined as the bus state in which:

- No bus requests (read or write) are pending (that is, all Mn_request signals are negated).
- The bus is not locked (that is, all Mn_busLock signals, and PLB_busLock, are negated).
- The bus is not granted or being granted to any master (that is, PLB_PAVailid is negated).
- The read and write data buses are not being used (that is, all SI_rdDack and SI_wrDack signals are negated and all SISI_rdDbus(0:31) and SI_rdWdAddr(0:3) signals are driven to a logic "0").

This signal must only be asserted, or negated, relative to the rising edge of Sys_plbClk. How long this signal must be kept asserted when forcing the PLB to an idle state in a system will depend on the actual implementation of that system's PLB arbiter, master, and slave devices.

Note: In addition to the SYS_plbReset input, a PLB master may have other means by which it can force itself into a reset state but without affecting the state of other masters and slaves attached to the PLB, or the PLB arbiter. However, if currently involved in a PLB transfer, the master must allow for the transfer to be completed, or properly terminate it by using Mn_abort. Otherwise, if a master's request is acknowledged by a slave, and the master wishes to enter its reset state before all the data associated with that request is transferred, then the master must be tolerable of receiving the data acknowledges while entering, during, and after the reset state. Furthermore, the master must negate the Mn_busLock and Mn_rdBurst signals if currently asserted.

2.2.3 PLB Arbitration Signals

The PLB address cycle consists of three phases: request, transfer, and termination. During the request phase, the Mn_request, Mn_priority, and Mn_busLock signals are used to compete for the ownership of the bus.

Once the PLB arbiter has granted the bus to a master, the master's address and transfer qualifier signals are presented to the PLB slaves during the transfer phase. The transfer phase is marked by the PLB arbiter's assertion of the PLB_PValid or PLB_SValid signal. The maximum length of the transfer phase is controlled by the slave's SI_wait signal and by the PLB arbiter address cycle time-out mechanism.

During the termination phase, the address cycle is completed by the slave through the SI_addrAck or SI_rearbitrate signals, or by the master through the Mn_abort signal, or by the PLB timing out.

Note: It is possible for all three phases of the address cycle to occur in a single PLB clock cycle.

2.2.3.1 Mn_request (Bus Request)

This signal is asserted by the master to request a data transfer across the PLB. Once Mn_request has been asserted, this signal, the address, and all of the transfer qualifiers must retain their values until:

- the slave has terminated the address cycle through the assertion of SI_addrAck (PLB_MnAddrAck) or SI_rearbitrate (PLB_MnRearbitrate), or
- the master has aborted the request through the assertion of Mn_abort, or
- the PLB arbiter has asserted PLB_MnAddrAck in the event of a time-out.

Once the address cycle has been properly terminated, the master may continue to assert Mn_request if another transfer is required across the PLB. In this case, the master address and transfer qualifiers will be updated in the clock cycle following the assertion of PLB_MnAddrAck, PLB_MnRearbitrate, or Mn_abort, to reflect the new request. If there are no other master requests pending, Mn_request should be negated in the clock cycle following the assertion of PLB_MnAddrAck, PLB_MnRearbitrate, or Mn_abort.

This signal must be negated in response to the assertion of SYS_plbReset.

2.2.3.2 Mn_priority(0:1) (Request Priority)

These signals are driven by the master to indicate to the PLB arbiter the priority of the master's request and are valid any time the Mn_request signal is asserted.

Note: It is permissible for the value of the Mn_priority(0:1) signals to change at any time during the address cycle and prior to the slave asserting SI_addrAck or SI_rearbitrate, or the master aborting the request through Mn_abort.

The PLB arbiter uses these signals in conjunction with the other master priority signals to determine which request should be granted and then presented to the PLB slaves. Table 2-2 shows Mn_priority(0:1) request priority level.

Table 2-2. Mn_priority(0:1) Request Priority Level

Mn_priority(0:1)	Priority Level
11	Highest
10	Next highest
01	Next highest
00	Lowest

2.2.3.3 Mn_busLock, PLB_busLock (Bus Arbitration Lock)

This signal is asserted by the master with the Mn_request signal and is sampled by the PLB arbiter in the clock cycle in which the SI_addrAck signal is asserted by the slave. This signal may be used by the current master to lock the arbitration and force the PLB arbiter to continue to grant the bus to that master and ignore all other requests that are pending. The PLB may only be locked by requesting a transfer with the Mn_busLock signal asserted and being the highest priority request presented to the PLB arbiter.

Once the bus has been successfully locked by the current master, it is not necessary for that master to continuously drive the request signal asserted. If the master negates Mn_request, but does not negate Mn_busLock, the bus will continue to be locked to that master and will remain locked until the master negates Mn_busLock. More specifically, the bus will continue to be locked with the current master until that master has negated its Mn_busLock signal for one complete clock cycle. On the clock cycle following the negation of the Mn_busLock signal, if there are no transfers in progress, the PLB arbiter will again re-arbitrate and grant the bus to the highest priority request.

Note: A master request with the Mn_busLock signal asserted is a special case in that the PLB arbiter will wait for both the read data bus and the write data bus to be available prior to granting the PLB to a master and presenting that master's address and transfer qualifiers to the slaves. Please refer to Section 2.2.3.4, "PLB_PAVAlid (PLB Primary Address Valid)," on page 2-12 and Section 2.2.3.5, "PLB_SAVAlid (Secondary Address Valid)," on page 2-15 for more detailed information on the PLB arbiter's handling of a master request with the Mn_busLock signal asserted.

This signal must be negated in response to the assertion of SYS_plbReset.

2.2.3.4 PLB_PAVAlid (PLB Primary Address Valid)

This signal is asserted by the PLB arbiter in response to the assertion of Mn_request and to indicate that there is a valid primary address and transfer qualifiers on the PLB outputs. The cycle in which PLB_PAVAlid is asserted, (relative to the assertion of Mn_request), is determined by the direction in which data is to be transferred, the current state of the data

buses, and the state of the Mn_busLock signal. The relationship between these three factors and the assertion of the PLB_PAVali d signal is summarized in Table 2-3.

Table 2-3. PLB_PAVali d Assertion

Current direction of data transfer requested	Requesting Bus Lock Y/N	Current state of read data bus	Current state of write data bus	The clock cycle in which PLB_PAVali d is asserted relative to the assertion of Mn_request
Read	N	Idle	Don't care	In the clock cycle Mn_request is first asserted
Read	N	Busy	Don't care	In the clock cycle SI_rdComp is asserted if PLB_rdPrim is not also asserted
Write	N	Don't care	Idle	In the clock cycle Mn_request is first asserted
Write	N	Don't care	Busy	In the clock cycle following the clock cycle SI_wrComp is asserted if PLB_wrPrim is not also asserted.
Read/Write	Y	Idle	Idle	In the clock cycle Mn_request is first asserted
Read/Write	Y	Idle	Busy	In the clock cycle following the clock SI_wrComp is asserted if PLB_wrPrim is not also asserted
Read/Write	Y	Busy	Idle	In the clock cycle SI_rdComp is asserted if PLB_rdPrim is not also asserted
Read/Write	Y	Busy	Busy	In the clock cycle SI_rdComp is asserted, or the clock cycle following the clock cycle SI_wrComp is asserted, whichever is later, provided PLB_rdPrim and PLB_wrPrim are not also asserted.

Note: For read data bus, the busy state corresponds to the window of time starting with the clock cycle following the assertion of SI_addrAck and ending with the assertion of SI_rdComp. For the write data bus, the busy state corresponds to the window of time starting with the clock cycle following the assertion of SI_addrAck and ending with the clock cycle in which SI_wrComp is asserted.

All slaves should sample the PLB_PAValiD signal and if asserted, and the address is within their address range and they are capable of performing the transfer, they should respond by asserting their SI_addrAck signal. If a slave detects a valid primary address on the PLB but is unable to latch the address and transfer qualifiers, or perform the requested transfer, then it should either assert the SI_wait signal to require the PLB arbiter to wait for the request to be properly terminated, or assert the SI_rearbitrate signal to require the PLB arbiter to re-arbitrate the bus.

Note 1: Once PLB_PAValiD has been asserted, the PLB arbiter will wait for sixteen clock cycles for the request to be properly terminated. If no slave responds with SI_wait, SI_AddrAck, or SI_rearbitrate, or the request is not aborted by the master, by the sixteenth clock cycle, the PLB arbiter will time-out and complete the necessary handshaking to the master as well as assert the PLB_MErr signal (see section 2.4.1.22, "Bus Transfer Time-Out Notes," on p. 2-63 for more detailed information on the handshaking provided by the PLB arbiter).

Note 2: Once the PLB_PAValiD signal is asserted, the PLB arbiter will not re-arbitrate the bus until either SI_rearbitrate, or SI_addrAck, or Mn_abort, is asserted, or the PLB arbiter times-out.

Note 3: Once a slave has asserted the SI_addrAck signal the PLB arbiter will wait indefinitely for the slave to assert the read or write complete signal. It is up to the slave design to ensure that a deadlock does not occur on the bus due to an address acknowledge occurring without the corresponding data acknowledge(s).

This signal must be negated in response to the assertion of SYS_plbReset.

2.2.3.5 PLB_SAVValid (Secondary Address Valid)

This signal is asserted by the PLB arbiter to indicate to a PLB slave that there is a valid secondary address and transfer qualifiers on the PLB outputs. The clock cycle in which PLB_SAVValid is asserted, (relative to the assertion of Mn_request), is determined by the direction in which data is to be transferred, the current state of the data buses, and the state of the Mn_busLock signal. The relationship between these three factors and the assertion of the PLB_SAVValid signal is summarized in Table 2-4 below..

Table 2-4. PLB_SAVValid Assertion

Current direction of data transfer requested	Requesting Bus Lock Y/N	Current state of read data bus	Current state of write data bus	The clock cycle in which PLB_SAVValid is asserted relative to the assertion of Mn_request
Read	N	Idle	Don't care	PLB_SAVValid is not asserted. The request is considered a primary request.
Read	N	Busy	Don't care	In the clock cycle Mn_request is first asserted if a secondary request has not been previously acknowledged or, in the clock cycle PLB_rdPrim is asserted if otherwise.
Write	N	Don't care	Idle	PLB_SAVValid is not asserted. The request is considered a primary request.
Write	N	Don't care	Busy	In the clock cycle Mn_request is first asserted if a secondary write request has not been previously acknowledged or, in the clock cycle following the assertion of PLB_wrPrim if a secondary write request has been previously acknowledged.
Don't care	Y	Don't care	Don't care	PLB_SAVValid is not asserted.

Note: For read data bus, the busy state corresponds to the window of time starting the clock cycle following the assertion of SI_addrAck and ending with the clock cycle in which SI_rdComp is asserted. For the write data bus, the busy state corresponds to the window of time starting the clock cycle following the assertion of SI_addrAck and ending the clock cycle in which SI_wrComp is asserted.

Once PLB_SAVValid has been asserted for a secondary read request, the PLB arbiter will wait indefinitely for either of the following conditions to occur:

1. SI_addrAck is asserted by a slave, or
2. The request is aborted by the requesting master, or
3. SI_rdComp is asserted for the primary read request.

In the first and second case, the PLB arbiter will re-arbitrate the bus in the following clock cycle. In the third case, the PLB arbiter will not re-arbitrate the bus but instead will negate PLB_SAVValid and assert PLB_PAVValid, in the same clock cycle SI_rdComp is asserted, to indicate that there is a new valid primary address and transfer qualifiers on the PLB outputs.

Once PLB_SAVValid has been asserted for a secondary write request, the PLB arbiter will wait indefinitely for either of the following conditions to occur:

1. SI_addrAck is asserted by a slave, or
2. The request is aborted by the requesting master, or
3. SI_wrComp is asserted for the primary write request.

In the first and second cases, the PLB arbiter will re-arbitrate the bus in the following clock cycle. In the third case, the PLB arbiter will not re-arbitrate the bus but instead will negate the PLB_SAVValid signal and assert the PLB_PAVValid signal, in the clock cycle following the assertion of SI_wrComp, to indicate that there is a new valid primary address and transfer qualifiers on the PLB outputs.

Note: It is not possible for a secondary request to time-out on the PLB. Accordingly, if a slave detects a valid secondary address on the PLB but is unable to latch the address and transfer qualifiers, or perform the requested transfer, it is not necessary for the slave to assert SI_wait or SI_rearbitrate signals since these will be ignored by the PLB arbiter until the secondary request has become a primary request, PLB_SAVValid has been negated, and PLB_PAVValid has been asserted.

This signal must be negated in response to the assertion of SYS_plbReset.

2.2.3.6 SI_wait (Wait for Address Acknowledge)

This signal is asserted to indicate that the slave has recognized the PLB address as a valid address, but is unable to latch the address and all of the transfer qualifiers at the end of the current clock cycle. The slave may assert this signal anytime it recognizes a valid address and type on the PLB and it is not required to negate it before asserting SI_addrAck or SI_rearbitrate.

Note: The PLB arbiter will qualify the SI_wait signal with PLB_PAVValid and thus the slaves are not required to qualify the assertion of SI_wait with PLB_PAVValid.

When asserted in response to the assertion of PLB_PAVValid, the PLB arbiter will use this signal to disable its address cycle time-out mechanism and wait indefinitely for the slave to assert its SI_addrAck or SI_rearbitrate signals. Otherwise, the PLB arbiter will wait a maximum of sixteen clock cycles for SI_addrAck or SI_rearbitrate to be asserted before timing out.

When asserted in response to the assertion of PLB_SAVValid, the PLB arbiter will ignore this signal and wait indefinitely for the slave to assert its SI_addrAck signal, or the master to abort the request, or for the secondary request to become a primary request.

The SI_wait signal is an input to the PLB arbiter only, and is not driven back to the PLB masters.

2.2.3.7 SI_addrAck, PLB_MnAddrAck (Address Acknowledge)

This signal is asserted to indicate that the slave has acknowledged the address and will latch the address and all of the transfer qualifiers at the end of the current clock cycle. This signal is asserted by the slave only while PLB_PAVValid or PLB_SAVValid are asserted and should remain negated at all other times.

Note: It is possible for the slave to acknowledge a valid address in the same clock cycle in which PLB_PAVValid or PLB_SAVValid are first asserted.

2.2.3.8 SI_rearbitrate, PLB_MnRearbitrate (Rearbitrate PLB)

This signal is asserted to indicate that the slave is unable to perform the currently requested transfer and require the PLB arbiter to re-arbitrate the bus. This signal is asserted by the slave only while PLB_PAVValid or PLB_SAVValid are asserted and should remain negated at all other times.

When asserted in response to the assertion of PLB_PAVValid, the PLB arbiter will pass this signal to the masters and re-arbitrate and grant the bus to the highest priority request in the next clock cycle. Furthermore, to avoid a possible deadlock scenario, the PLB arbiter will ignore the original master request during re-arbitration.

When asserted in response to the assertion of PLB_SAVValid, the PLB arbiter will not pass this signal to the masters or re-arbitrate the bus. Instead, the PLB arbiter will wait for the current request to become a primary request, and PLB_PAVValid to be asserted, and then sample SI_wait, SI_addrAck, SI_rearbitrate, and Mn_abort.

Note 1: If SI_addrAck and SI_rearbitrate are sampled asserted in the same clock cycle, the PLB arbiter will ignore the SI_rearbitrate signal and will not re-arbitrate the bus. As a result, the slave must perform the requested data transfer (read or write) to avoid a possible deadlock scenario.

Note 2: If the bus had been previously locked, the SI_rearbitrate signal will be ignored by the PLB arbiter to prevent the interruption of an "atomic" operation. Hence, to prevent a deadlock scenario in this case, the locking master must negate its Mn_request and Mn_busLock signals for a minimum of two clock cycles following the sampling of PLB_MnRearbitrate asserted. See section 2.4.1.20, "Slave Requested Re-arbitration With Bus Locked," on p. 2-61 for detailed information.

2.2.3.9 Mn_abort, PLB_abort (Abort Request)

This signal is asserted by the master to indicate that it no longer requires the data transfer it is currently requesting. This signal is only valid while the Mn_request signal is asserted and may only be used to abort a request which has not been acknowledged or is being acknowledged in the current clock cycle. In the clock cycle following the assertion of Mn_abort, the master should either negate Mn_request or make a new request. However, starting in the clock cycle following the assertion of SI_addrAck by the slave, a request may no longer be aborted by the master and the slave is required to perform the necessary handshaking to complete the transfer. This signal will have a minimum amount of set-up time to allow for its assertion late in the clock cycle.

Note 1: A slave may assert SI_wrDack and SI_wrComp with SI_addrAck for a primary write request, even if the request is being aborted by the master in the same clock cycle. In this case, the master is required to ignore these signals and no data will be stored by the slave.

Note 2: If SI_rearbitrate is asserted in the same clock cycle as Mn_abort, the PLB arbiter will ignore the SI_rearbitrate signal and the master will not be “backed-off” during re-arbitration. In the clock cycle following the assertion of Mn_abort, the PLB arbiter will re-arbitrate and grant the highest priority request.

The PLB_abort signal is sampled by the slaves only while PLB_PAVvalid or PLB_SAVvalid are asserted.

2.2.4 PLB Status Signals

PLB status signals are driven by the PLB arbiter and reflect the PLB master ownership status. These signals can be used by PLB masters and slave devices to help resolve arbitration on the PLB or other buses attached to the PLB via a bridge or cross-bar switch.

2.2.4.1 PLB_pendReq (PLB Pending Bus Request)

This signal is asserted by the PLB arbiter to indicate that a master has a request pending on the PLB. This signal is a combined logic ‘OR’ of all the master request inputs. This signal may be sampled by any PLB master or slave and can be used by itself, or in conjunction with the PLB_pendPri(0:1) signals, to determine when to negate the Mn_busLock or Mn_rdBurst and Mn_wrBurst signals due to another master requesting the bus.

This signal is always valid and will not be negated during a clock cycle in which a request is being aborted by the master.

2.2.4.2 PLB_pendPri(0:1) (Pending Request Priority)

These signals are driven by the PLB arbiter and are valid any time the PLB_pendReq signal is asserted. These signals indicate the highest priority of any pending request input from all masters attached to the PLB. Only the priority inputs of masters with their respective Mn_request signals asserted may be used in determining the PLB_pendPri outputs. These signals may be used by masters to determine when to negate the Mn_busLock or

Mn_rdBurst and Mn_wrBurst signals due to another master requesting a higher priority request.

2.2.4.3 PLB_reqPri(0:1) (Current Request Priority)

These signals are driven by the PLB arbiter and are valid any time the PLB_pendReq signal is asserted. These signals indicate the priority of the current request that the PLB arbiter has granted and is gating to the slaves. This priority will remain valid from the clock cycle that PLB_PAVali d or PLB_SAVali d are asserted, until the clock cycle in which the request has been acknowledged by the slave. These signals may also be used by slaves to resolve arbitration when requesting access to other buses.

2.2.4.4 PLB_masterID(0:3) (PLB Master Identification)

These signals are driven by the PLB arbiter and are valid in any clock cycle in which PLB_PAVali d or PLB_SAVali d are asserted. These signals indicate to the slaves the identification of the master of the current transfer. The slave must use these signals to determine which master busy signal and master signal should be driven on the PLB bus. The master ID may also be latched by the slave in an error syndrome register to indicate which master request caused the error.

Note: The width of the PLB_masterID signal (as shown in Table 2-5) is determined by the maximum number of masters supported by the particular PLB arbiter implementation.

Table 2-5. PLB Master Identification

Maximum # of Masters Supported by PLB Arbiter	PLB_masterID(0:n) Width
2	n = 0
3 to 4	n = 1
5 to 8	n = 2
9 to 16	n = 3

2.2.5 PLB Transfer Qualifier Signals

The PLB master address and transfer qualifier signals must be valid any time the Mn_request signal is asserted. These signals should continue to be driven by the master, unchanged, until the clock cycle following the assertion of PLB_MnAddrAck, PLB_MnRearbitrate, or Mn_abort. On the PLB slave interface, these signals are valid anytime PLB_PAVali d or PLB_SAVali d are asserted. The PLB slave should latch the transfer qualifier signals at the end of the address acknowledge cycle.

2.2.5.1 Mn_RNW, PLB_RNW (Read/NotWrite)

This signal is driven by the master and is used to indicate whether the request is for a read or a write transfer. If Mn_RNW = 0b1, the request is for the slave to supply data to be read

into the master. If Mn_RNW = 0b0, the request is for the master to supply data to be written to the slave.

2.2.5.2 Mn_BE(0:3), PLB_BE(0:3) (Byte Enables)

These signals are driven by the master. For a non-line and non-burst transfer they identify which bytes of the target word being addressed on Mn_ABus(0:31) are to be read from or written to. For a read transfer, the slaves should access the indicated bytes and place them on SI_rdDBus(0:31) in the proper memory alignment. These will then be steered to PLB_MnRdDBus(0:31). For a write transfer, the slaves should only write out the indicated bytes from Mn_wrDBus(0:31) to the external devices.

Note: The Mn_ABus(30:31) must always address the leftmost byte that is being transferred across the bus as shown in the Table 2-6 below.

Table 2-6. Byte Enable Signals

Mn_BE(0:3)	Transfer Request	Mn_ABus(30:31)
0000	Invalid	Invalid
0001	Byte 3	11
0010	Byte 2	10
0011	Halfword 2, 3	10
0100	Byte 1	01
0101	Invalid	Invalid
0110	Unaligned halfword 1, 2	01
0111	Bytes 1, 2, 3	01
1000	Byte 0	00
1001	Invalid	Invalid
1010	Invalid	Invalid
1011	Invalid	Invalid
1100	Halfword 0, 1	00
1101	Invalid	Invalid
1110	Bytes 0,1, 2	00
1111	Word	00

For line transfers, the Mn_BE(0:3) signals are ignored by the slave and the Mn_size(0:3) signals are used to determine the number of bytes that are to be read or written.

For burst transfers, the Mn_BE signals may optionally indicate the number of transfers that the master is requesting. The definition of the Mn_BE signals during burst transfers is shown in Table 2-7:

Table 2-7. Byte Enable Signals During Burst Transfers

Mn_BE(0:3)	Read Burst Length
0000	Burst length determined by PLB_rd/wrBurst signal.
0001	Burst of 2
0010	Burst of 3
0011	Burst of 4
0100	Burst of 5
0101	Burst of 6
0110	Burst of 7
0111	Burst of 8
1000	Burst of 9
1001	Burst of 10
1010	Burst of 11
1011	Burst of 12
1100	Burst of 13
1101	Burst of 14
1110	Burst of 15
1111	Burst of 16

Note: The burst length refers to the number of transfers of the data type selected by the Mn_size signals. The Mn_size = 1000 and Mn_BE = 1111 will transfer sixteen bytes, Mn_size = 1001 and Mn_BE = 1111 will transfer sixteen halfwords, and Mn_BE = 1111 and Mn_size = 1010 will transfer sixteen words.

Masters which do not implement the fixed length transfer should drive all 0's on the BE signals during burst transfers to be compatible with slaves which have implemented this feature. Slaves which do not implement the fixed length transfer will ignore the PLB_BE signals during a burst transfer and will continue bursting until the PLB_rd/wrBurst signal is negated by the master (see section 2.4.1.15, "Fixed Length Burst Read Transfer," on p. 2-56 for detailed description).

2.2.5.3 Mn_size(0:3), PLB_size(0:3) (Transfer Size)

The Mn_size(0:3) signals are driven by the master to indicate the size of the requested transfer. Table 2-8 defines all PLB transfer size signals.

Table 2-8. PLB Transfer Size Signals

Mn_size(0:3)	Definition
0000	Transfer one to four bytes of a word starting at the target address. See note 1.
0001	Transfer the 4-word line containing the target word. See note 2.
0010	Transfer the 8-word line containing the target word. See note 2.
0011	Transfer the 16-word line containing the target word. See note 2.
0100	Reserved
0101	Reserved
0110	Reserved
0111	Reserved
1000	Burst transfer - bytes - length determined by master. See Note 3 and 4.
1001	Burst transfer - halfwords - length determined by master. See Note 3 and 4.
1010	Burst transfer - words - length determined by master. See note 3 and 4.
1011	Burst transfer - double words - length determined by master. See note 3 and 5.
1100	Burst transfer - quad words - length determined by master. See note 3 and 5.
1101	Burst transfer - octal words - length determined by master. See note 3 and 5.
1110	Reserved
1111	Reserved

Note 1: A 0b0000 value indicates that the request is to read/write one to four bytes starting at the target address. The number of bytes to be read will be indicated on the Mn_BE(0:3) signals.

Note 2: For line read transfers, the target word may or may not be the first word transferred, depending on the design of the slave. For line read transfers, the SI_rdWdAddr(0:3) signals will indicate the word that is being transferred. For line write transfers, words must always be transferred sequentially, starting with the first word of the line (that is, Mn_ABus(28:31) = 4b0000, Mn_ABus(27:31) = 5b00000,

and Mn_ABus(26:31) = 6b000000 for a 4-word line write, an 8-word line write, and a 16-word line write, respectively).

Note 3: The Mn_BE(0:3) signals are ignored for a burst transfer.

Note 4: If Mn_size(0:3) is 0b1000, 0b1001, or 0b1010, the request is to burst read or write bytes, halfwords, or words, respectively. The slave should start transferring data at the address indicated by the PLB_ABus(0:31) and width as indicated by the size bits. The slave should then continue to read/write bytes, halfwords, or words, until the Mn_burst signal is negated indicating that the master is no longer in need of additional data.

Note 5: For PLB and devices supporting wider datapaths, double word, quad word, and octal word encodings are used to transfer 64-bits, 128-bits, and 256-bits, respectively. Here too, the slave should continue to read/write double words, quad words, or octal words, until the Mn_burst signal is negated indicating that the master is no longer in need of additional data.

2.2.5.4 Mn_type(0:2), PLB_type(0:2) (Transfer Type)

These signals are driven by the master and are used to indicate the type of transfer being requested. Table 2-9 defines all PLB transfer type signals.

Table 2-9. PLB Transfer Type Signals

Mn_type(0:2)	Definition
000	Memory transfer.
001	DMA flyby transfer. See note 1.
010	DMA buffered external peripheral transfer. See note 2.
011	DMA buffered OPB peripheral transfer. See note 2.
100	PLB slave buffered memory to memory transfer. See note 2.
101	PLB slave buffered peripheral to/from memory transfer. See note 2.
110	DMA buffered memory transfer. See note 3.
111	PLB slave buffered memory to memory transfer with sideband signals

Note 1: Must be used with Mn_size(0:3) values of 0b0000 and 0b1000 - 0b1101 only.

Note 2: Must be used with Mn_size(0:3) values of 0b0000 only.

Note 3: Slaves not supporting DMA peripheral transfers must also decode Mn_type(0:2) = 0b110 as a “memory transfer” in order to support DMA buffered memory-to-memory transfers.

Memory Transfers (Mn_type = 0b000):

This transfer type is used to read or write data from or to a device in the memory address space. Each PLB slave should decode the address on the PLB_ABus(0:31) to determine if the transfer is to/from the memory area that is controlled by the slave.

See section 2.5.5, "DMA Transfer Operations," on p. 2-77 for detailed description of DMA transfers using the PLB).

2.2.5.5 Mn_compress, PLB_compress (Compressed Data Transfer)

This signal is driven by the master to indicate whether or not the requested transfer is for a memory area containing compressed data. If the master is requesting a read data transfer and the PLB_compress signal is asserted, then the master is indicating that the data corresponding to the requested address is compressed and the slave must decompress the data prior to transferring it back to the master. If the master is requesting a write data transfer and the PLB_compress is asserted, then the master is indicating that the data corresponding to the requested address must be compressed and the slave must compress the data prior to writing it out to memory.

2.2.5.6 Mn_guarded, PLB_guarded (Guarded Memory Access)

This signal is driven by the master to indicate that the requested transfer may be for a non-well behaved memory. If a master is requesting a non-burst transfer (that is, Mn_size = 0nnn), and the PLB_guarded signal is negated, then the master is indicating that the 1K page of memory corresponding to the requested address is well behaved and that the slave can access all of the 1K page, but may stop at the 1K page boundary. If the master is requesting a non-burst transfer with the PLB_guarded signal asserted, then the master is indicating that the 1K page of memory corresponding to the requested address might be non-well behaved and the slave should restrict itself to access only exactly what was requested by the master.

If the master is requesting a burst transfer (that is, Mn_size = 1nnn), and the signal is negated, then the master is indicating that the 1K page corresponding to the requested address is well behaved and all subsequent 1K pages of memory are also well behaved, and the slave may access all memory on this page as well as subsequent pages. If the master is requesting a burst transfer with the PLB_guarded signal asserted, then the master is indicating that the 1K page of memory corresponding to the requested address is well-behaved but, the next 1K page of memory may not be well behaved and the slave should restrict itself to access only within the 1K page that corresponding to the initial requested address.

When stopping a burst transfer at a 1K page boundary, the slave may use the SI_BTerm signals to force the master to terminate the burst transfer. However, masters must not depend on a slave using the SI_BTerm signals to avoid crossing into a guarded page. Rather, masters must also include logic to detect the second-to-last read/write data acknowledge and negate the Mn_rdBurst/Mn_wrBurst signals in the following clock cycle in order to guarantee that the 1K page boundary will not be crossed. Following the detection of

the second-to-last data acknowledge, if a master decides that it is okay to cross into the next page, then it can indicate so by leaving the Mn_rdBurst/Mn_wrBurst signals asserted.

Similarly, slaves can also use the “wait before crossing a page” technique to help guarantee that a guarded page is not accessed if not explicitly required by a master. If the “wait” technique is used, slaves must not cross the 1K page boundary until they have returned the second-to-last read/write data acknowledge and given the masters the opportunity to negate their Mn_rdBurst/Mn_wrBurst signals in the following clock cycle. Following the detection of the second-to-last data acknowledge, if the Mn_rdBurst/Mn_wrBurst signals are still asserted, the slave can assume that the master has indeed requested data from the next page and so it can be accessed.

2.2.5.7 Mn_ordered, PLB_ordered (Ordered Transfer)

This signal is driven by the master for a write request to indicate whether or not the write transfer must be ordered. This signal is a transfer qualifier and must be valid anytime the Mn_request signal is asserted and the Mn_RNW signal is low (that is, logic ‘0’) indicating a write transfer. PLB slaves should ignore the PLB_ordered signal during read transfers.

When acknowledging a write request with the Mn_ordered signal asserted, the slave must not allow any subsequent requests (reads or writes) to get in between or ahead of the ordered write request. When acknowledging a write request with the Mn_ordered signal negated, the slave may decide to hold this request in a buffer and perform subsequent requests (reads or writes) prior to completing the un-ordered write request.

Although the Mn_ordered signal may be asserted with a burst write request, it does not prevent the slave from being able to terminate the burst transfer to allow other system resources to access data.

This signal is to be used by a master that needs to insure that a write transfer is completed prior to the written data being accessed by any other system resource.

2.2.5.8 Mn_lockErr, PLB_lockErr (Lock Error Status)

This signal is asserted by the master to indicate whether or not the slave must lock the Slave Error Address Register (SEAR) and the Slave Error Status Register (SESR) when an error is detected during the transfer. If the value of this signal is low, the slave should not lock the SEAR and SESR when the error occurs. Instead, the error address and syndrome should be latched into the SEAR and SESR and not locked. If a subsequent error is detected by this transfer or any other transfer the values in the SEAR and SESR will be overwritten. If the value of this signal is high (that is, logic ‘1’) the slave should lock the SEAR and SESR on the occurrence of any errors as a result of this transfer. Any errors that occur after the SEAR and SESR are locked will not override the values that were written with the first error. Once the SESR and SEAR registers are locked with an error, they will remain locked until software clears the SESR.

2.2.5.9 Mn_ABus(0:31), PLB_ABus(0:31) (Address Bus)

Each master is required to provide a valid 32-bit address when its request signal is asserted. The PLB will then arbitrate the requests and allow the highest priority master's address to be gated onto the PLB_ABus. For non-line transfers, this 32-bit bus indicates the lowest numbered byte address of the target data to be read/written over the PLB. The Mn_BE(0:3) signals will indicate which bytes of the word are to be read or written for this transfer. (See Section 2.2.5.2 for a more detailed description of the Mn_BE signals)

For line read transfers, the address bus may indicate the target byte address within the line of data that is being requested by the master. Slaves may read the data in any order and may use the target address to optimize performance by transferring the target word first. For line write transfers, the line word address must be zero since line writes transfers are required to be performed in sequential order across the PLB starting with the first word of the line. Table 2-10 indicates the bits of the Mn_ABus which must be zeroed for line write transfers.

Table 2-10. PLB Address Bus Signal Bits

Line Size	Line address	Word Address	Byte Address
4-word line	Mn_ABus(0:27)	Mn_ABus(28:29) = 00	Mn_ABus(30:31)
8-word line	Mn_ABus(0:26)	Mn_ABus(27:29) = 000	Mn_ABus(30:31)
16-word line	Mn_ABus(0:25)	Mn_ABus(26:29) = 0000	Mn_ABus(30:31)

The slave must latch the address at the end of the clock cycle in which it asserts SI_addrAck.

2.2.6 PLB Write Data Bus Signals

The PLB write data cycle is divided into two phases: transfer and data acknowledge. During the transfer phase, the master places the data to be written on the write data bus. The master then waits for a slave to indicate the completion of the write data transfer during the data acknowledge phase.

Note: A single-beat transfer will have one transfer phase and one data acknowledge phase associated with it. A line or burst transfer will have a multiple number of transfer and data acknowledge phases. It is also possible for both phases of the write data cycle to occur in a single PLB clock cycle.

A master begins a write transfer by asserting its Mn_request signal and placing a low value on the Mn_RNW signal and the first bytes of data to be written on the Mn_wrDBus(0:31) bus. Once it has granted the bus to the master, the PLB arbiter gates the data on Mn_wrDBus(0:31) onto the PLB_wrDBus(0:31) bus. The master then awaits for the slave to assert the SI_wrDAck signal to acknowledge the latching of the write data.

For single-beat transfers, the slave will assert the SI_wrDAck signal for one clock cycle only. For four-beat, eight-beat, or 16-beat transfers, the slave will assert the SI_wrDAck signal for 4, 8, and 16 clock cycles, respectively. For burst transfers, the slave will assert the SI_wrDAck signal for as many clock cycles as required by the master via the Mn_wrBurst signal. But in all cases, the slave will indicate the end of the current transfer by asserting the SI_wrComp signal for one clock cycle.

A slave may request the termination of a write burst transfer by asserting the SI_wrBTerm signal at anytime during the write data cycle.

In the case of address-pipelined write transfers, the PLB arbiter will assert the PLB_wrPrim signal to indicate the end of the data cycle for the current transfer and the beginning of the data cycle for the new transfer.

2.2.6.1 Mn_wrDBus(0:31), PLB_wrDBus(0:31) (Write Data Bus)

This 32-bit data bus is used to transfer data between a master and a slave during a PLB write transfer. For a primary write request, the master must place the first bytes of data to be written on the Mn_wrDBus(0:31) bus in the same clock cycle Mn_request is first asserted. For a secondary write request, the master must place the first bytes of data to be written on Mn_wrDBus(0:31) in the clock cycle immediately following the last data acknowledge for the primary write request.

Once a master has requested a write transfer it must begin to sample the PLB_MnWrDAck signal continuously. Note that the master must then retain the data on Mn_wrDBus(0:31) until the end of the clock cycle in which PLB_MnWrDAck is sampled asserted.

For non-line, non-burst transfers, (that is, Mn_size(0:3) = 0b0000), the master must retain the data on Mn_wrDBus(0:31) until the end of the clock cycle in which PLB_MnWrDAck is first sampled asserted, at which time the master will consider the transfer to be complete.

For line write transfers, the master must retain the first word of data (that is, Word-0) on Mn_wrDBus(0:31) until the end of the clock cycle in which PLB_MnWrDAck is first sampled asserted. The master will then continue to place a new word of data (that is, Word-1, Word-2, etc.) on Mn_wrDBus(0:31) every time PLB_MnWrDAck is sampled asserted, until this signal is sampled asserted for the last word of the line, at which time the master will consider the transfer to be complete.

For burst write transfers, the master must retain the first byte, halfword, or word of data (that is, Data-0) on Mn_wrDBus(0:31) until the end of the clock cycle in which PLB_MnWrDAck is first sample asserted. The master will then continue to place a new byte, halfword, or word of data (that is, Data-1, Data-2, etc.) on Mn_wrDBus(0:31) every time PLB_MnWrDAck is sampled asserted, until the burst transfer is completed.

Note: In the case of byte and halfword write burst transfers, the master is required to place the write data on the correct memory-aligned byte or halfword of the PLB. For example, if the master is performing a byte burst starting with address 0, the master must put the first byte on Mn_wrDBus(0:7), the second byte on Mn_wrDBus(8:15), the third byte on Mn_wrDBus(16:23), etc.

2.2.6.2 SI_wrDack, PLB_MnWrDack (Write Data Acknowledge)

This signal is driven by the slave for a write transfer to indicate that the data currently on the PLB_wrDBus(0:31) bus is no longer required by the slave (that is, the slave has either already latched the data or will latch the data at the end of the current clock cycle). For a primary write request, the slave may begin to assert the SI_wrDack signal in the clock cycle SI_addrAck is asserted. For a secondary write request, the slave may begin to assert the SI_wrDack signal in the clock cycle immediately following the assertion of PLB_wrPrim.

For single-beat write transfers, the signal is asserted for one clock cycle only. For line write transfers, the signal will be asserted for 4, 8, or 16 clock cycles. For burst write transfers, this signal will be asserted for as many clock cycles as the length of the burst requires, as indicated via the Mn_wrBurst signal.

Note: This signal must be driven by the slave to a low value any time that the slave is not selected or the slave is selected but not ready to transfer write data on the write data bus.

2.2.6.3 SI_wrComp, (Data Write Complete)

This signal is asserted by the slave to indicate the end of the current write transfer. It is asserted once per write transfer, either during the last beat of the data transfer or any number of clock cycles following the last beat of data transfer, but not before the last beat of the data transfer. The PLB arbiter uses this signal to allow a new write request to be granted in the following clock cycle.

Note: The slave may assert this signal in the same clock cycle that the request was granted if only one data transfer on the PLB is required and the address and data acknowledge signals are also asserted.

2.2.6.4 Mn_wrBurst, PLB_wrBurst (Write Burst)

This signal is driven by the master to control the length of a burst write transfer. A burst write of sequential bytes, halfword, or words may be requested by a master on the PLB by indicating a transfer size of 0b1000, 0b1001, or 0b1010, respectively, and a low value on the Mn_RNW signal.

When a write burst transfer is acknowledged by a slave, the slave will sample the PLB_wrBurst signal during every clock cycle in which the SI_wrDack signal is asserted to determine when to terminate the burst transfer. A high value indicates that the master requires at least one additional sequential byte, halfword, or word of data. A low value indicates that the current transfer is the last sequential transfer that the master requires to write. Note that this signal may be asserted only during a burst write transfer and must remain negated at all other times.

Once the write burst request has been acknowledged by the slave, the slave must sequentially increment its address for each of the following transfers and continue to do so until the PLB_wrBurst signal is sampled negated during a cycle in which the SI_wrDack signal is asserted.

The slave will complete the burst transfer by asserting the SI_wrComp signal. Note that since it is permissible for the slave to assert the SI_wrComp signal several clock cycles after the last data transfer clock cycle, the slave must ignore the PLB_wrBurst signal once this signal has been negated and until SI_wrComp has been asserted for the current burst transfer.

Note: In the case of a master requesting two back-to-back write burst requests, where the second request is acknowledged prior to all the data being transferred for the first request (that is, the second request is a secondary write request), the master must guarantee that Mn_wrBurst is re-asserted for the second request in the clock cycle immediately following the last data acknowledge for the first request. This is illustrated in section 2.4.2.7, "Pipelined Back-to-Back Write Burst Transfers," on p. 2-71.

This signal must be negated in response to the assertion of SYS_plbReset.

2.2.6.5 SI_wrBTerm, PLB_MnWrBTerm (Write Burst Terminate)

This signal is asserted by the slave to indicate that the current burst write transfer in progress must be terminated by the master. The slave may assert this signal with SI_addrAck, or during any clock cycle thereafter, up to and including the clock cycle in which SI_wrComp is asserted for the current transfer. In response to the assertion of this signal, the master is required to negate its Mn_wrBurst signal in the following clock cycle. The Mn_wrBurst signal is then sampled by the slave and when detected negated, the burst write transfer will then complete and the slave will assert the SI_wrComp signal.

Note: If the slave asserts the SI_wrBTerm signal in the same clock cycle the master negates its Mn_wrBurst signal, no further response is required by the master.

2.2.6.6 PLB_wrPrim (Write Secondary to Primary Indicator)

This signal is asserted by the PLB arbiter to indicate that a secondary write request may be considered a primary write request in the following clock cycle. Slaves supporting address pipelining must begin to sample this signal in the clock cycle that PLB_SAVValid is asserted for a secondary write request. In the clock cycle following the assertion of PLB_wrPrim, the PLB arbiter will gate the secondary write data onto the PLB_wrDBus, provided the secondary write request has already been acknowledged, or it is currently being acknowledged and Mn_abort is not asserted. Accordingly, the slave may begin to assert its SI_wrDack signal for a secondary write request in the clock cycle following the assertion of PLB_wrPrim.

Note: If a secondary write request which is either aborted or address acknowledged in the same clock cycle that SI_wrComp is asserted for a primary write request, the PLB_wrPrim signal will be asserted by the arbiter and should be ignored by the slaves.

2.2.7 PLB Read Data Bus Signals

The PLB read data cycle is divided into two phases: transfer and data acknowledge. During the transfer phase, the slave places the data to be read on the read data bus. The master then waits for the slave to indicate that the data on the read data bus is valid during the data acknowledge phase.

Note: A single-beat transfer will have one transfer phase and one data acknowledge phase associated with it, and a line or burst transfer will have a multiple number of transfer and data acknowledge phases. It is also possible for both phases of the read data cycle to occur in a single PLB clock cycle.

A master begins a read transfer by asserting its Mn_request signal and placing a high value on the Mn_RNW signal. Once it has granted the bus to the master, the PLB arbiter will gate the data on SI_rdbus(0:31) onto the PLB_MnRdbus(0:31) bus. The master will then await for the slave to assert the SI_rdAck signal to acknowledge that the data currently on the read data bus is valid.

For single-beat transfers, the slave will assert the SI_rdAck signal for one clock cycle only. For four-beat, eight-beat, or 16-beat line transfers, the slave will assert the SI_rdAck signal for 4, 8, and 16 clock cycles, respectively. For burst transfers, the slave will assert the SI_rdAck signal for as many clock cycles as required by the master via the Mn_rdBurst signal. But in all cases, the slave will indicate the end of the current transfer by asserting the SI_rdComp signal for one clock cycle.

A slave may request the termination of a read burst transfer by asserting the SI_rdBTerm signal at anytime during the read data cycle.

In the case of address-pipelined read transfers, the PLB arbiter will assert the PLB_rdPrim signal to indicate the end of the data cycle for the current transfer and the beginning of the data cycle for the new transfer.

2.2.7.1 SI_rdbus(0:31), PLB_MnRdbus(0:31) (Read Data Bus)

This 32-bit data bus is used to transfer data between a slave and a master during a PLB read transfer.

For a primary read request, the slave may begin to drive data on SI_rdbus(0:31) two clock cycles following the assertion of SI_addrAck. For a secondary read request, the Slave may begin to drive data on SI_rdbus(0:31) two cycles following the assertion of PLB_rdPrim. In both cases, the Slave may drive data on SI_rdbus(0:31) thru one clock cycle following the assertion of SI_rdComp.

Also for a primary read request, the master must begin to sample the PLB_MnRdAck signal two clock cycles following the assertion of PLB_MnAddrAck. For a secondary read request, the master must begin to sample the PLB_MnRdAck signal in the clock cycle immediately following the last data acknowledge for the primary. In both cases, the master must latch the data on PLB_MnRdbus(0:31) at the end of the clock cycle in which PLB_MnRdAck is sampled asserted.

For non-line read transfers, data must always be transferred at the requested width. Byte, halfword, three byte, and fullword transfers are possible. However, in the case of a read transfer involving a slave device whose datapath width is smaller than the width of the requested PLB transfer, the slave must first accumulate all of the requested data internally and then perform the read data transfer on the PLB at the requested width.

Note: Since the PLB read data bus is a shared bus, SI_rDBus(0:31) must be driven low (that is, with logic '0s') by the slave any time that the slave is not selected for a read transfer or SYS_plbReset is asserted.

2.2.7.2 SI_rdWdAddr(0:3), PLB_MnRdWdAddr(0:3) (Read Word Address)

These signals are driven by the slave and are used to indicate the word-address-within-the-line-of-data requested of a data word transferred as part of a read line transfer. Masters will sample these signals in the clock cycle SI_rdDack is asserted for a read line transfer.

Table 2-11. PLB Read Word Address Signals

Line Transfer Size	PLB_MnRdWdAddr(0:3)			
	(0)	(1)	(2)	(3)
4-word	Undefined	Undefined	Valid	
8-word	Undefined	Valid		
16-word	Valid			

Note: Since the PLB read data bus is a shared bus, SI_rdWdAddr(0:3) must be driven low (that is, with logic '0s') by the slave any time that the slave is not selected for a read transfer or SYS_plbReset is asserted. If selected for a read transfer the slave may begin to drive the SI_rdWdAddr(0:3) signals with non-zero logic values two clock cycles after it has asserted the SI_addrAck signal, thru one clock cycle following the assertion of SI_rdComp.

2.2.7.3 SI_rdDack, PLB_MnRdDack (Read Data Acknowledge)

This signal is driven by the slave to indicate that the data on the SI_rDBus(0:31) bus is valid and must be latched at the end of the current clock cycle. For a primary read request, the slave may begin to assert the SI_rdDack signal two clock cycles following the assertion of SI_addrAck. For a secondary read request, the slave may begin to assert the SI_rdDack signal two clock cycles following the assertion of PLB_rdPrim.

For single-beat read transfers, the signal is asserted for one clock cycle only. For line read transfers, the signal will be asserted for 4, 8, or 16 clock cycles. For burst read transfers, this

signal will be asserted for as many clock cycles as the length of the burst requires, as indicated via the Mn_rdBurst signal.

Note: This signal must be driven by the slave to a low value any time that the slave is not selected or the slave is selected but not ready to transfer read data on the read data bus.

2.2.7.4 SI_rdComp, (Data Read Complete)

This signal is driven by the slave and is used to indicate to the PLB arbiter that the read transfer is either already complete, or will be complete by the end of the next clock cycle. In order to optimize performance on the PLB, the slave should assert this signal one clock cycle before the data acknowledge phase for the last data transfer cycle and thus allow the next read transfer to be overlapped with data being transferred on the PLB. If this is not possible, then this signal should be asserted in the same clock cycle as the last data transfer (for minimum latency) or in any clock cycle following the last data transfer. The assertion of this signal will cause the PLB arbiter to gate the next read request to the slaves in that clock cycle.

During read burst transfers the SI_rdComp signal will normally be asserted either in the clock cycle in which the last SI_rDack is asserted, or in a subsequent clock cycle. The SI_rdComp signal may only be asserted in the cycle prior to the last SI_rDack during read burst transfers if the PLB_rdBurst signal is asserted by the master, and the SI_rBTerm signal is also asserted by the slave (see section 2.4.1.14, "Fixed Length Burst Transfer - Notes," on p. 2-54 for more details).

Note: The assertion of the SI_rdComp signal causes arbitration of the next request in the same clock cycle whereas assertion of SI_wrComp causes arbitration of the next request in the following clock cycle.

2.2.7.5 Mn_rdBurst, PLB_rdBurst (Read Burst)

This signal is driven by the master to control the length of a burst read transfer. A burst read of sequential bytes, halfword, or words may be requested by a master on the PLB by indicating a transfer size of 0b1000, 0b1001, or 0b1010, respectively, and a high value on the Mn_RNW signal. Once a read burst transfer has been acknowledged by a slave, the slave will start sampling the PLB_rdBurst signal in the clock cycle following the assertion of SI_addrAck, or PLB_rdPrim in the case of a read burst transfer acknowledged as a secondary request, to determine when to terminate the transfer. A high value indicates that the master requires additional sequential bytes, halfwords, or words of data. A low value indicates the master requires one, and only one, additional sequential byte, halfword, or word of data.

Once the first data transfer has been completed for a burst request, the slave must sequentially increment its address for each of the following data transfers and continue to do so until the PLB_rdBurst signal is sampled negated. In the clock cycle the PLB_rdBurst signal is sampled negated, the slave will also sample its SI_rDack signal. If asserted, the slave will terminate the transfer by asserting its SI_rdComp signal in the following clock cycle or in a later clock cycle. If negated, the slave will supply one, and only one, additional byte,

halfword, or word of data in the following clock cycle (or in a later clock cycle, depending on the number of wait states) and terminate the transfer by asserting its SI_rdComp signal with SI_rdBComp or in a later clock cycle.

Note 1: In the case of a master requesting two back-to-back read burst requests, where the second request is acknowledged prior to all the data being transferred for the first request (that is, the second request is a secondary read request), the master must guarantee that Mn_rdBurst is negated during the last data transfer for the first request (that is, the primary read request) before this signal is re-asserted for the second request. Furthermore, for the second request, this signal must be asserted in the clock cycle immediately following the last data acknowledge for the first request. This is illustrated in section 2.4.2.6, "Pipelined Back-to-Back Read Burst Transfers," on p. 2-70.

Note 2: It is not permissible for the master to assert the Mn_rdBurst signal until one cycle following the assertion of SI_addrAck and this signal may only be asserted during read burst transfers and must remain negated for all other transfer types.

This signal must be negated in response to the assertion of SYS_plbReset.

2.2.7.6 SI_rdBTerm, PLB_MnRdBTerm (Read Burst Terminate)

This signal is asserted by the slave to indicate to a master that the current burst read transfer in progress must be terminated. This signal may be asserted by the slave starting one clock cycle following the assertion of SI_addrAck, up to and including the clock cycle in which SI_rdBComp is asserted. In response to the assertion of this signal, the master is required to negate its Mn_rdBurst signal in the following clock cycle. The Mn_rdBurst signal is then sampled by the slave and when detected negated, the slave will supply one, and only one, additional piece of data. The transfer is then completed when the SI_rdBComp signal is asserted while the PLB burst signal is negated.

Note: If the slave asserts the SI_rdBTerm signal in the same clock cycle the master negates its Mn_rdBurst signal, no further response is required by the master.

2.2.7.7 PLB_rdBPrim (Read Secondary to Primary Indicator)

This signal is asserted by the PLB arbiter to indicate that a secondary read request which has already been acknowledged by a slave, may now be considered a primary read request. Slaves supporting address pipelining must begin to sample this signal in the clock cycle following the assertion of SI_addrAck in response to the assertion of PLB_SAVValid. When transferring data for a secondary read request, the slave may begin to drive the SI_rdBComp(0:31) bus two clock cycles after it has sampled the PLB_rdBPrim signal asserted.

Note: If there is not a secondary read request on the PLB or a secondary read request has not been acknowledged by a slave, then the PLB_rdBPrim signal will not be asserted.

2.2.8 Additional Slave Output Signals

In addition to signals described in the previous sections, the following slave output signals are defined here:

2.2.8.1 SI_MBusy(0:n), PLB_MBusy(0:n) (Master Busy)

These signals are driven by the slave and are used to indicate that the slave is either busy performing a read or a write transfer, or has a read or write transfer pending. Each slave is required to drive a separate busy signal for each master attached on the PLB bus (ie. SI_MBusy(0) corresponds to Master ID0, SI_MBusy1 corresponds to master ID1 etc.). The slave should latch the master ID and use this ID to drive the corresponding master busy signal until the data transfer has been completed.

During read transfers, the SI_MBusy signal will remain asserted until the final SI_rdDAck is asserted by the slave. During write transfers, the signal may remain asserted following the assertion of the last SI_wrDAck and should remain asserted until the write transfer is completed from the perspective of the slave. Normally, this is the completion of the write data transfer on the slave bus.

If a slave is using a store queue, the slave must drive the master's busy signal starting in the clock cycle following the address acknowledge cycle, during the time that the request is in the queue and during the time that the request is being transferred. If the queue can store multiple requests, then the slave will be required to latch the master ID of each request being held and drive multiple master busy signals at the same time. Each slave's busy signals are or'ed together and sent to the appropriate master. The PLB will 'or' together all of the slave busy outputs for each master and send one busy signal to each master on the PLB.

The master busy signals may be used by a master to determine if all of its transfers have been completed by the slaves.

Note: The width of the SI_MBusy(0:n) and PLB_MBusy(0:n) signals is determined by the number of masters supported by the particular PLB based system.

2.2.8.2 SI_MErr(0:n), PLB_MErr(0:n) (Master Error)

These signals are driven by the slave and are used to indicate that the slave has encountered an error during a transfer that was initiated by this master. Each slave is required to drive a separate error signal for each master attached on the PLB bus (that is, SI_MErr(0) corresponds to master ID 0, SI_MErr(1) corresponds to master ID 1 etc.). The slave will drive this signal for one clock cycle for each error that is encountered while trying to complete the transfer.

On read transfers, the error signal is guaranteed to be asserted during the data acknowledge phase of the data transfer cycle. During write transfers, the error signal may be asserted several clock cycles after the PLB write transfer has completed and therefore may not be asserted during the write data transfer cycle. The master will need to examine the Slave Error Address Register (SEAR) and Slave Error Status Register (SESR) in each slave

to determine which transfer the error occurred on. Again, as with the master busy signal, the slave should latch the master ID input to determine which master's error line should be asserted. The PLB will 'or' together all of the slave error inputs for each master and send one error signal to each master on the PLB.

Note: The width of the SI_Merr(0:n) and PLB_MErr(0:n) signals is determined by the number of masters supported by the particular PLB based system.

2.3 PLB Interfaces

The PLB bus I/O signals are grouped under the following interface categories depending on their function. For detailed functional description of various signals see section 2.2, "PLB Signals," on p. 2-6.

- PLB Master Interface
- PLB Slave Interface
- PLB Arbiter Interface

2.3.1 PLB Master Interface

Figure 2-4 demonstrates all PLB master interface input/output signals. See section 2.2, "PLB Signals," on p. 2-6 for detailed functional description of the signals. Note that the use of the PLB_pendReq and PLB_pendPri signals by a master is optional and not required by the PLB architecture.

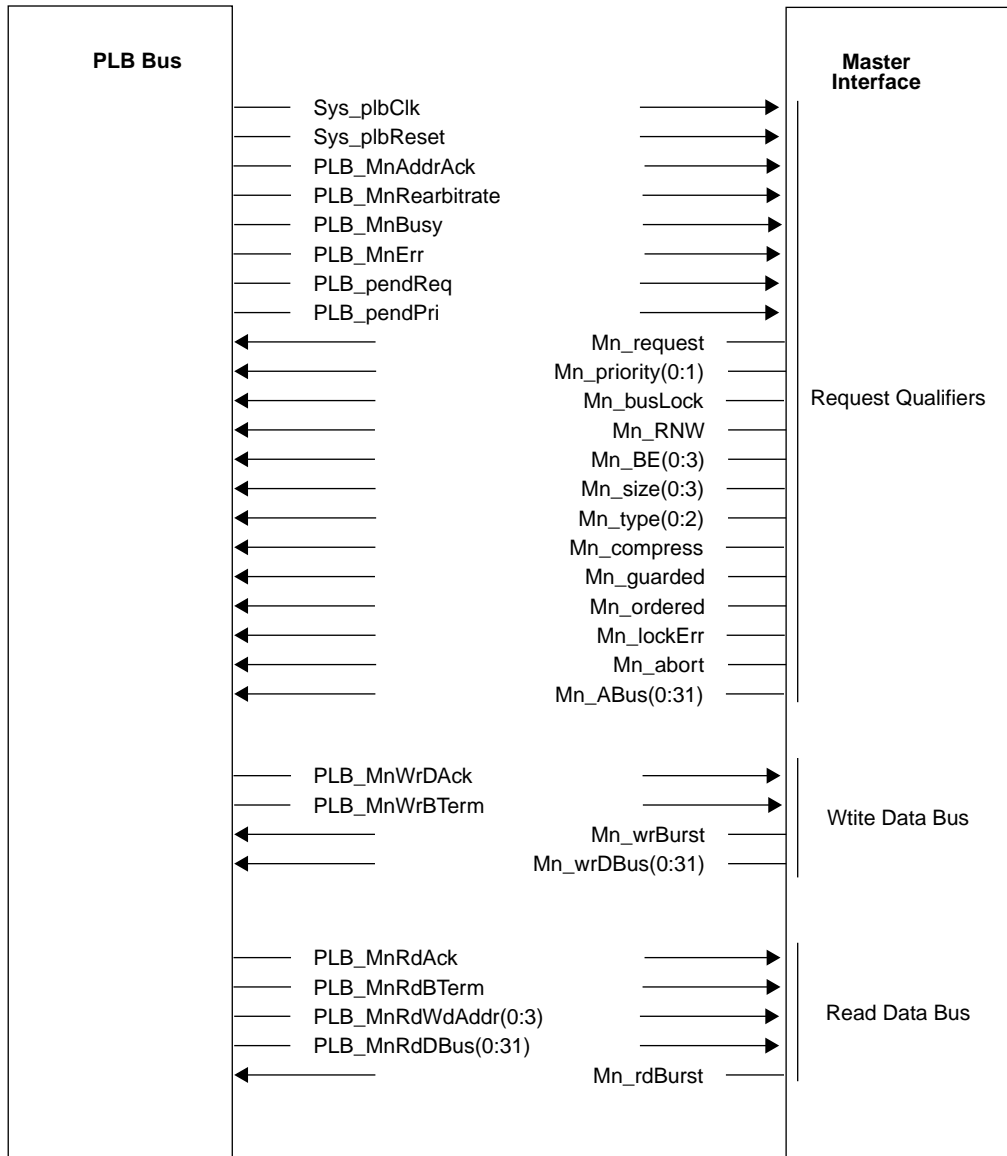


Figure 2-4. Master Interface

2.3.2 PLB Slave Interface

Figure 2-5 demonstrates all PLB slave interface input/output signals. See section 2.2, "PLB Signals," on p. 2-6 for detailed functional description of the signals.

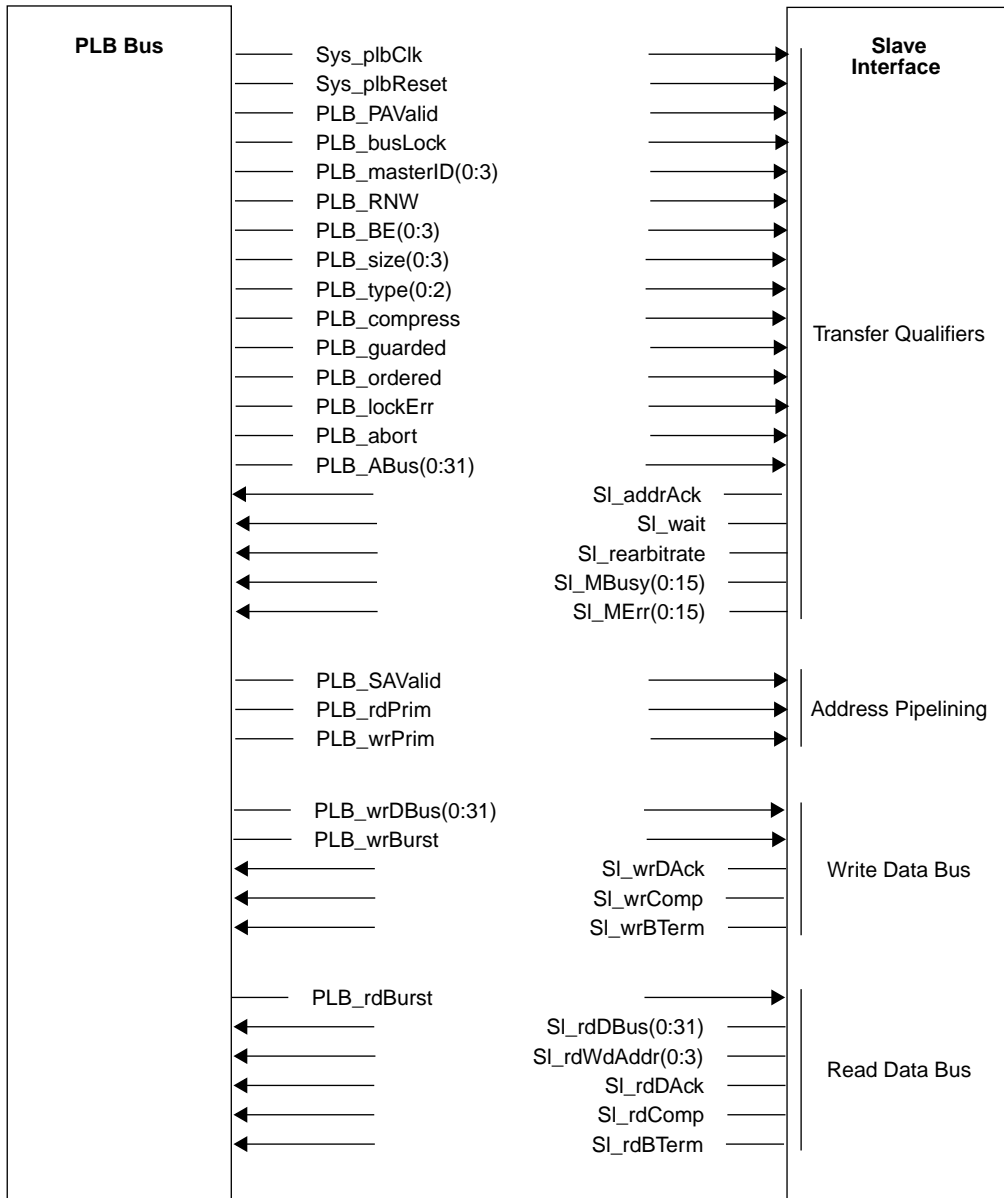


Figure 2-5. PLB Slave Interface

2.3.3 PLB Arbiter Interface

Figure 2-6 demonstrates all PLB arbiter interface input/output signals. See section 2.2, "PLB Signals," on p. 2-6 for detailed functional description of the signals.

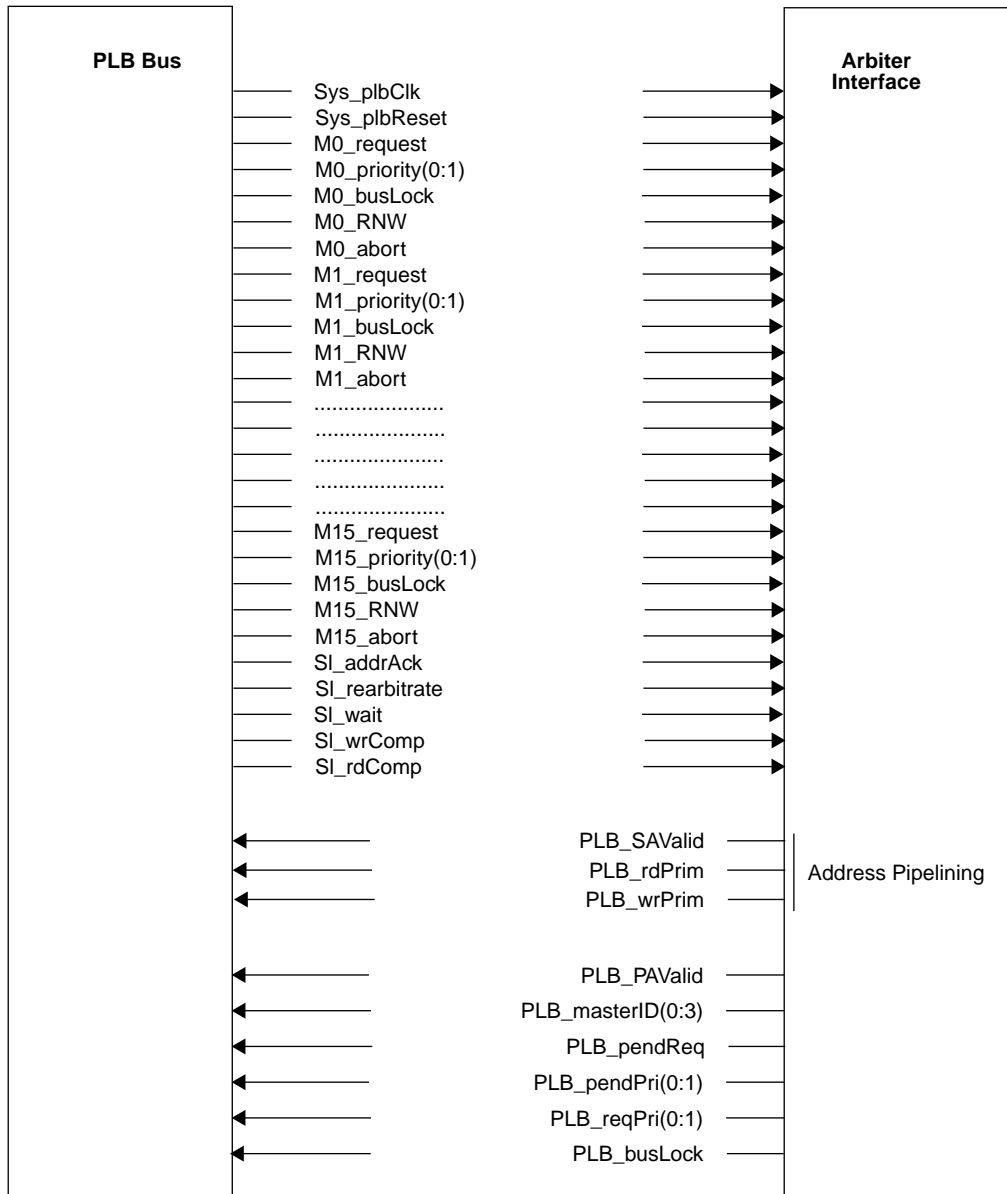


Figure 2-6. PLB Arbiter Interface

2.4 PLB Operations

This section on PLB operations discusses in detail the following topics with appropriate timing diagrams:

- PLB Non-Address Pipelining
- PLB Address Pipelining
- PLB Bandwidth and Latency

All signals on the PLB are positive active and are either direct outputs of edge triggered latches which are clocked by Sys_plbClk, or are derived from the output of a register using several levels of combinatorial logic. In addition, all input signals should be captured in the masters or slaves on the rising edge of Sys_plbClk.

2.4.1 PLB Non-Address Pipelining

The timing diagrams included in this section are examples of non-address pipelined read and write transfers on the PLB. However, it is important to note that signal assertion and negation times as shown in these diagrams, are only meant to illustrate their dependency on the rising edge of Sys_plbClk and in no way are they intended to show real signal timing.

Furthermore since set-up and hold times for the PLB inputs will be dependent on the technology used, and the physical implementation of the bus, these parameters will be specified as a percentage of the bus clock cycle relative to the rise of Sys_plbClk. A set of signal timing guidelines to be used in the design of PLB masters and slaves has been developed and described in section 5.1, "PLB Timing Guidelines," on p. 5-1.

2.4.1.1 Read Transfers

Figure 2-7 shows the operation of a single read data transfer on the PLB. The slave asserts `SI_wait` to indicate to the PLB arbiter that the address is valid but is unable to latch the address or transfer qualifiers at this time. The PLB arbiter will continue to drive the `PLB_PValid` signal as well as the address and transfer qualifier signals until the slave device asserts the `SI_addrAck` signal. The slave then asserts the `SI_rdComp` signal in the clock cycle preceding the data acknowledge phase to indicate that the transfer will complete in the following clock cycle and that the arbiter may arbitrate the next read request.

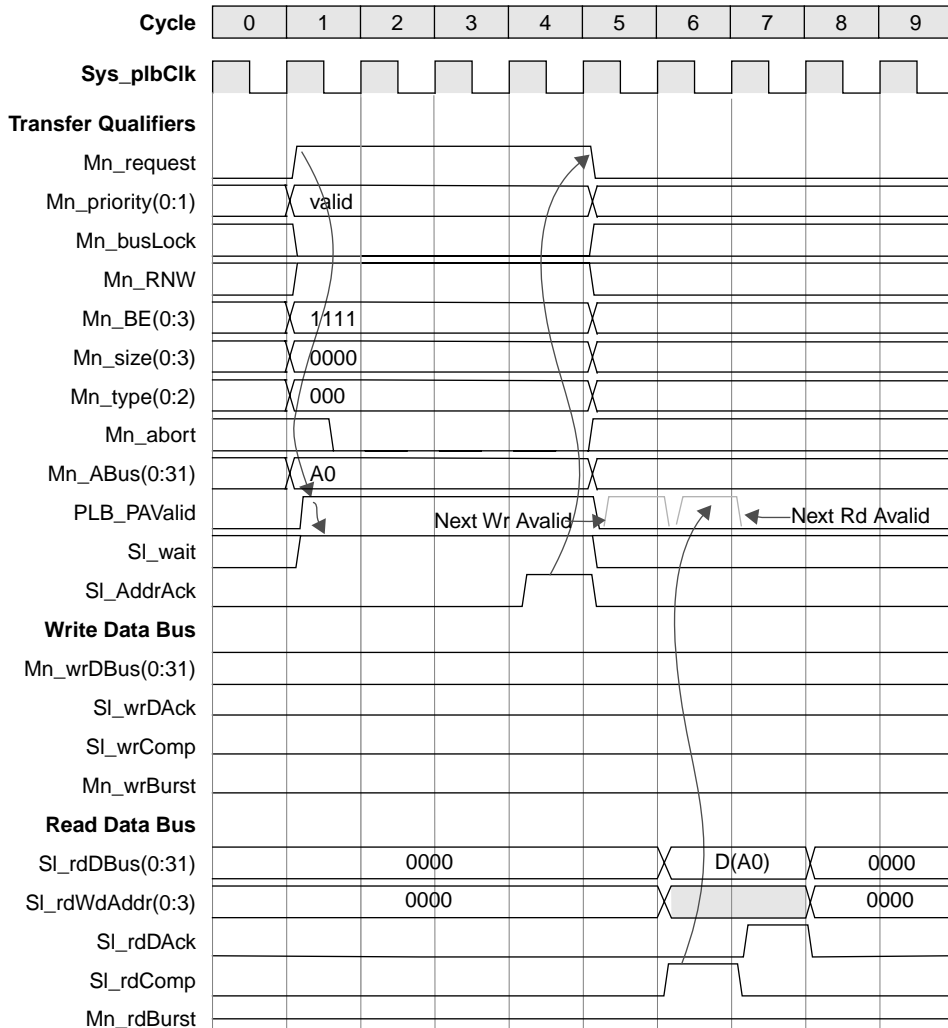


Figure 2-7. Read Transfers

2.4.1.2 Write Transfers

Figure 2-8 shows the operation of a single write data transfer on the PLB. The slave asserts the SI_wait signal to indicate to the PLB arbiter that the address is valid but that the slave is unable to latch the address or transfer qualifiers at this time. The PLB arbiter will continue to drive the PLB_PAValid signal as well as the address and transfer qualifier signals until the slave device asserts the SI_addrAck signal. The slave then asserts the SI_wrComp and SI_wrDack to indicate that data is valid on the bus and that the transfer is complete. Note that the write data bus must be valid at the time Mn_request is first asserted and held until the end of the clock cycle in which SI_wrDack signal is asserted.

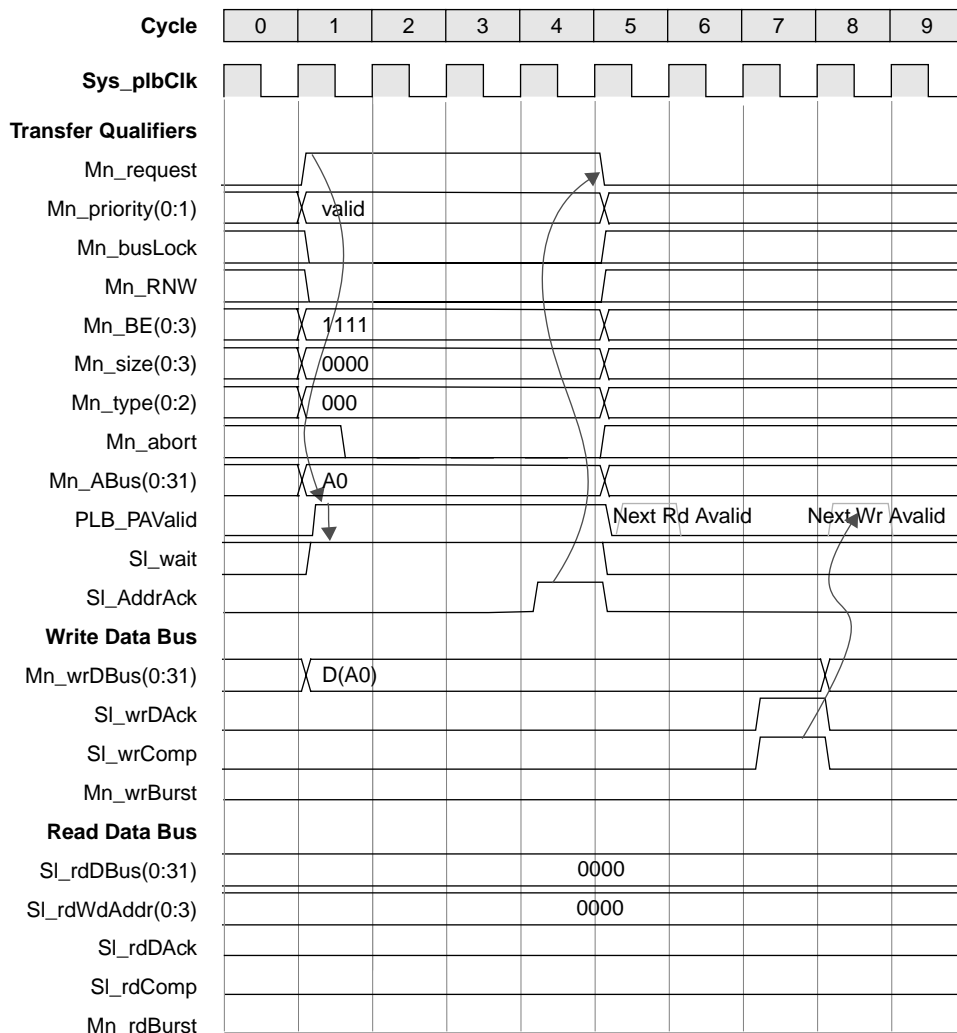


Figure 2-8. Write Transfers

2.4.1.3 Transfer Abort

Figure 2-9 shows a transfer aborted by the master in the same clock cycle the request was being acknowledged by the slave. When the master asserts Mn_abort signal in clock cycle 4, the PLB arbiter and PLB slaves ignore the address acknowledge and abort the requested transfer. All active requests are then sampled in the next clock cycle when the PLB arbiter re-arbitrates. The Mn_abort signal will have a minimal amount of set-up time to allow this signal to be asserted late in a clock cycle. Note that the data handshaking will not be completed via the assertion of the data acknowledge signals. The master may either negate its request signal or make a new request in the clock cycle following the aborted request.

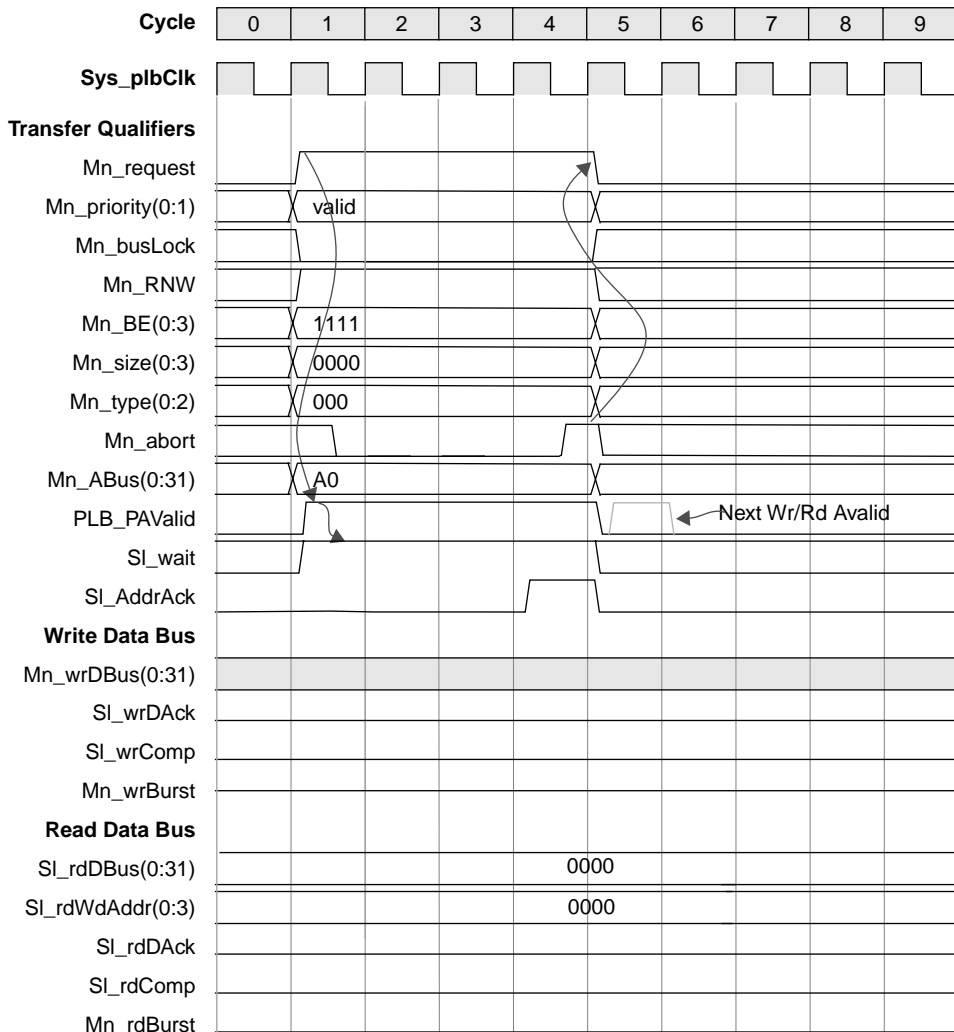


Figure 2-9. Transfer Abort

2.4.1.4 Back-to-Back Read Transfers

Figure 2-10 shows the operation of several back-to-back single read transfers on the PLB. The slave asserts the SI_rdComp signal in the clock cycle preceding the SI_rdDack. This allows the next master's read request to be sent to slaves in the clock cycle preceding the data acknowledge phase on the PLB. The slave may not assert its SI_rdDack for the data read until two clock cycles following the assertion of the corresponding SI_addrAck. This allows time for the previous read data transfer to complete before the data is transferred for the subsequent read. Using this protocol, a master may read data every clock cycle from a slave which is capable of providing data in a single clock cycle.

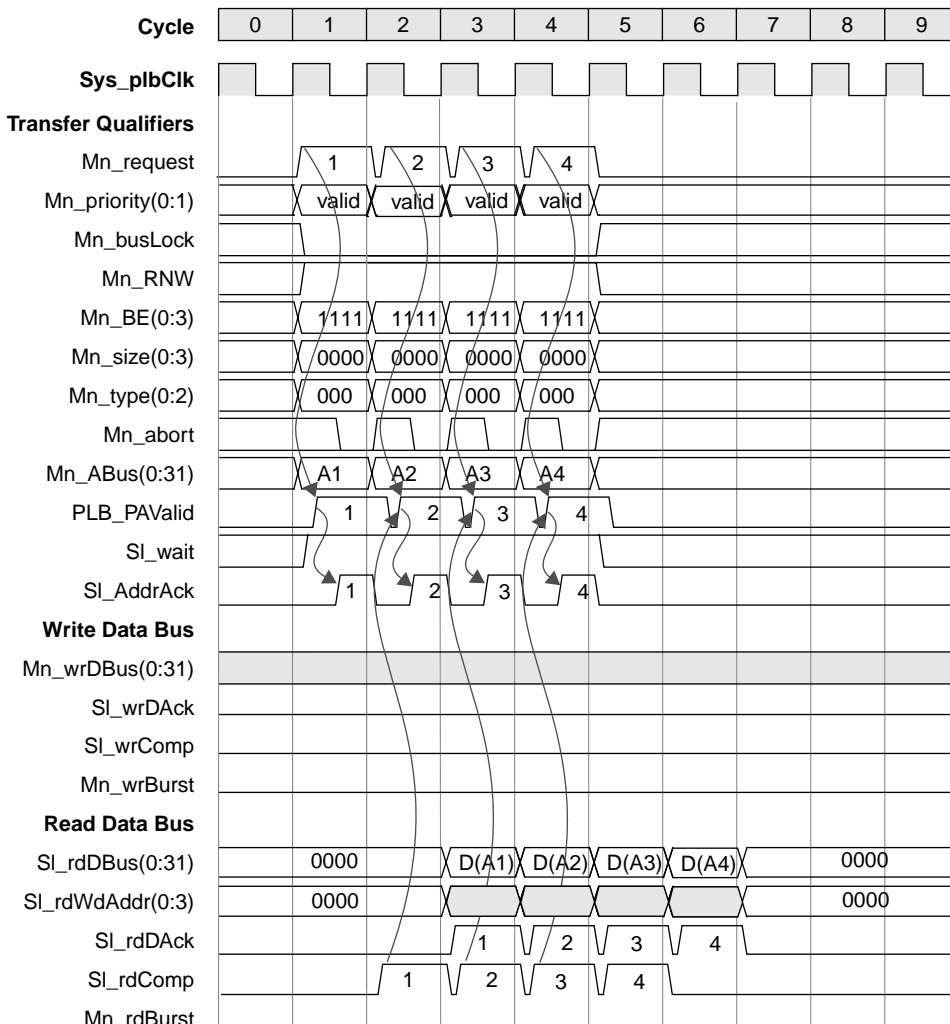


Figure 2-10. Back-to-Back Read Transfers

2.4.1.5 Back-to-Back Write Transfers

Figure 2-11 shows the operation of several back-to-back single write transfers on the PLB. The slave must assert the SI_addrAck, SI_wrDAck, and SI_wrComp signals in the same clock cycle that the PLB_PAVValid signal is asserted to complete the transfer within a single clock cycle on the PLB. The next valid write address cycle will occur in the clock cycle following the assertion of the SI_wrComp signal.

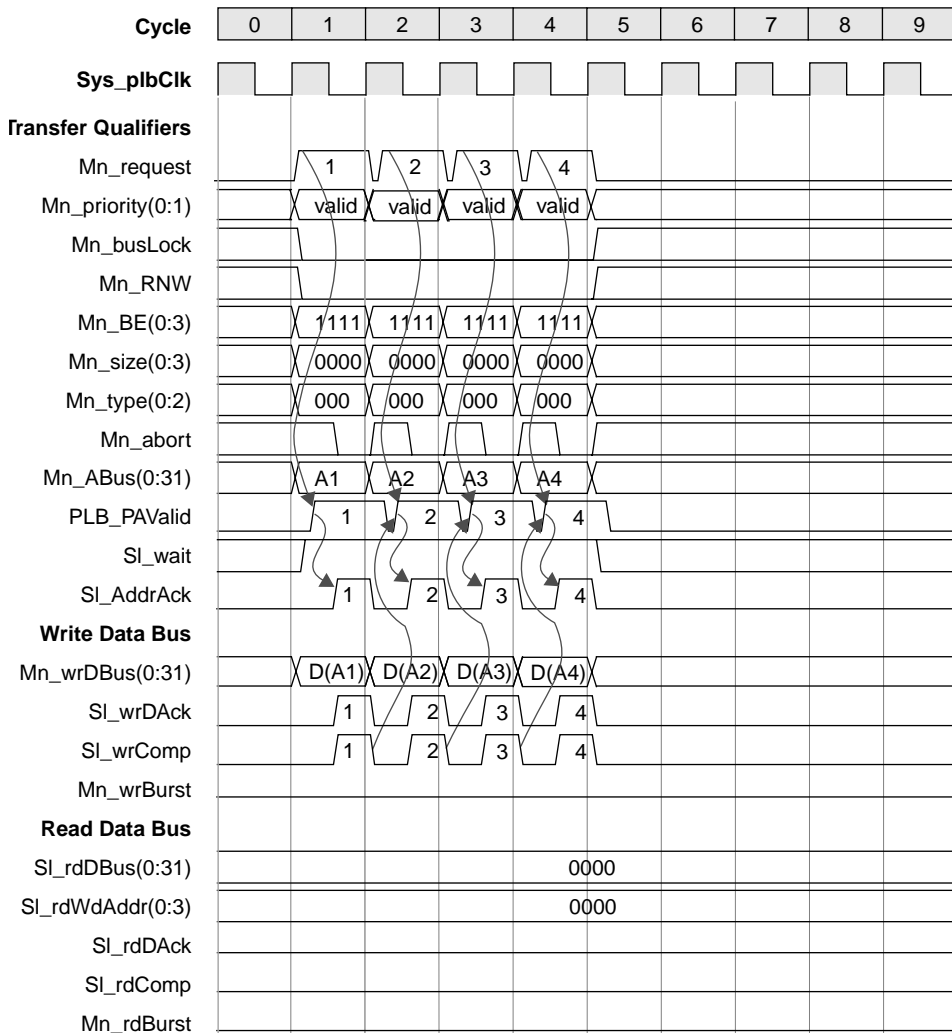


Figure 2-11. Back-to-Back Write Transfers

2.4.1.6 Back-to-Back Read - Write - Read - Write Transfers

Figure 2-12 shows the operation of several back-to-back single read and write transfers on the PLB. Note that although the PLB arbiter granted the requests in the order that they were presented, the data transfer for the write transfers occurs in the clock cycle previous to the data transfers for the read transfers. Using this protocol, a slave may be continuously read or written at a rate of one transfer per clock cycle.

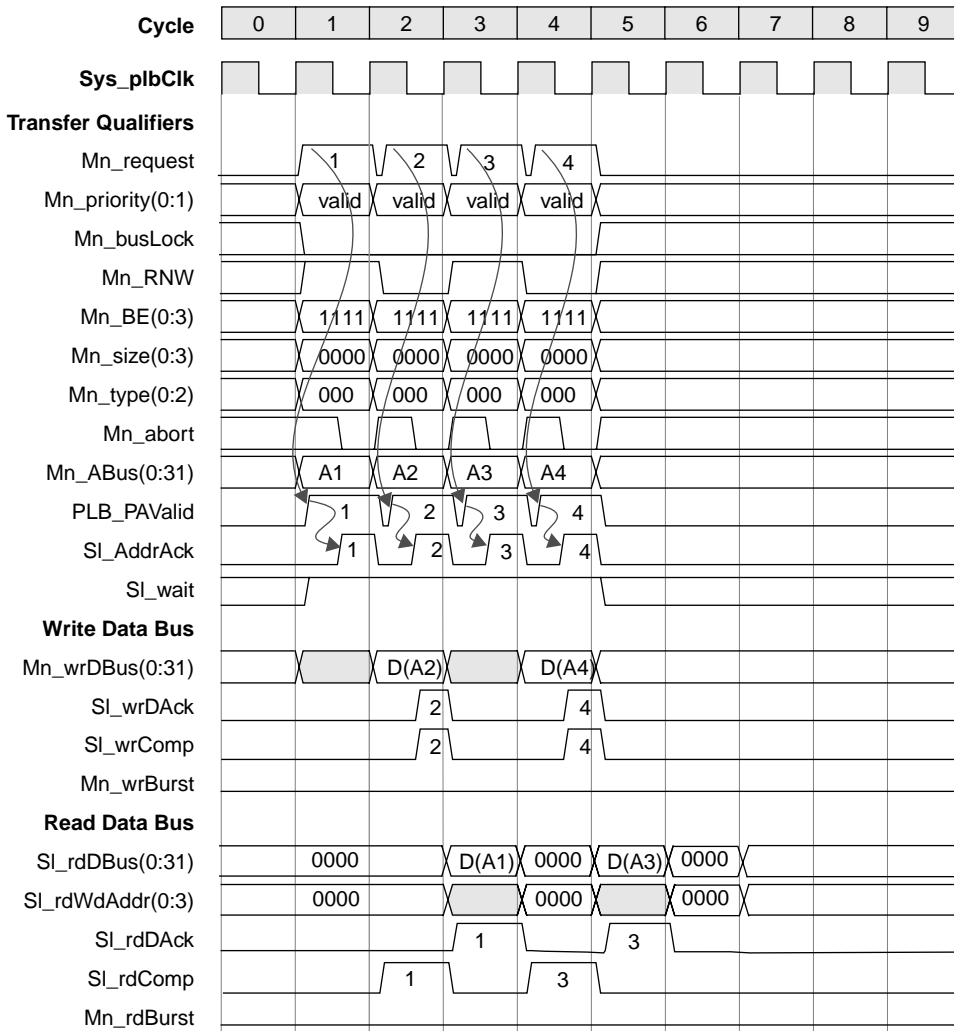


Figure 2-12. Back-to-Back Read - Write - Read - Write

2.4.1.7 Four-word Line Read Transfers

Figure 2-13 shows the operation of a single four word line read from a slave device which is capable of providing data in a single clock cycle. For line transfers, the words within the line may be transferred in any order and the SI_rdWdAddr(0:3) outputs of the slave will indicate to the master which word is being transferred in each data transfer cycle. The SI_rdComp signal is asserted in the clock cycle preceding the last data transfer indicating to the PLB arbiter that the line transfer will be complete in the following clock cycle.

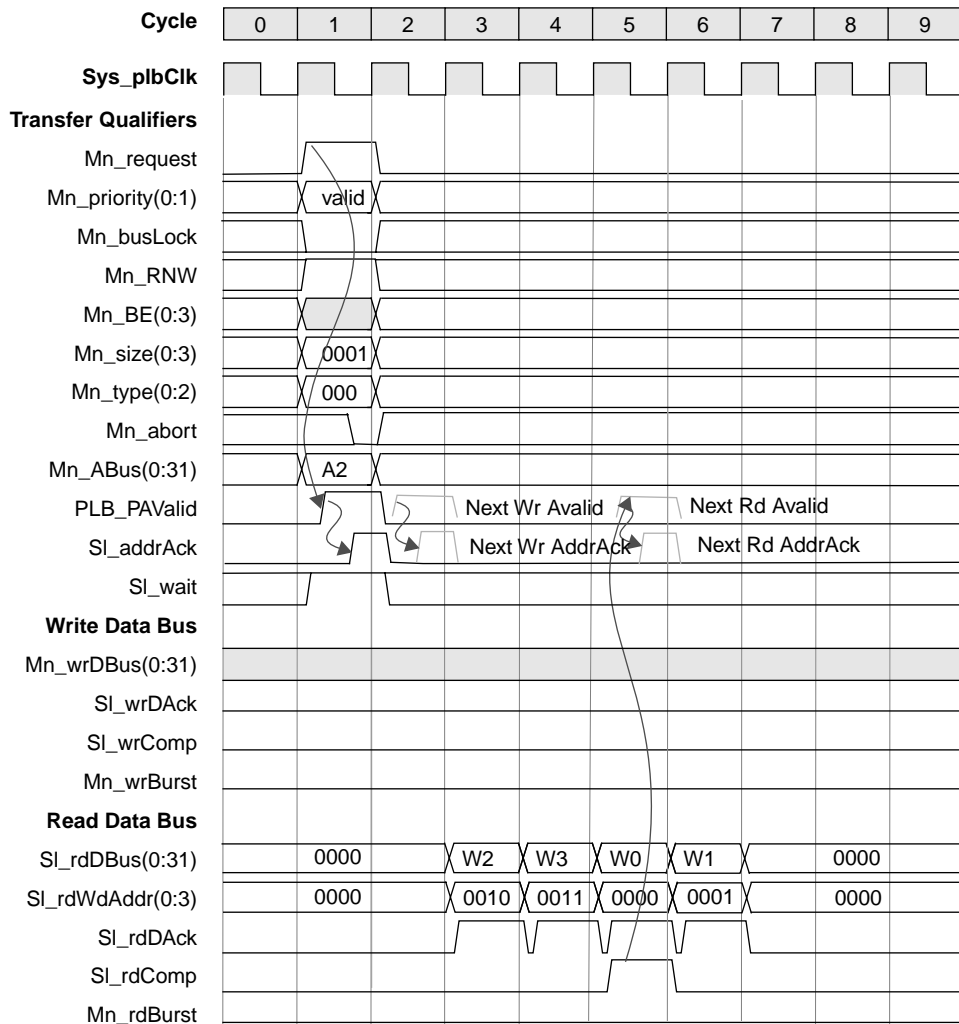


Figure 2-13. Four Word Line Read

2.4.1.8 Four-word Line Write Transfers

Figure 2-14 shows the operation of a single four-word line write to a slave device which is capable of latching data every clock cycle from the PLB. During the address cycle, the slave device asserts the *SI_addrAck* and the *SI_wrDack* signals but not the *SI_wrComp* signal. The *SI_wrComp* signal is asserted during the clock cycle in which the last *SI_wrDack* signal is asserted and is used by the PLB arbiter to allow the next write request to be gated onto the PLB.

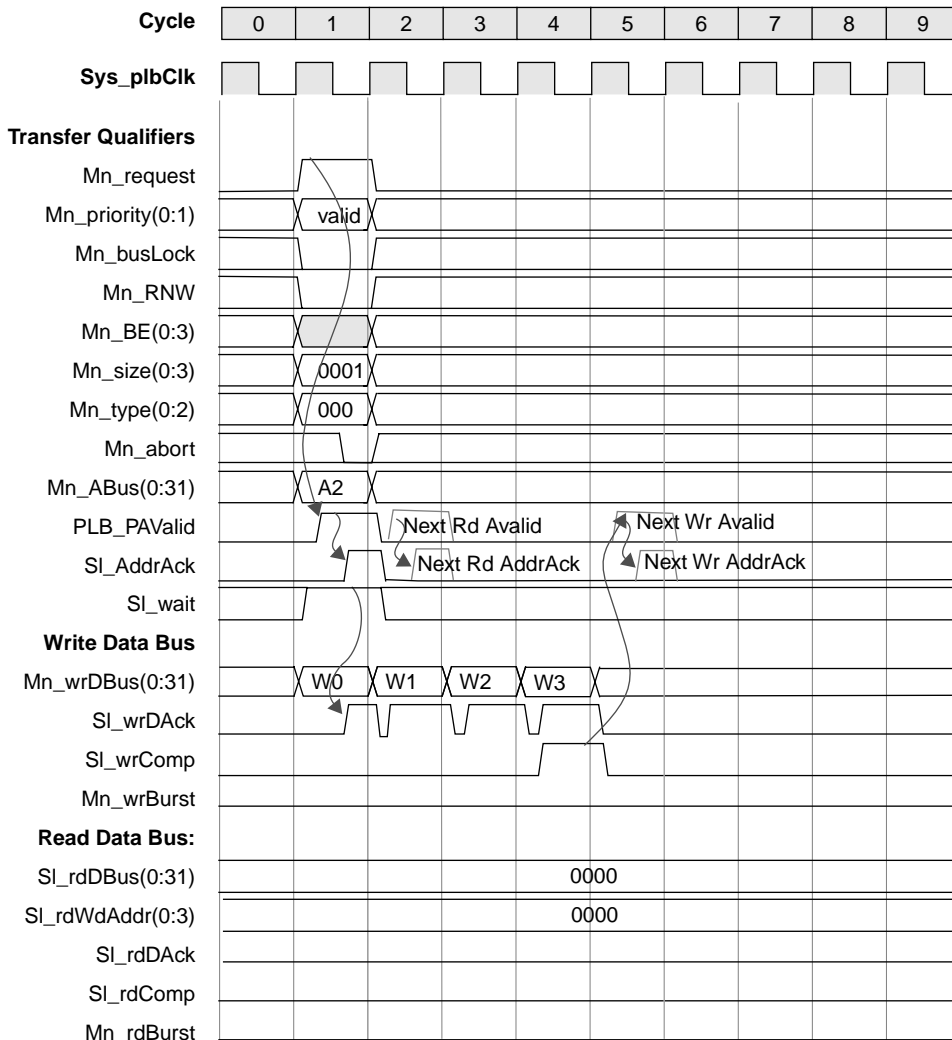


Figure 2-14. Four Word Line Write

2.4.1.9 Four-word Line Read Followed By Four-word Line Write Transfers

Figure 2-15 shows the operation of a four-word line read followed immediately by a four-word line write on the PLB. The read request is acknowledged in cycle 1 and the data transfers on the read data bus occur in cycles 3 through 6. During cycle 2, the PLB arbiter gates the write request to the slaves and this transfer is acknowledged by a Slave in the same clock cycle. The data transfer for the write line request occurs on the write data bus in cycles 2 through 5. The separate PLB read and write data buses allow the line write data transfers to be completely overlapped with the line read data transfers.

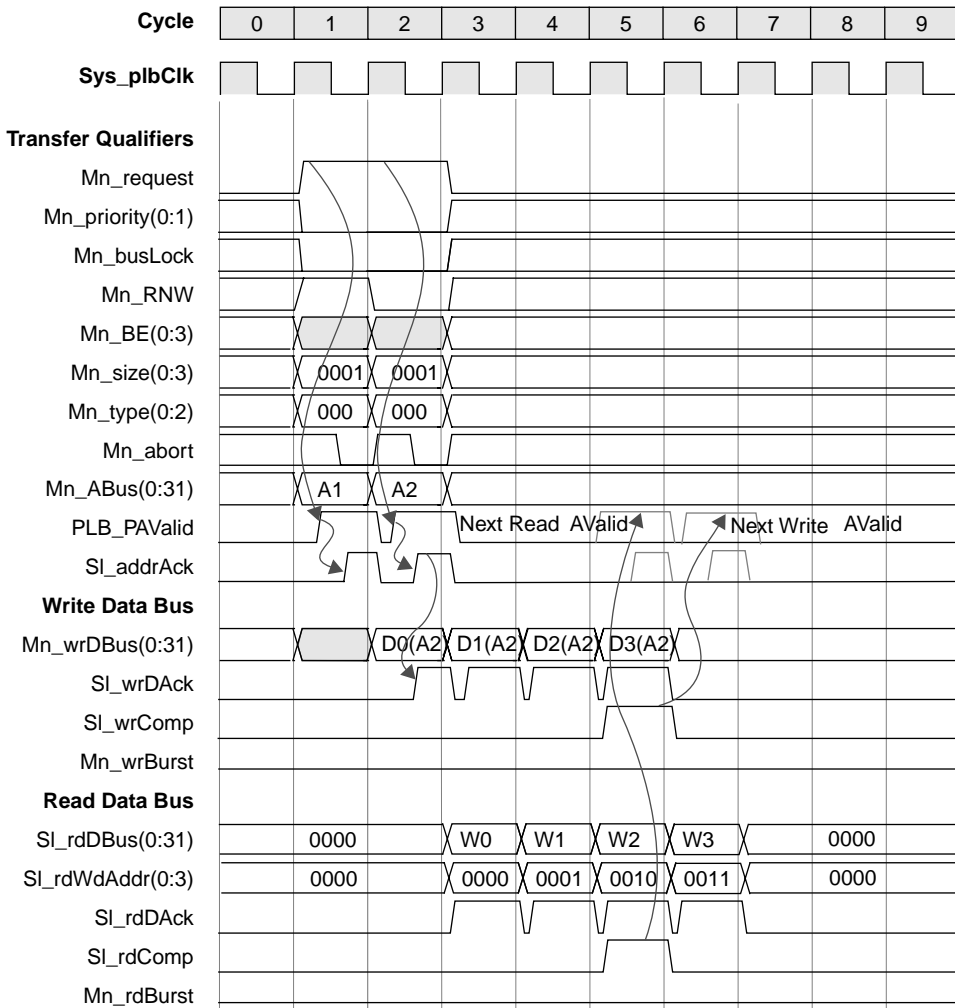


Figure 2-15. Four Word Line Read followed by Four Word Line Write

2.4.1.10 Sequential Burst Read Transfer Terminated by Master

Figure 2-16 shows the operation of a burst read from a slave device on the PLB. A master may request a burst transfer across the bus if it needs to read two or more sequential memory locations. The address bus and transfer qualifiers are latched by the slave when the SI_addrAck signal is asserted. The slave will internally increment the address sequentially for each data transfer and will continue to fetch data until it detects a low value on the Mn_rdBurst signal. The burst transfer is then completed by the slave device asserting the SI_rdComp in the data acknowledge phase of the last data transfer cycle following the negation of the Mn_rdBurst signal.

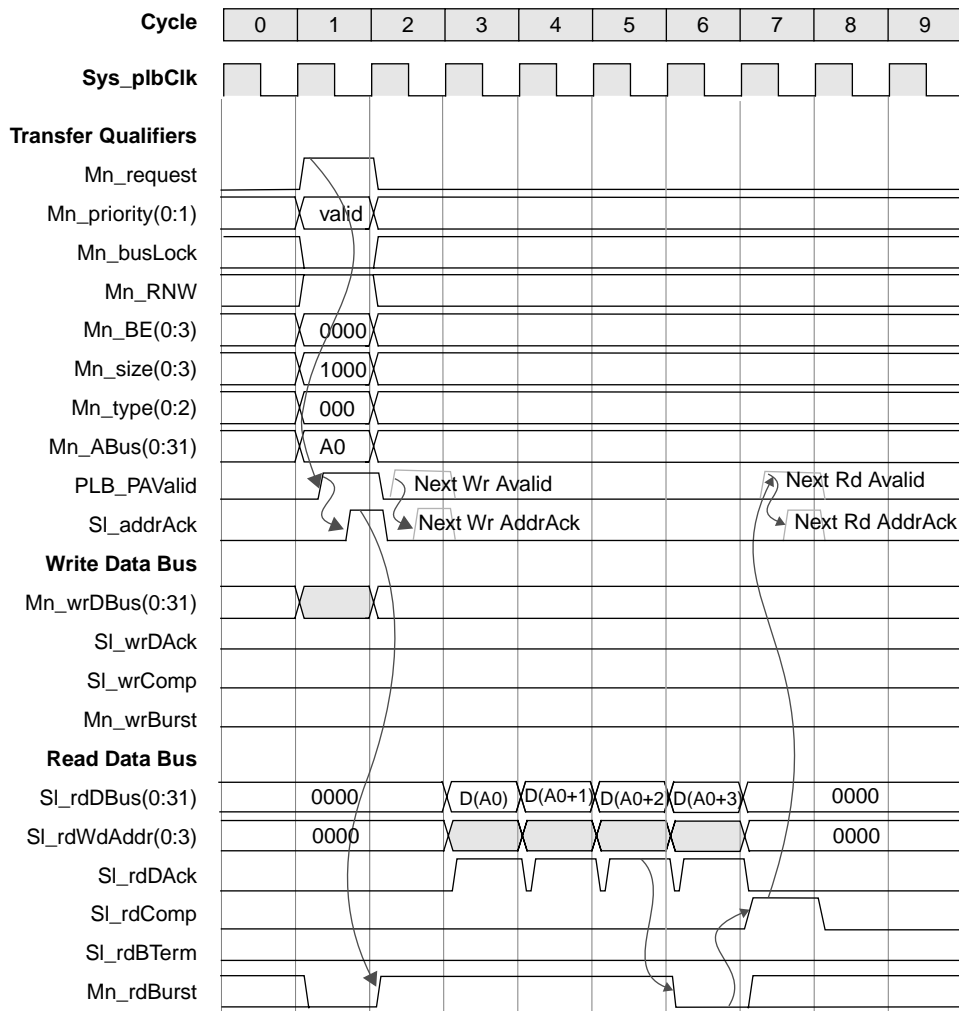


Figure 2-16. Burst Read Transfer Terminated By Master)

Figure 2-17 shows the operation of another burst read from a slave device on the PLB. This burst read transfer differs from the one shown in Figure 2-16 in that the transfer is terminated by the master negating the Mn_rdBurst signal in response to the assertion of the SI_rdBTerm by the slave device. The burst transfer is then completed by the slave device asserting the SI_rdComp in the data acknowledge phase of the last data transfer cycle.



2.4.1.12 Sequential Burst Write Transfer Terminated by Master

Figure 2-18 shows the operation of a burst write to a slave device on the PLB. A master may request a burst write transfer across the bus if it needs to write two or more sequential memory locations. The address bus and transfer qualifiers are latched by the slave when the SI_addrAck signal is asserted. The slave will internally increment the address sequentially for each data transfer. Once the slave detects a low value on the Mn_wrBurst signal, the slave will assert the SI_wrComp signal during the data acknowledge phase for the next (and last) data transfer cycle to indicate to the PLB arbiter that the burst transfer is complete.

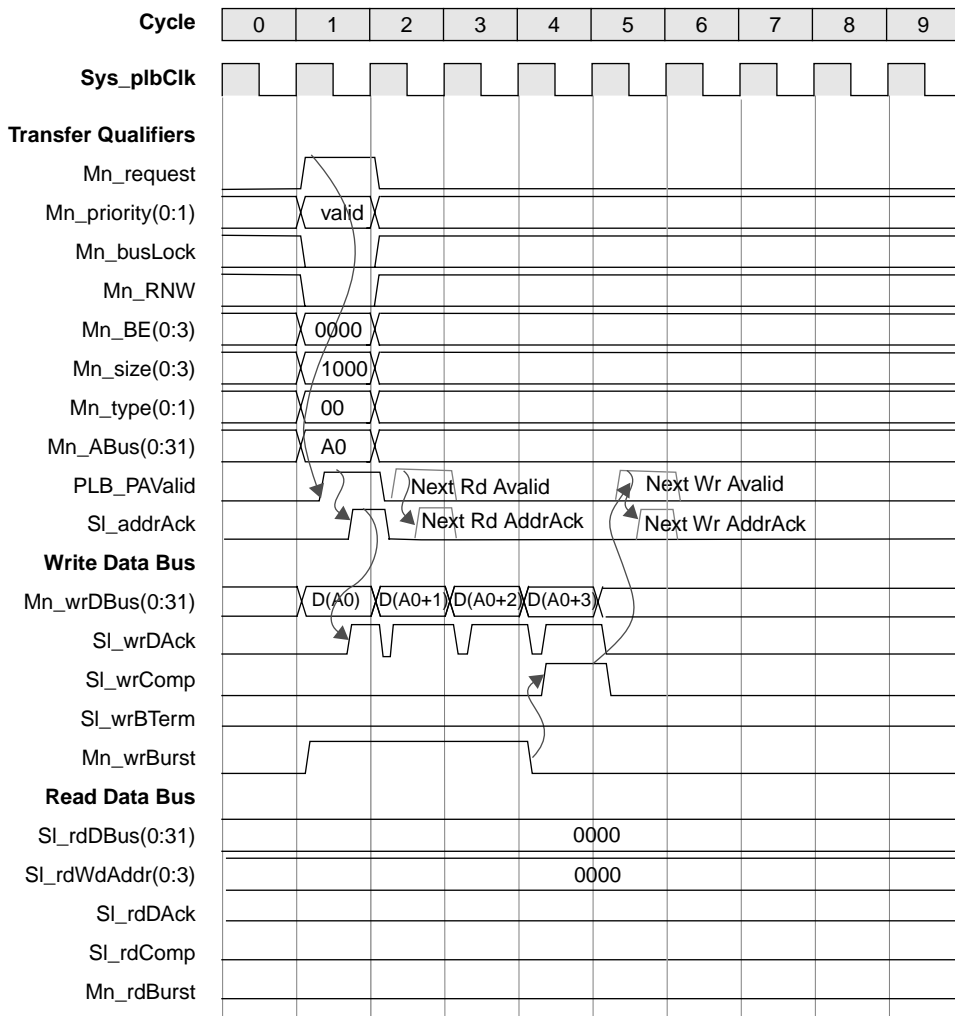


Figure 2-18. Burst Write Transfer Terminated by Master

2.4.1.13 Sequential Burst Write Transfer Terminated By Slave

Figure 2-19 shows the operation of another burst write to a slave device on the PLB. This burst write transfer differs from the burst write transfer illustrated in Figure 2-18 in that the transfer is terminated by the master negating the Mn_wrBurst signal in response to the assertion of the SI_wrBTerm signal by the slave device. The burst transfer is then completed by the slave device asserting the SI_wrComp during the data acknowledge phase for the next (and last) data transfer cycle.

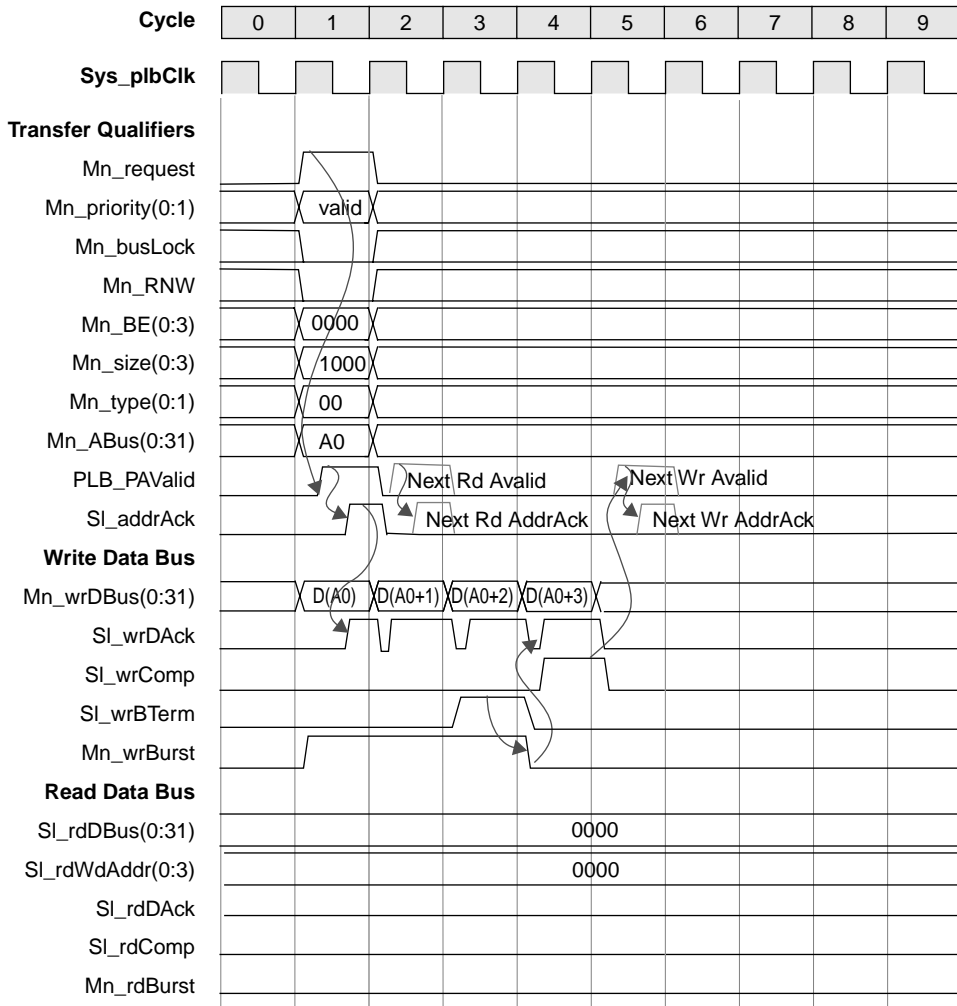


Figure 2-19. Burst Write Transfer Terminated By Slave

2.4.1.14 Fixed Length Burst Transfer - Notes

For PLB bandwidth critical situations, burst transfers can be used to maximize throughput. However, for back-to-back read burst transfers to single cycle slaves, there are two cycles in which the PLB_rdDBus cannot be utilized (see Figure 2-16 and Figure 2-17). In the case of a long burst, the two cycles may be acceptable, however, on short burst transfers these two cycles can significantly impact the overall throughput of the burst transfers. Additionally, some PLB slaves can improve their throughput during burst transfers if the length of the transfer is known when the transfer is requested.

In order to address this performance concern, an optional fixed length transfer protocol is provided and may be optionally implemented in both masters and slaves. This transfer is compatible with the existing burst protocol such that the burst transfers will occur using the normal transfer protocol for those masters and slaves which do not implement the fixed length transfer protocol.

During the request phase of a burst transfer, a master may indicate the number of byte, halfword, or word transfers by providing the length of the burst on the Mn_BE signals as shown in Table 2-12.

Table 2-12. Fixed Length Burst Transfer

Mn_BE(0:3)	Burst Length
0000	Burst length determined by PLB_rd/wrBurst signal
0001	Burst of 2
0010	Burst of 3
0011	Burst of 4
0100	Burst of 5
0101	Burst of 6
0110	Burst of 7
0111	Burst of 8
1000	Burst of 9
1001	Burst of 10
1010	Burst of 11
1011	Burst of 12
1100	Burst of 13
1101	Burst of 14

Table 2-12. Fixed Length Burst Transfer (cont.)

Mn_BE(0:3)	Burst Length
1110	Burst of 15
1111	Burst of 16

Note that the Burst length refers to the number of transfers of the data type selected by the Mn_size signals. The Mn_size = 0b1000 and Mn_BE = 0b1111 will transfer 16 bytes, Mn_size = 0b1001 and Mn_BE = 0b1111 will transfer 16 halfwords, and Mn_BE = 0b1111 and Mn_size = 0b1010 will transfer 16 words.

Masters which do not implement the fixed length transfer should drive all 0's on the BE signals to be compatible with slaves which have implemented the fixed length burst protocol. Slaves which do not implement the fixed length transfer will ignore the PLB_BE signals during a burst transfer and will continue bursting until the PLB_rd/wrBurst signal is negated by the master.

Slaves implementing the fixed length burst protocol must latch up the PLB_BE signals during the SI_addrAck clock cycle and use this value to count the number of transfers. If the PLB_rd/wrBurst signal is negated, then the slave must end the burst, regardless of the number of transfers remaining (based on the initial BE encoding).

For read burst transfers, if the PLB_rdBurst signal is not negated by the master, then the slave should assert SI_rdBTerm and SI_rdComp in the cycle prior to the last SI_rdDack. This will allow for a subsequent read or write transfer to be acknowledged in the SI_rdComp cycle and thus fully utilize the read data bus. However, it should be noted that if the SI_rdComp and SI_rdBTerm signals are asserted in the cycle prior to the last assertion of SI_rdDack, the slave must ignore the PLB_rdBurst signal in the following cycle. It may be asserted due to the arbiter switching to a new read burst transfer in the cycle following the assertion of the SI_rdComp.

For write burst transfers, if the PLB_wrBurst signal is not negated early, then the slave should assert SI_wrBTerm in the clock cycle prior to the last SI_wrDack and assert the SI_wrComp in the same clock cycle as the assertion of the last SI_wrDack. This will allow for a subsequent read or write transfer to be acknowledged in the cycle following the assertion of SI_wrComp and thus fully utilize the write data bus.

It is important to note that although the length of the transfer is provided to the slave during the request phase, the master is still required to assert the Mn_rd/wrBurst signal. Additionally, the master must negate the Mn_rd/wrBurst signal either in the clock cycle following the assertion of SI_rd/wrBTerm or in the clock cycle following the assertion of the second to last SI_rd/wrDack. This allows the transfer to complete when bursting to a slave which has not implemented the fixed length burst protocol.

2.4.1.15 Fixed Length Burst Read Transfer

Figure 2-20 shows the operation of the fixed length burst read from a slave device on the PLB. During the request phase of this transfer the master has optionally provided the length of the burst on the Mn_BE signals and is requesting to read four words. The slave uses this length value to count the number of transfers and assert SI_rdComp and SI_rdBTerm in the cycle prior to the last assertion of SI_rdDack. This allows a subsequent read transfer to be acknowledged.

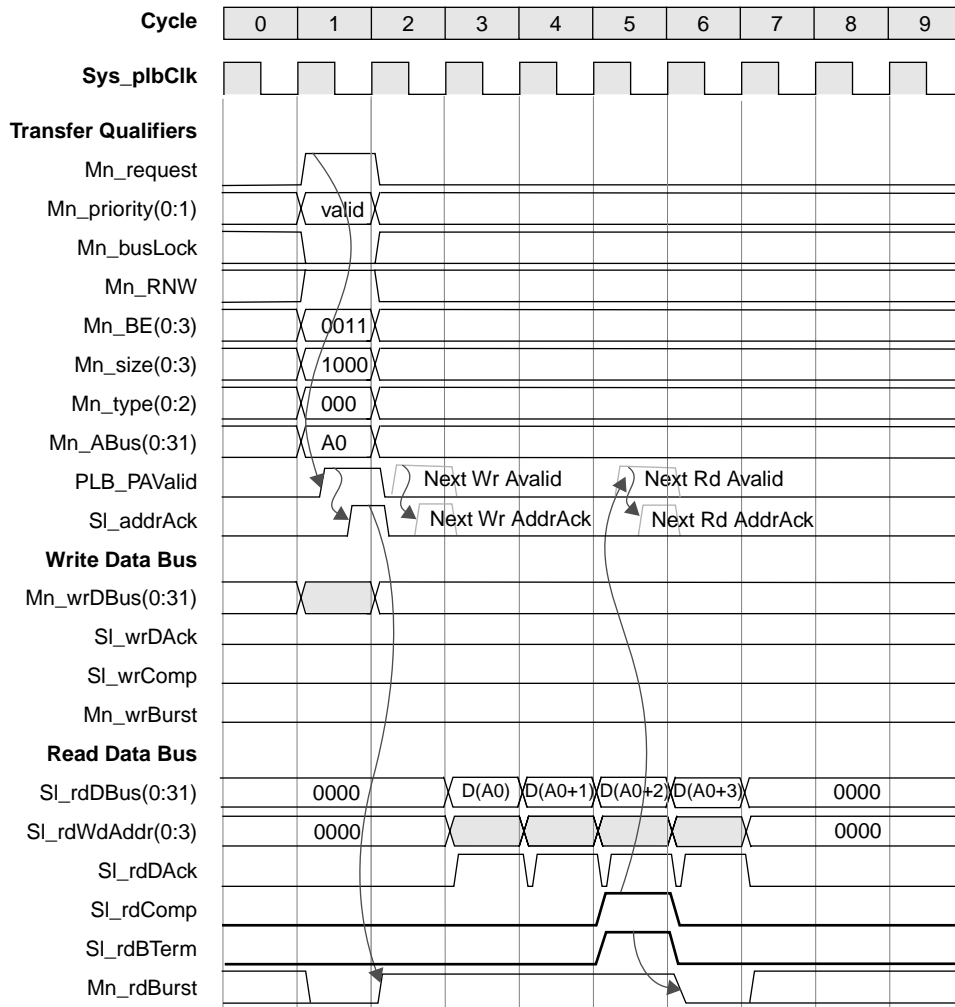


Figure 2-20. Fixed Length Burst Read Transfer

2.4.1.16 Fixed Length Burst Write Transfer

Figure 2-21 shows the operation of a fixed length burst write from a slave device on the PLB. During the request phase of the transfer the master has continuously provided the length of the burst on the Mn_BE signals and is requesting to write four words. The slave uses this length value to count the number of transfers and assert SI_wrBTerm in the cycle prior to the last assertion of SI_wrDack.

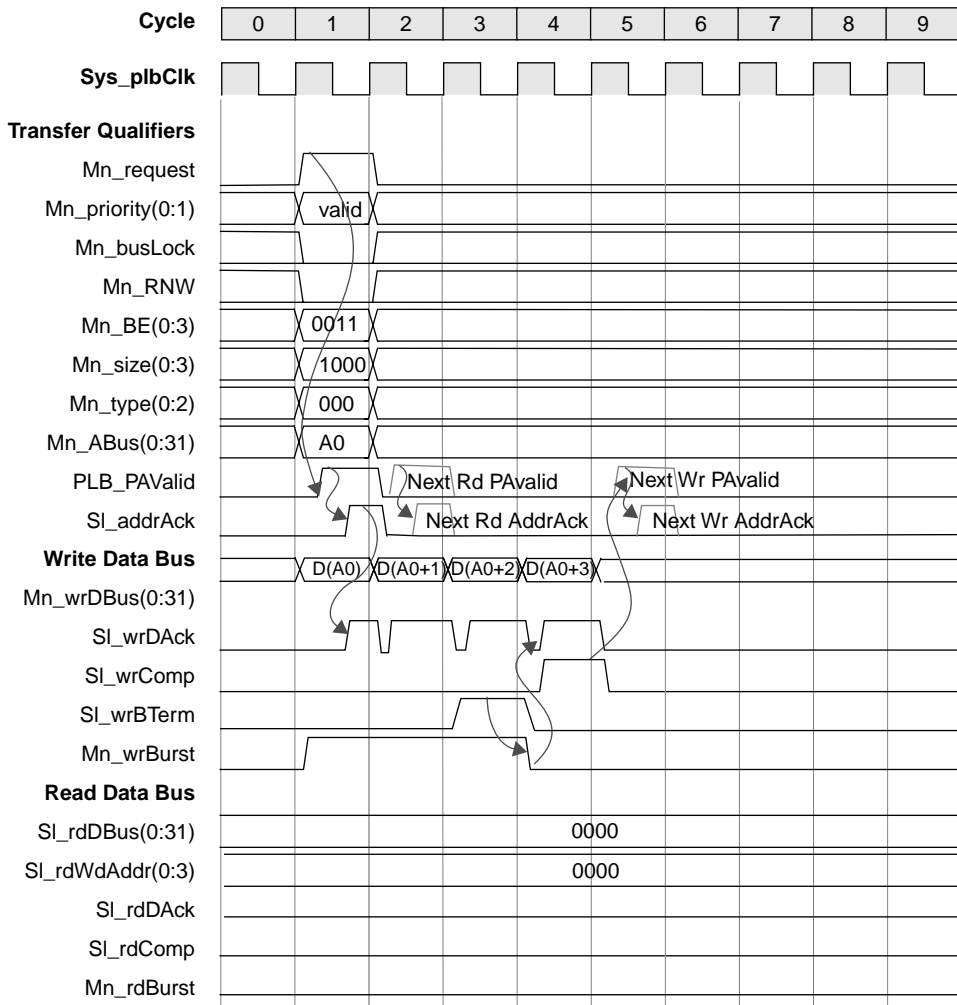


Figure 2-21. Fixed Length Burst Write Transfer

2.4.1.17 Back-to-Back Burst Read - Burst Write Transfers

Figure 2-22 shows the operation of a burst read followed immediately by a request for a burst write transfer on the PLB. Note that the address bus and transfer qualifiers are only required to be driven by the master until the address has been acknowledged by the slave. This allows the burst write request and write data transfers on the PLB write data bus to occur completely overlapped with the burst read that is on-going on the PLB read data bus. These burst transfers may continue up to the maximum burst length that is supported by the slave device.

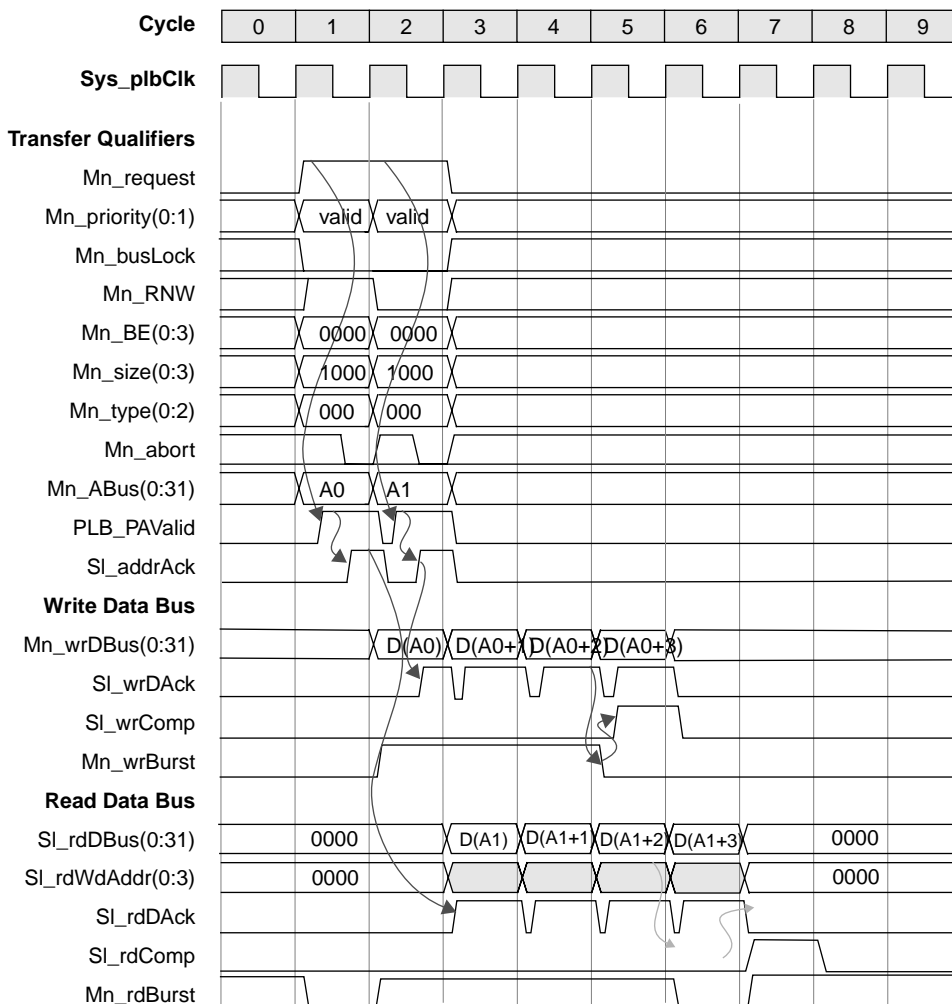


Figure 2-22. Back-to-Back Burst Read - Burst Write Transfers (Wait = 0, Hold = 0)

2.4.1.18 Locked Transfer

Figure 2-23 shows the operation of a locked data transfer on the PLB. A first master asserts its Mn_busLock signal to indicate to the arbiter that it wishes to lock the bus during the current data transfer. Although not illustrated in the diagram, the arbiter asserts PLB_PAVali only after detecting that both data busses are idle. The slave then asserts the SI_addrAck signal, causing the arbiter to lock the bus. A second master request is ignored by the arbiter until the first master negates its Mn_busLock signal. On the clock cycle following the clock cycle in which the first master negates its Mn_busLock signal, the PLB is re-arbitrated and granted to the second master.

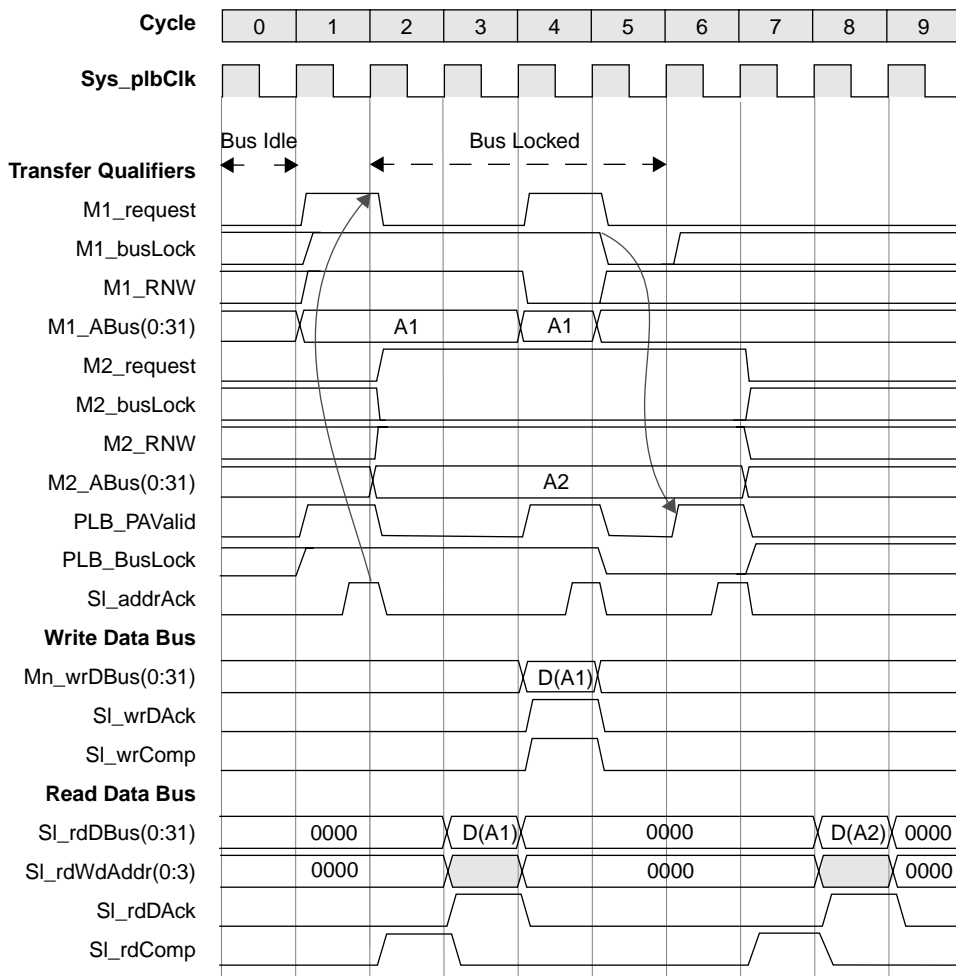


Figure 2-23. Locked Transfer

2.4.1.19 Slave Requested Re-arbitration With Bus Unlocked

Figure 2-24 illustrates a scenario in which a master/slave device is unable to respond to a PLB data transfer initiated by another master until it has first executed a PLB transfer of its own. As a result, the master/slave device asserts its SI_rearbitrate signal to request re-arbitration of the PLB bus. In response to the assertion of the SI_rearbitrate signal, the PLB arbiter “backs-off” the initial request and re-arbitrates the bus in the following clock cycle, allowing the master/slave device to have its request serviced ahead of the initial request. Note that the PLB_PValid signal is never dropped, instead the arbiter simply gates the newly arbitrated request onto the PLB on the clock cycle immediately following the clock cycle in which the SI_rearbitrate signal was asserted.

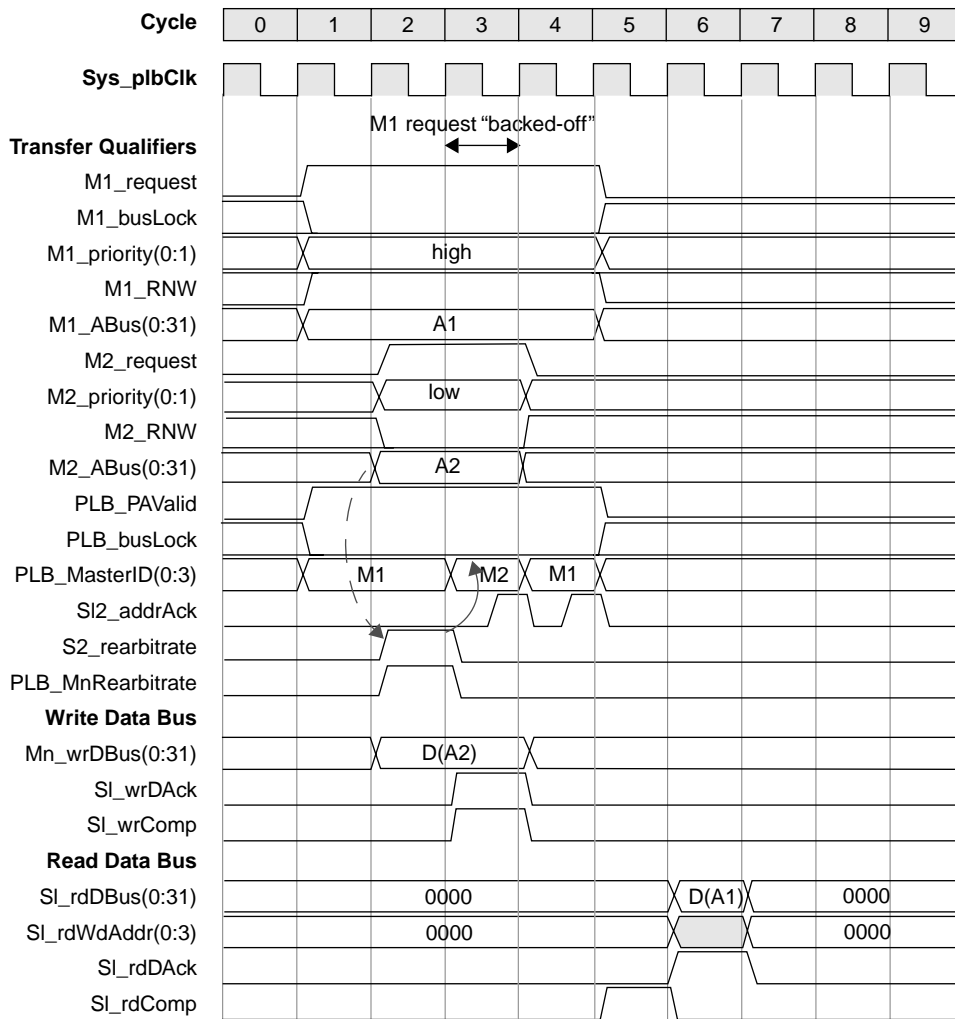


Figure 2-24. Slave Requested Re-arbitration With Bus Unlocked

2.4.1.20 Slave Requested Re-arbitration With Bus Locked

Figure 2-25 illustrates a scenario in which a master/slave device is unable to respond to a PLB data transfer initiated by another master until it has first executed a PLB transfer of its own. As a result, the master/slave device asserts its SI_rearbitrate signal to request re-arbitration of the PLB bus. In response to the assertion of the PLB_M1Rearbitrate signal, master 1 negates the M1_request and M1_busLock signals for a minimum of two clocks, allowing the PLB arbiter to re-arbitrate the bus in the following clock cycle and the master/slave device request to be serviced ahead of its initial request, averting a possible dead-lock scenario.

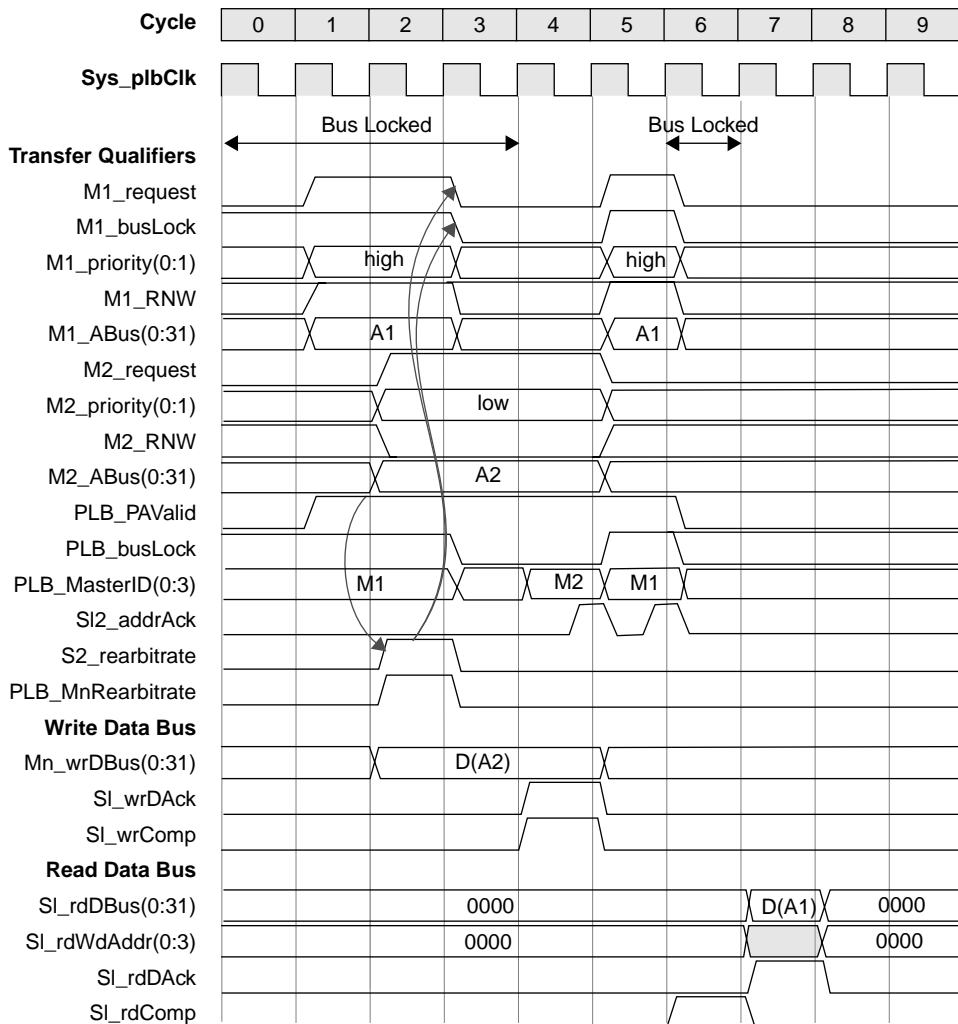


Figure 2-25. Slave Requested Re-arbitration With Bus Locked

2.4.1.21 Bus Time-Out Transfer

Figure 2-26 shows a bus time-out for a read transfer on the PLB. The PLB arbiter asserts the PLB_MnAddrAck signal, sixteen cycles after the initial assertion of the PLB_PValid signal. Two cycles after PLB_MnAddrAck is asserted, the arbiter completes the handshaking to the master by asserting the PLB_MnRdDack and PLB_MnErr signals. Note that the arbRdComp signal is not part of the PLB spec but is included here to illustrate when the PLB arbiter is ready to arbitrate the next read transfer.

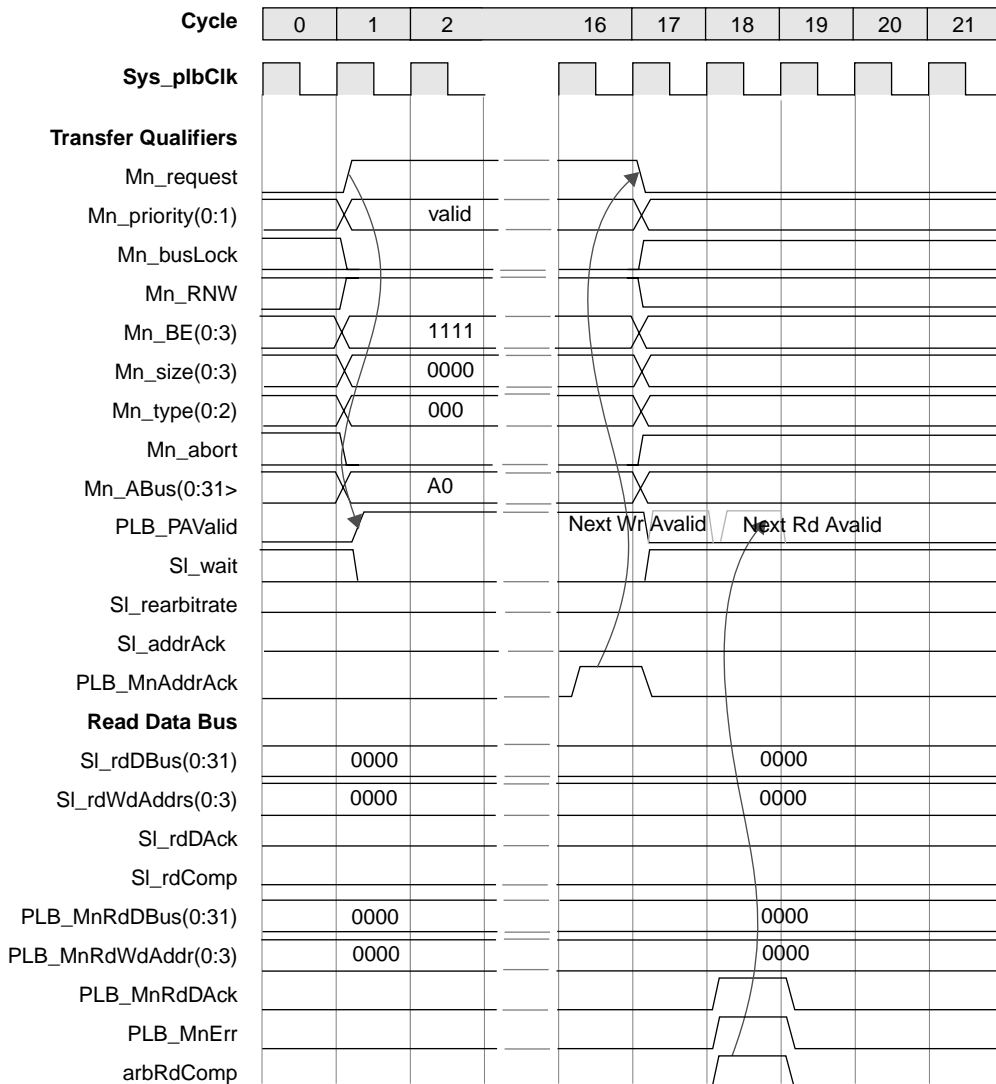


Figure 2-26. Bus Time-Out Transfer

2.4.1.22 Bus Transfer Time-Out Notes

As mentioned previously, once PLB_PAVValid has been asserted for a master request, the PLB arbiter will wait for sixteen clock cycles for the assertion of either SI_rearbitrate, SI_wait, SI_addrAck, or Mn_abort. If neither of these signals is sampled asserted within sixteen clock cycles, the PLB arbiter will time-out and complete the necessary handshaking to the master. More specifically, the PLB arbiter will assert PLB_MnAddrAck to complete the address cycle, and PLB_MnRdDack/PLB_MnWrDack to complete the data read/write portion of the transfer. The PLB arbiter will also assert the PLB_MnErr signal during each data acknowledge phase.

For line transfers, the number of data transfer cycles run after a bus time-out is detected, will be determined by the size of the transfer as defined by the PLB_size(0:3) during the address cycle. More specifically, the PLB_MnRdDack/PLB_MnWrDack and PLB_MnErr signals will be asserted four times for a 4-word line transfer, eight times for a 8-word line transfer, and sixteen times for a 16-word line transfer.

For burst read transfers (as indicated by the PLB_size(0:3) signals), if the PLB_rdBurst signal is detected asserted, the PLB arbiter will assert the PLB_MnRdBTerm signal to indicate to the master that the transfer should be terminated, and the master will negate its Mn_rdBurst signal in the following clock cycle. Following the negation of PLB_rdBurst, the PLB arbiter will assert the PLB_MnRdDack and PLB_MnErr signals for one, and only one, additional clock cycle.

For burst write transfers (as indicated by the PLB_size(0:3), if the PLB_wrBurst signal is detected asserted, the PLB arbiter will assert the PLB_MnWrBTerm signal to indicate to the master that the transfer should be terminated, and the master will negate its Mn_wrBurst signal in the following clock cycle. Following the negation of the PLB_wrBurst signal, the PLB arbiter will assert the PLB_MnWrDack and PLB_MnErr signals for one additional clock cycle.

Note: A timed-out master request with the Mn_busLock signal asserted is a special case in that it will not result in the PLB arbiter locking the bus, if previously unlocked.

2.4.2 PLB Address Pipelining

The timing diagrams included in this section are examples of address-pipelined read and write transfers on the PLB. However, it is important to note that signal assertion and negation times as shown in these diagrams, are only meant to illustrate their dependency on the rising edge of Sys_plbClk and in no way are they intended to show real signal timing.

2.4.2.1 Pipelined Back-to-Back Read Transfers

Figure 2-27 shows the operation of three back-to-back read transfers involving three masters and a slave device which support address pipelining on the PLB. For all transfers, the slave asserts the SI_rldComp signal in the clock cycle preceding the SI_rldAck. This allows the next master's read request to be sent to slaves in the clock cycle preceding the data transfer cycle on the PLB. For the primary read request, the slave may not assert its SI_rldAck for the data read until two clock cycles following the assertion of the corresponding SI_addrAck. For the secondary read requests, the slave may not assert its SI_rldAck or drive the SI_rldDBus until two clock cycles following the assertion of PLB_rldPrim. This allows time for the previous read data transfers to complete before the data is transferred for the subsequent read. Using this protocol, a master may read data every clock cycle from a slave which is capable of providing data in a single clock cycle.

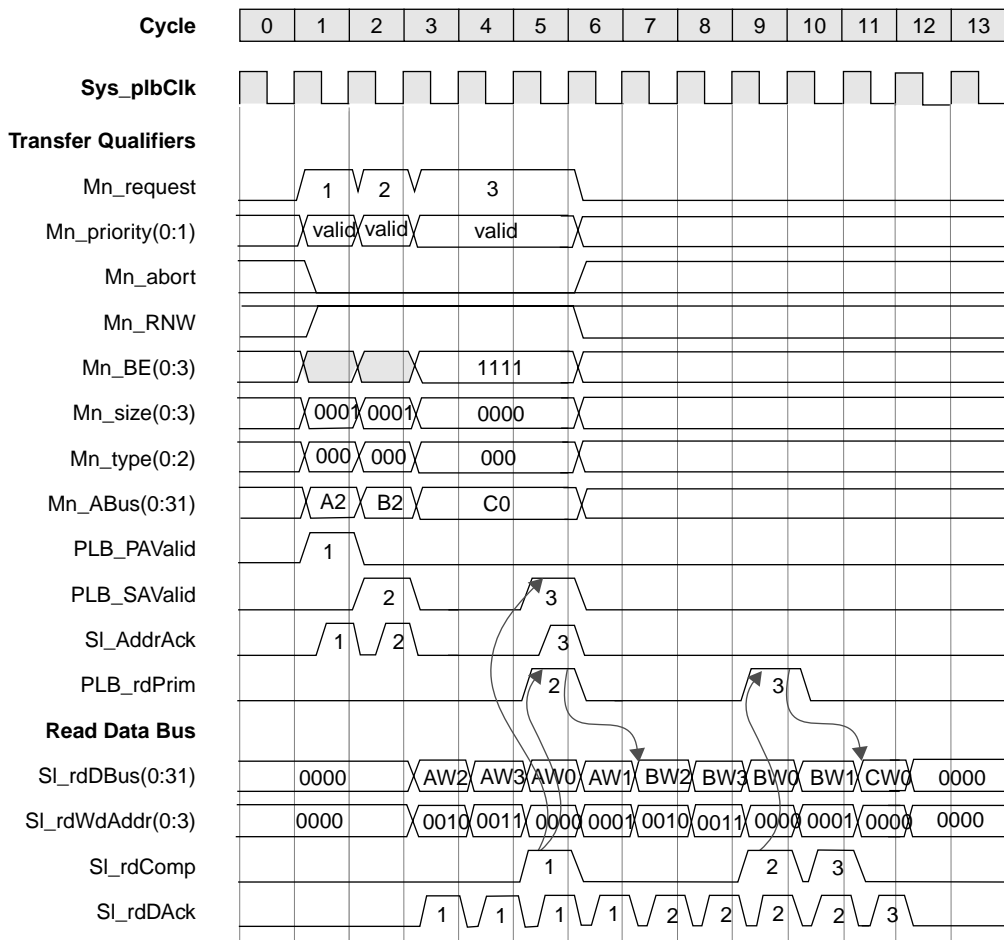


Figure 2-27. Pipelined Back-to-Back Read Transfers

2.4.2.2 Pipelined Back to Back Read Transfers - Delayed AAck

Figure 2-28 is similar to Figure 2-27, with one exception. For master 3's request, PLB_SAVValid is negated and PLB_PAVValid is asserted prior to the slave's assertion of SI_addrAck. Note that the assertion of PLB_PAVValid for the last read request is made possible by the assertion of SI_rdComp for the previous secondary request. Note also that PLB_rdPrim is not asserted for this request.

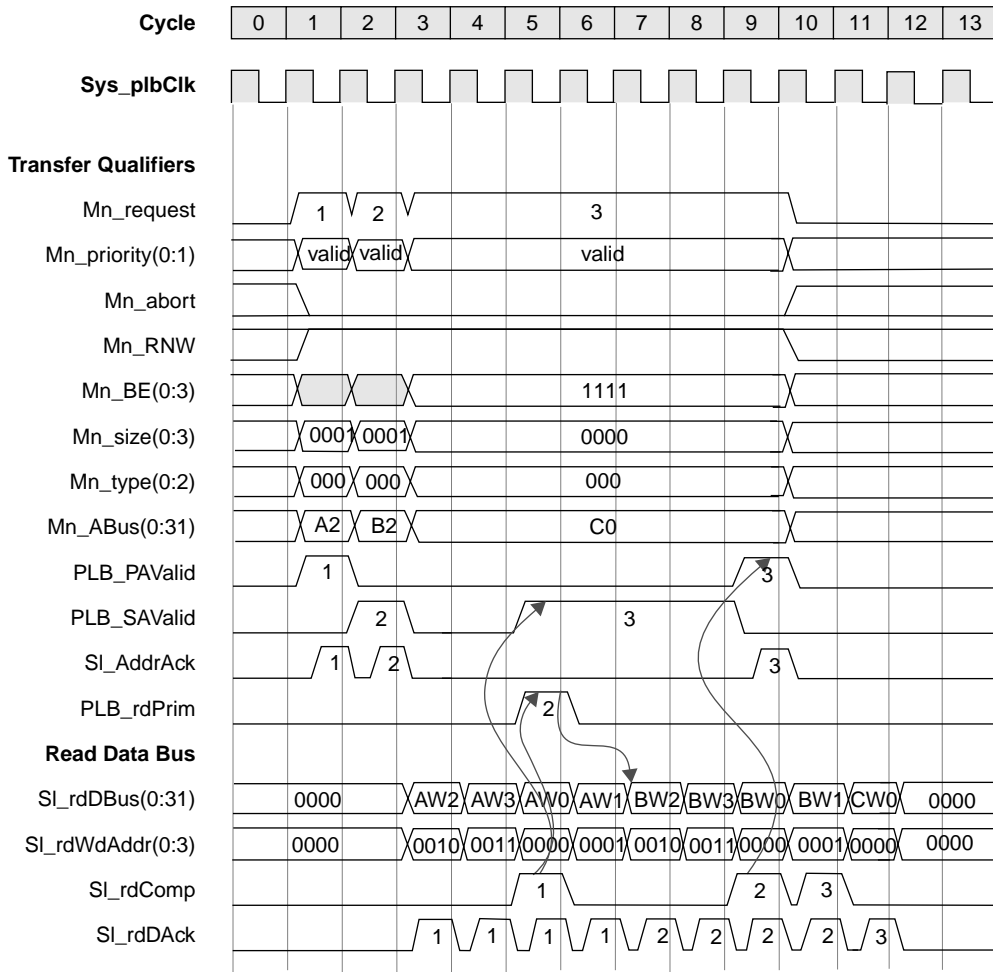


Figure 2-28. Pipelined Back-to-Back Read Transfers - Delayed AAck

2.4.2.3 Pipelined Back-to-Back Write Transfers

Figure 2-29 shows the operation of three back-to-back write transfers involving three masters and a slave device which support address pipelining on the PLB. For the primary write request, the slave may assert the SI_wrComp and SI_wrDack signals in the same clock cycle SI_addrAck is asserted. For the secondary write requests, the slave may not assert its SI_wrDack for the written data until the clock cycle following the assertion of PLB_wrPrim. It is important to note that for the case of a same master having requested both a primary and a secondary request, the master must drive the first piece of data for the secondary request in the clock cycle following the last data transfer for the primary request.

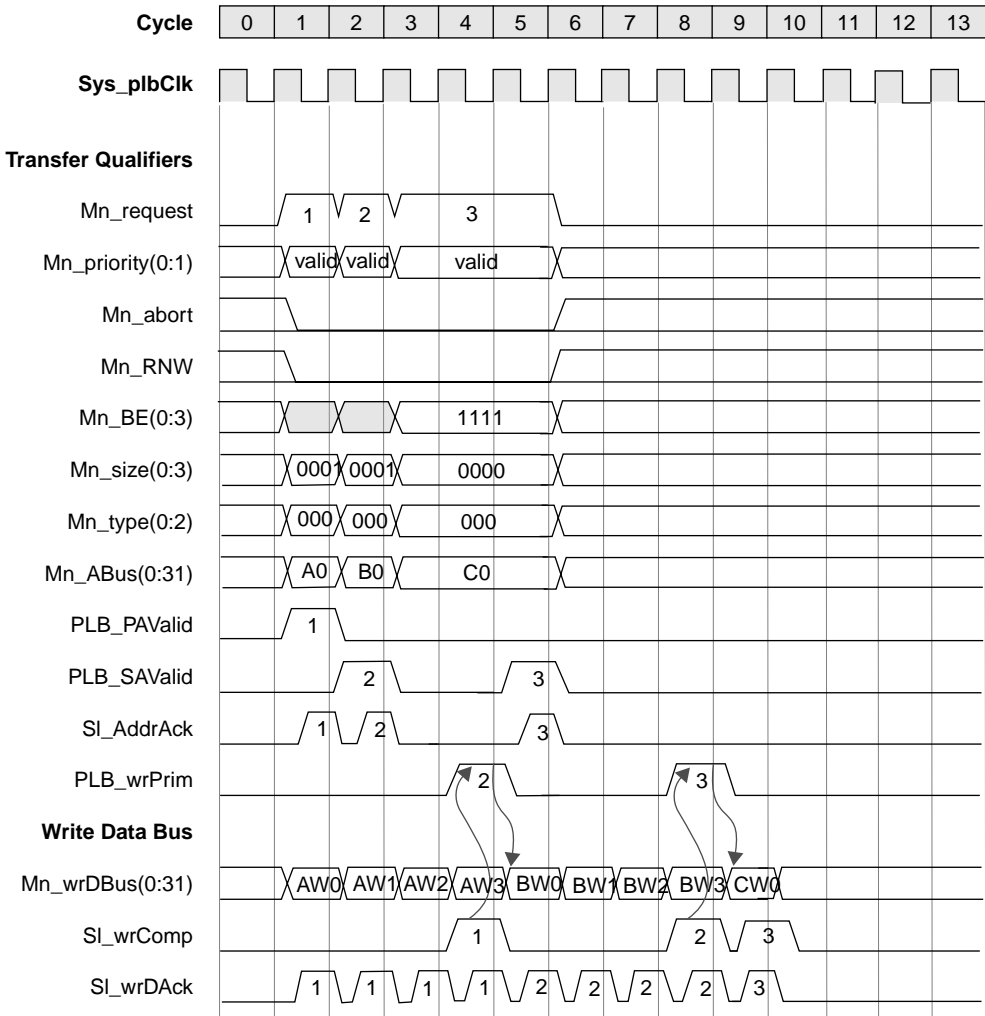


Figure 2-29. Pipelined Back-to-Back Write Transfers

2.4.2.4 Pipelined Back-to-Back Write Transfers - Delayed AAck

Figure 2-30 is similar to Figure 2-29, with one exception. For master 3's write request in the series, PLB_SAValid is negated and PLB_PAVValid is asserted prior to the slave's assertion of SI_addrAck. Note that the assertion of PLB_PAVValid for the last write request is made possible by the assertion of SI_wrComp for the previous secondary request. Note also that PLB_wrPrim is not asserted for this request.

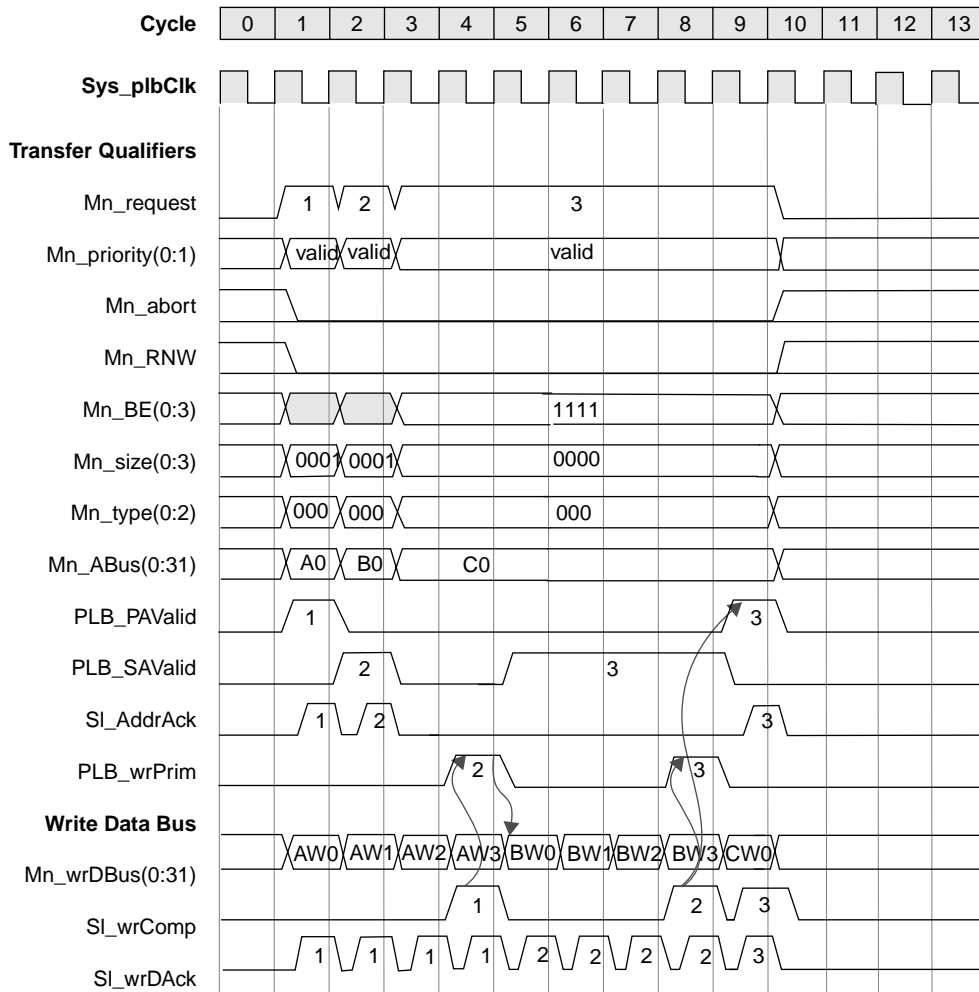


Figure 2-30. Pipelined Back-to-Back Write Transfers - Delayed AAck

2.4.2.5 Pipelined Back-to-Back Read and Write Transfers

Figure 2-19 shows the operation of four back-to-back read and write transfers involving four masters and a slave device which support address pipelining on the PLB.

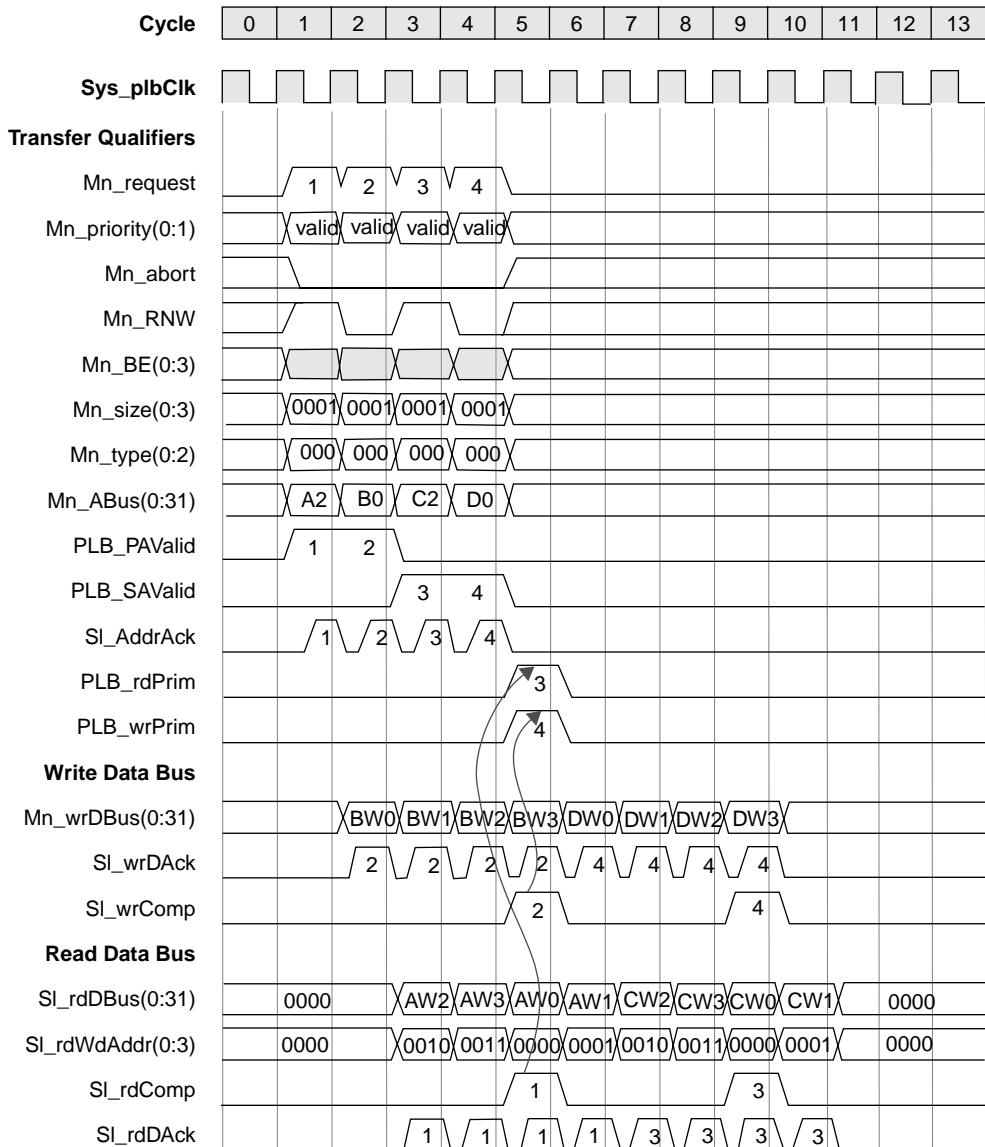


Figure 2-31. Pipelined Back-to-Back Read and Write Transfers

2.4.2.6 Pipelined Back-to-Back Read Burst Transfers

Figure 2-32 shows the operation of two back-to-back read burst transfers involving a master and a slave device which support address pipelining on the PLB. Note that the Mn_rdBurst signal must be negated during the last data transfer for the first request before it is re-asserted for the second request.

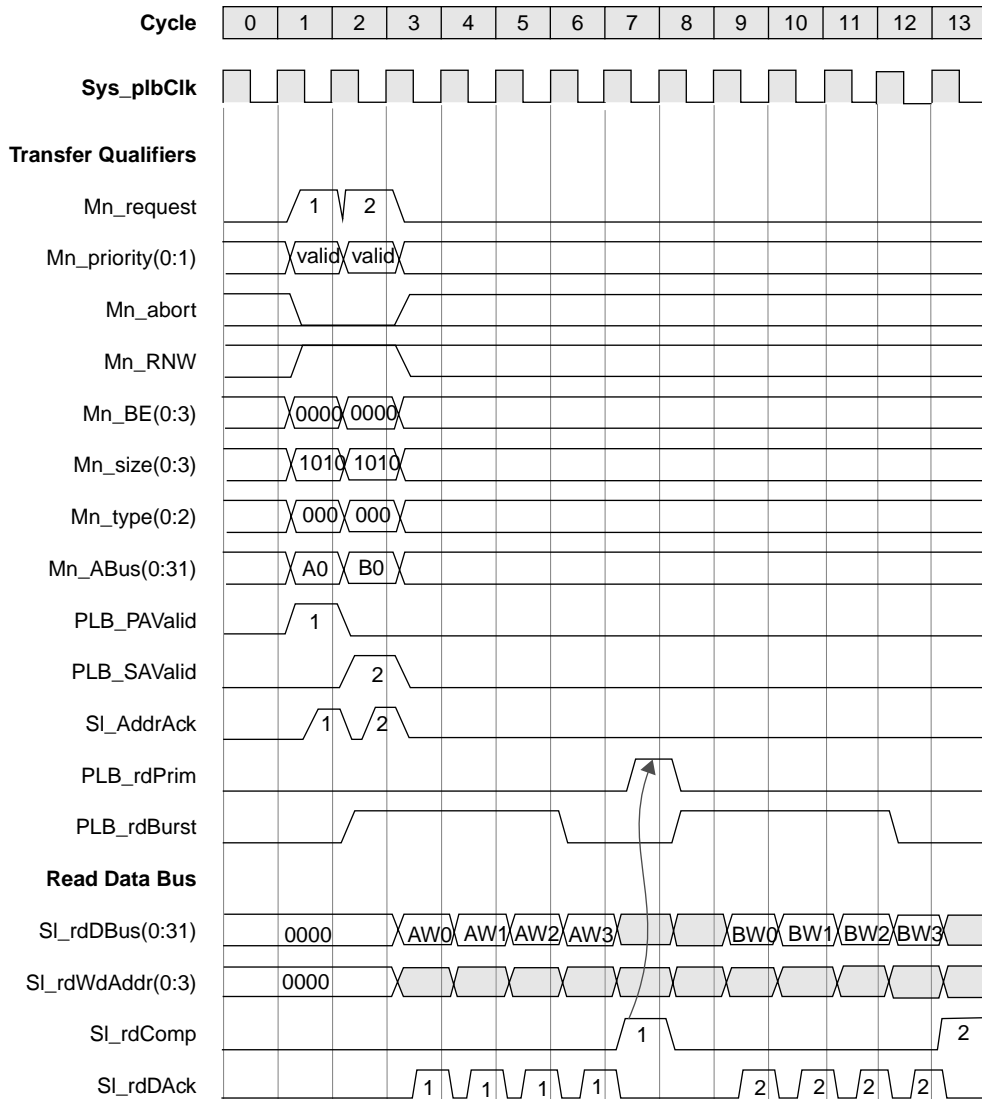


Figure 2-32. Pipelined Back-to-Back Read Burst Transfers

2.4.2.7 Pipelined Back-to-Back Write Burst Transfers

Figure 2-33 shows the operation of two back-to-back write burst transfers involving a master and a slave device which support address pipelining on the PLB. Note that the Mn_wrBurst signal must be re-asserted for the second request in the cycle immediately following the last data transfer for the first request.

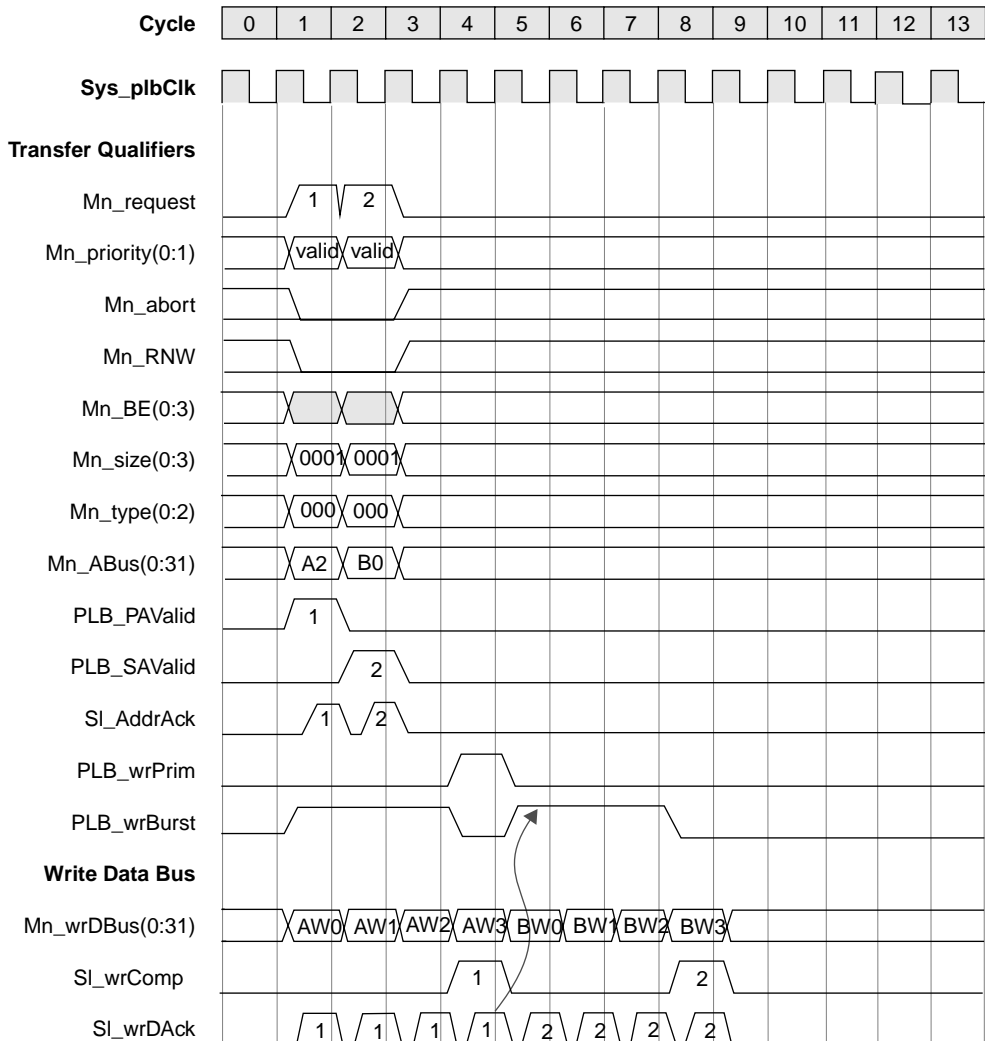


Figure 2-33. Pipelined Back-to-Back Write Burst Transfers

2.4.3 PLB Bandwidth and Latency

High bandwidth (throughput) can be achieved by allowing PLB devices to transfer data using long burst transfers. However, to control the maximum throughput (and hence, maximum latency) in a particular application, a master latency timer is provided in each master.

2.4.3.1 PLB Master Latency Timer

The master latency timer is a programmable timer which limits a master's tenure on the PLB bus when using burst transfers. Each master capable of performing a burst of at least two data transfers is required to have a latency timer. Two registers are required to implement the master latency timer:

1. The Latency Count Register (8-bits, 4 low-order bits may be hardwired)
2. The Latency Counter (8-bits)

The Latency Count Register is programmable, and can be read or written by software, and can be either memory mapped or DCR bus mapped. The four low-order bits of the Latency Count register may be hardwired such that the minimum latency value is sixteen clock cycles and the granularity of the Latency Count is sixteen clock cycles (that is, you could program latency counts of 16, 32, 48, etc. clock cycles).

The Latency Counter is used as clock cycle counter and is not accessible via code. The Latency Counter is cleared and disabled when the master is not performing a burst data transfer on the bus. During burst data transfers, the Latency Counter is enabled and will begin counting the clock cycle after the PLB_MnAddrAck signal is asserted by the slave device. Upon expiration of the Latency Counter, if a request of equal or higher priority is pending on the PLB, the master is required to negate its burst signal and thus cause the slave device to terminate the burst transfer by asserting its SI_rdComp or SI_wrComp signal. To facilitate compliance with this requirement, anyone of the three following options can be implemented in a master:

1. The master may monitor the PLB_pendReq and PLB_pendPri(0:1) signals continuously and negate the burst signal immediately after the Latency Counter has expired and a pending request of equal or higher priority is detected.
2. The master may monitor the PLB_pendReq signal continuously and negate the burst signal immediately after the Latency Counter has expired and a pending request is detected.
3. The master may negate the burst signal immediately after the Latency Counter has expired.

Note: With options 1 and 2, the master must keep its own request signal negated during the burst in order to determine if other requests are pending.

Both the Latency Count Register and the Latency Counter should be designed such that they will be cleared by Reset. Additionally, if a master is pipelining transfers and receives a secondary address acknowledge prior to the completion of the initial burst transfer, the master may reset the latency counter and continue the initial burst transfer until the latency timer expires. However, if the latency counter expires after the secondary address

acknowledge and a request of equal or higher priority is pending on the PLB the master is required to terminate both the primary burst transfer and the secondary transfer and thus relinquish control of the bus.

2.5 PLB/DMA Operations

This section on PLB/DMA operations discusses PLB/DMA interface with PLB/DMA sideband and DMA transfer signals followed by PLB based DMA transfer operations with timing diagrams that describe how the PLB may be used to perform third party DMA operations.

2.5.1 PLB/DMA Interface

The PLB/DMA interface consists of PLB/DMA side band and DMA transfer signals. Figure 2-34 shows all input output signals.

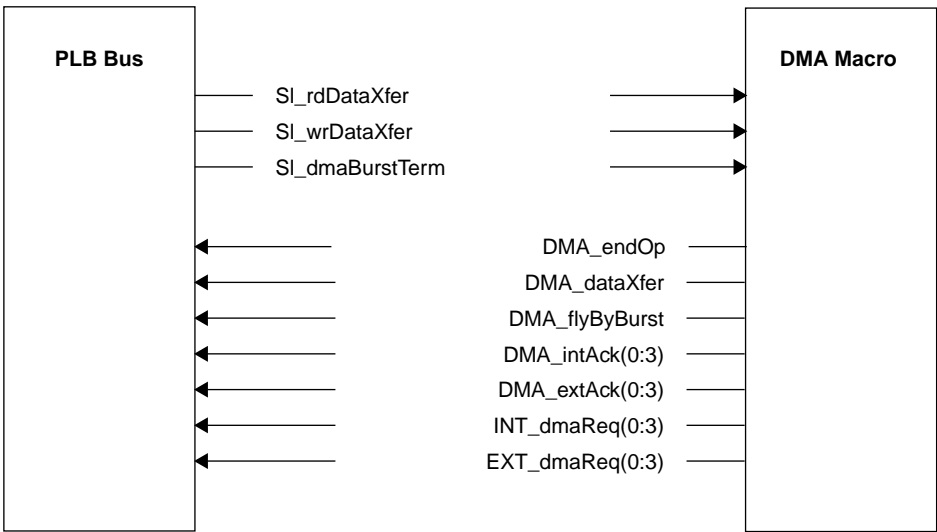


Figure 2-34. PLB/DMA Interface

2.5.2 PLB/DMA Signals

Table 2-13 provides a summary of all DMA input/output signals in alphabetical order, the interfaces under which they are grouped, followed by a brief description and page reference for detailed functional description.

Table 2-13. Summary of DMA Signals

Signal Name	Interface	I/O	Description	Page
DMA_data Xfer	PLB	I	DMA data transfer	2-75
DMA_flyByBurst	PLB	I	DMA flyby burst	2-75
DMA_endOp	PLB	I	DMA end	2-75
DMA_intAck(0:3)	PLB	I	Internal DMA transfer acknowledge	2-76
DMA_extAck(0:3)	PLB	I	External DMA transfer acknowledge	2-76
EXT_dmaReq(0:3)	PLB	I	External DMA transfer request	2-76
INT_dmaReq(0:3)	PLB	I	Internal DMA transfer request	2-76
SI_rdDataXfer	PLB	O	Slave read data transfer	2-74
SI_wrDataXfer	PLB	O	Slave write data transfer	2-75
SI_dmaBurstTerm	PLB	O	Slave DMA burst terminate	2-76

2.5.3 PLB/DMA Side-Band Signals

2.5.3.1 SI_rdDataXfer (Slave Read Data Transfer)

This signal is driven by the PLB slave and its meaning depends on the type of DMA transfer being performed.

During DMA flyby read transfers (PLB_type = 001) this signal is asserted to indicate to the DMA controller that the memory read transfer has been completed, not including any hold time. The signal is used by the DMA controller to negate the DMA_Ack signal to the peripheral and thus synchronize the transfer. During DMA FlyBy burst read transfers this signal is asserted in the last clock cycle of wait time for each data transfer of the burst and is used by the DMA controller to increment to the next data transfer or negate the DMA_Ack signal to complete the burst.

During PLB slave buffered memory to peripheral transfer (PLB_type = 101), this signal is asserted to indicate that the slave has latched the read data in its local buffer or will latch the read data at the end of this cycle. This signal is used to indicate to the DMA controller that the read transfer is complete including any hold time and that the peripheral write transfer may start in the following clock cycle.

During DMA memory read line transfers (PLB_type = 110) this signal is asserted to indicate to the DMA controller that the last word of the transfer will be available on the PLB bus in the next clock cycle. Note that the signal is only asserted once, in the clock cycle prior to the last SI_rdDack on the PLB bus. The signal is used by the DMA controller to overlap the DMA read and write line transfers on the PLB bus.

2.5.3.2 SI_wrDataXfer (Slave n Write Data Transfer)

This signal is driven by the PLB slave and its meaning depends on the type of DMA transfer being performed.

During DMA flyby write transfers (PLB_type = 001) this signal is asserted to indicate to the DMA controller that the memory write transfer has been completed, not including any hold time. The signal is used by the DMA controller to negate the DMA_Ack signal to the peripheral and thus synchronize the transfer. During DMA FlyBy burst write transfers this signal is asserted in the last clock cycle of wait time for each data transfer of the burst and is used by the DMA controller to increment to the next data transfer or negate the DMA_Ack signal to complete the burst.

During PLB slave buffered memory to memory write transfers with side-band signals (PLB_type= 111), and DMA memory write transfers (PLB_type = 110) this signal is asserted by the PLB slave to indicate to the DMA controller the clock cycle of the last data transfer, not including any hold time. This signal is used by the DMA controller to indicate when to sample the DMA request signal during device paced memory to memory transfers.

2.5.3.3 DMA_endOp (DMA End of Operation)

This signal is asserted by the DMA controller during a write transfer to a peripheral device to indicate that the PLB slave bus controller may stop driving data. This signal is also asserted by the DMA controller during a read transfer from a peripheral device to indicate the end of data hold time.

2.5.3.4 DMA_dataXfer (DMA Data Transfer)

This signal is asserted by the DMA controller during a peripheral read transfer to indicate that the PLB slave bus controller must latch the data currently being driven by the peripheral device.

2.5.3.5 DMA_flyByBurst (DMA Flyby Burst)

When a DMA FlyBy burst read or write transfer is acknowledged by a PLB slave device the DMA controller will drive DMA_flyByBurst signal to the PLB slaves. The PLB slaves must sample the DMA_flyByBurst signal in the clock cycle in which the PLB slave asserts either SI_wrDataXfer or SI_rdDataXfer to determine when to terminate the burst transfer. A high value indicates that the DMA requires additional bytes, halfwords, or words of data beyond the current transfer. A low value indicates that the current transfer is the last data transfer that the DMA controller is requesting and no further data transfers should be performed by the PLB slave.

2.5.3.6 SI_dmaBurstTerm (Slave n DMA Burst Terminate)

This signal is driven by a PLB slave during DMA FlyBy burst transfers to indicate that the current burst transfer in progress must be terminated. This signal may be asserted by the PLB slave starting one clock cycle following the assertion of the SI_addrAck, up to and including the clock cycle in which the second to last SI_xxDataXfer of the burst is asserted. Note that the DMA will only sample the SI_dmaBurstTerm signal during a clock cycle in which SI_xxDataXfer signal is also asserted. In response to the assertion of the SI_dmaBurstTerm and SI_xxDataXfer signals, the DMA controller will complete the DMA FlyBy transfer by negating the DMA_flyByBurst signal in the following clock cycle. This signal may only be asserted while the PLB slave is busy performing a DMA FlyBy burst transfer.

2.5.4 DMA Transfer Signals

The DMA transfer signals described below are only a subset of the signals used in the external and OPB DMA peripherals. For a complete description of the internal workings of the DMA controller and additional DMA transfer signals please refer to the DMA User's Manual.

2.5.4.1 INT_dmaReq(0:3) (Internal DMA transfer request)

These signals are asserted by the on-chip DMA peripherals to request a DMA data transfer. The assertion of these signals is single latched, but their negation is synchronous relative to the assertion of the DMA_intAck(0:3) signals.

2.5.4.2 DMA_intAck(0:3) (Internal DMA transfer acknowledge)

These signals are asserted by the DMA controller to indicate that an on-chip DMA peripheral is selected and must provide or receive data.

2.5.4.3 EXT_dmaReq(0:3) (External DMA transfer request)

These signals are asserted by the off-chip DMA peripherals to request a DMA data transfer. The assertion of these signals is double latched, but their negation is synchronous relative to the assertion of the DMA_extAck(0:3) signals.

2.5.4.4 DMA_extAck(0:3) (External DMA transfer acknowledge)

These signals are asserted by the DMA controller to indicate that an off-chip DMA peripheral must provide or receive data.

2.5.5 DMA Transfer Operations

2.5.5.1 DMA Flyby Transfers (Mn_type = 0b001)

In DMA flyby transfers data is transferred directly between a DMA peripheral and a memory device, on either an external or an on-chip peripheral bus. The DMA peripheral is controlled by the DMA controller while the memory device is controlled by a PLB slave bus controller (SBC).

When the DMA requests a flyby transfer, the PLB SBC decodes the memory address on PLB_ABus(0:31) and asserts SI_addrAck as if it was a normal memory transfer. The PLB_RNW will indicate if memory will be read or written. The assertion of SI_addrAck implies that the PLB SBC is ready to perform the flyby transfer. In the next clock cycle, the DMA controller will assert the DMA_Ack. The PLB SBC will disable its data bus drivers and assert the memory address and controls. When the memory clock cycle is completed, the SBC asserts either SI_rdDataXfer for a memory read, or SI_wrDataXfer for a memory write to the DMA controller.

Note: Since there is no data transfer across the PLB, the SBC must not drive the SI_rdDack anytime during these transfers.

2.5.5.2 DMA Buffered External Peripheral Transfers (Mn_type = 0b010)

DMA buffered external peripheral transfers are decoded explicitly by Mn_type = 0b010. There may be only one PLB slave which responds to this transfer. The PLB_ABus(0:31) bus is invalid for this transfer type. Once the PLB SBC decodes this transfer, it responds by asserting SI_addrAck. The PLB_RNW indicates if data will be read from or written to the peripheral, and the PLB_BE(0:3) signals indicate which bytes are being transferred as well as the width of the peripheral.

For a peripheral write transfer, the PLB SBC will latch the data on PLB_wrDBus(0:31) and assert SI_wrDack simultaneously with SI_addrAck. This will complete the PLB portion of the transfer. The assertion of SI_addrAck implies that the PLB SBC is ready to perform a peripheral transfer and will drive the data onto the external data bus in the following clock cycle. Also in the clock cycle following the assertion of SI_addrAck, the DMA controller asserts DMA_extAck. DMA_extAck will be kept asserted for the programmed number of clocks and the PLB SBC will continue to drive the external data bus until DMA_endOp is asserted by the DMA controller. This will complete the peripheral write transfer and the SBC now can respond to any other PLB requests.

For a peripheral read transfer, the PLB SBC asserts SI_addrAck only, and waits until it samples DMA_dataXfer asserted. The DMA controller asserts DMA_extAck for the preprogrammed number of clock cycles and then asserts DMA_dataXfer during the last active clock cycle of DMA_extAck. This indicates to the PLB SBC to latch the data off the external data bus and drive it onto SI_rdDBus(0:31) in the following clock cycle. The assertion of SI_rdDack with this data completes the DMA peripheral read transfer.

2.5.5.3 DMA Buffered OPB Peripheral Transfer (Mn_type = 0b011)

DMA buffered OPB peripheral transfers are decoded explicitly by Mn_type = 0b011. The PLB_ABus(0:31) bus is invalid for this transfer type. Once the OPB bridge decodes this transfer, it responds with SI_addrAck. The PLB_RNW indicates if data will be read from or written to the peripheral, and the PLB_BE(0:3) signals indicate which bytes are being transferred as well as the width of the peripheral.

For a peripheral write transfer the OPB bridge will latch the data on PLB_wrDBus(0:31) and assert SI_wrDAck simultaneously with SI_addrAck. This will complete the PLB portion of the transfer. The assertion of SI_addrAck implies that the OPB bridge is ready to perform a peripheral transfer and will drive the data onto the OPB data bus in the following clock cycle. Also in the clock cycle following the assertion of SI_addrAck, the DMA controller asserts DMA_intAck. DMA_intAck will be kept asserted for the programmed number of clocks and the OPB bridge will continue to drive the data bus until DMA_endOp is asserted. This completes the peripheral write transfer to the OPB bridge, which now can respond to any other PLB requests.

For a peripheral read transfer the OPB bridge asserts SI_addrAck only, and waits until it samples DMA_dataXfer asserted. The DMA controller asserts DMA_intAck for the preprogrammed number of clock cycles and then asserts DMA_dataXfer during the last active clock cycle of DMA_intAck. This indicates to the OPB bridge to latch the data off the OPB data bus and drive it onto SI_rDBus(0:31) in the following clock cycle. The assertion of SI_rDAck with this data completes the DMA peripheral read transfer.

2.5.5.4 PLB Slave Buffered Memory to Memory Transfers (Mn_type = 0b100)

PLB slave buffered memory to memory transfers are used by a DMA master to request that data be read into or written from a PLB SBC's buffer. In this type of DMA transfer the DMA controller performs memory to memory operations with both memory source and memory destination residing on the same slave bus. The use of the slave buffer increases performance because the transfer of data is limited to the slave bus and does not need to be transferred across the PLB bus and through the DMA.

The PLB slave buffered memory to memory transfer is accomplished via two PLB transfers. First, the DMA controller requests a PLB slave buffered memory to memory read operation which will cause the SBC to read the data into its memory buffer. During the read request the DMA must assert the PLB_buslock and it must remain asserted until the clock cycle prior to the assertion of Mn_request for the write transfer. This guarantees the atomic nature of the operation and that the contents of the write buffer are not corrupted. The SBC will complete the PLB read transfer by asserting the SI_rDComp in the clock cycle following the SI_addrAck cycle or in any cycle following the assertion of SI_addrAck and must not assert SI_rDAck anytime during this transfer since no data is transferred across the PLB.

The DMA controller will then request a PLB slave buffered memory to memory write transfer. This causes the SBC to write the contents of the memory buffer out to the address generated by the DMA during the PLB write operation. The SBC must then assert SI_wrComp either with its SI_addrAck or in any cycle following the assertion of SI_addrAck

and must not assert SI_wrDAck anytime during this transfer since no data is transferred across the PLB.

2.5.5.5 PLB Slave Buffered Peripheral To/From Memory Transfers (Mn_type = 0b101)

PLB slave buffered peripheral transfers are used by a DMA master to request that data be read into or written from a PLB slave buffer. In this transfer the DMA controller performs a memory to peripheral or a peripheral to memory transfer with both source and destination on the same slave bus. These transfers are unique in that only one PLB transfer occurs for both the read and write portions of the transfer. The PLB portion of the transfer corresponds to the memory portion of the transfer. The peripheral portion of the transfer occurs through the DMA sideband signals.

A buffered memory to peripheral transfer is started by the DMA requesting a PLB slave buffered peripheral to/from memory transfer type. The PLB_RNW is asserted indicating a memory read will occur and the memory address is placed on PLB_ABus(0:31).

Note: The PLB_busLock is not required to be asserted for these transfers since both the memory read and the peripheral write transfer are requested in a single PLB request. Once the SBC decodes and acknowledges the address by asserting SI_addrAck, the SBC must assert the SI_rdComp anytime after the SI_addrAck cycle and must not assert the SI_rdDAck signal anytime during the transfer since no data is transferred across the PLB.

Following the SI_addrAck cycle, the DMA controller begins sampling the SI_rdDataXfer while the SBC performs the memory read and loads the data into its buffer. The SBC then asserts the SI_rdDataXfer in the last cycle of the memory read, including hold time, to indicate to the DMA controller that the memory transfer is now complete and that the peripheral write transfer may start in the clock cycle following the assertion of SI_rdDataXfer. The DMA controller then begins the peripheral write transfer and the SBC buffer contents are driven onto the slave data bus by the SBC. DMA_extAck is asserted for the programmed number of clock cycles and the SBC must continue to drive the data bus until the DMA_endOp is asserted by the DMA controller. This completes the peripheral write transfer to the peripheral.

A buffered peripheral to memory transfer is started by the DMA requesting a PLB slave buffered peripheral to/from memory transfer type. The PLB_RNW is negated indicating a memory write will occur and the memory address is placed on PLB_ABus(0:31).

Note: The PLB_busLock is not required to be asserted for these transfers since both the peripheral read and the memory write transfers are requested in a single PLB request.

Once the PLB slave decodes and acknowledges this transfer by asserting SI_addrAck, the SBC must assert the SI_wrComp signal either with the SI_addrAck or in any clock cycle following the SI_addrAck and must not assert the SI_wrDAck signal anytime during the transfer since no data is transferred across the PLB. In the clock cycle following the assertion of the SI_addrAck, the DMA controller asserts DMA_extAck for the

preprogrammed number of clock cycles and then asserts DMA_dataXfer during the last active clock cycle of DMA_extAck. This indicates to the SBC to latch the data off the slave data bus and load it into its buffer. Upon assertion of the DMA_endOp signal, the SBC then performs the memory write transfer writing the contents of the buffer out to memory.

2.5.5.6 DMA Buffered Memory Transfers (Mn_type = 0b110)

DMA buffered memory transfers are used by a DMA master to request that data be read into or written out of the DMA. The DMA contains a 16-byte line buffer which it uses to temporarily store the byte, halfword, word, or line data being transferred. The PLB address and control signals appear the same as if any other PLB master was performing a memory transfer. This transfer differs from the regular memory transfer type, Mn_type = 0b000, because the DMA sideband signals SI_rdDataXfer and SI_wrDataXfer are valid during certain DMA memory transfers.

For a read transfer the only difference between regular memory transfers and DMA transfers is during DMA line reads. The DMA has the capability to optionally overlap line read and write transfers on the PLB. This is accomplished by the PLB SBC driving SI_rdDataXfer asserted one clock prior to the assertion of the final SI_rdDack. This effectively allows the DMA to start the line write operation in the clock that the last word of data is being read by the DMA.

Note: If the PLB SBC does not drive the SI_rdDataXfer asserted during the line read the DMA will simply wait until the final SI_rdDack of the line has been received and then begin the line write.

For a write transfer the only difference between regular memory transfers and DMA transfers is that the PLB memory slave must drive SI_wrDataXfer asserted in the clock cycle that the actual write on the slave memory bus occurred. This is used by the DMA to sample the peripheral request signal in device paced memory to memory transfers and is not valid until the write to the peripheral has occurred.

2.5.5.7 PLB Slave Buffered Memory to Memory Transfers With Sideband Signals (Mn_type = 0b111)

DMA buffered memory to memory transfers are used by a DMA master to request that data be read into or written from a PLB SBC's buffer. In this transfer the DMA performs memory to memory operations with both memory source and memory destination residing on the same slave bus. The use of the slave buffer increases performance because the transfer of data is limited to the slave bus and does not need to be transferred across the PLB bus and through the DMA.

This transfer operates in the same manner as the PLB slave buffered memory to memory transfer and in addition, the DMA_rdDataXfer and DMA_wrDataXfer signals are asserted by the PLB slave. These sideband signals are used by the DMA controller to determine when to sample the peripherals request signal during device paced memory to memory transfers. The PLB slave should assert the SI_rdDataXfer during the final clock cycle of a read transfer and assert the SI_wrDataXfer during the final clock cycle of a write transfer.

2.5.6 PLB/DMA Example Timing Diagrams

The following DMA transfer examples are based on the bus controller and DMA controller functionality used in Core+ASIC design environment. The PLB based DMA transfer operations with timing diagrams describe how the PLB may be used to perform third party DMA operations.

2.5.6.1 DMA Flyby Memory Read Transfer

Figure 2-35 shows DMA flyby memory read transfer. The transfer begins by the DMA slave peripheral asserting EXT_dmaReq. The DMA controller responds by requesting a PLB read transfer. The PLB SBC decodes the address and transfer type on the PLB and acknowledges the request by asserting SI_addrAck. Next, the DMA asserts DMA_extAck and the PLB SBC begins a memory read from the requested address. Once the slave bus memory read is complete the PLB SBC asserts SI_rdDataXfer and the DMA negates DMA_extAck. The PLB SBC completes the PLB bus transfer by asserting SI_rdComp. If the DMA slave peripheral request is asserted at either DMA sample point, another transfer is performed.

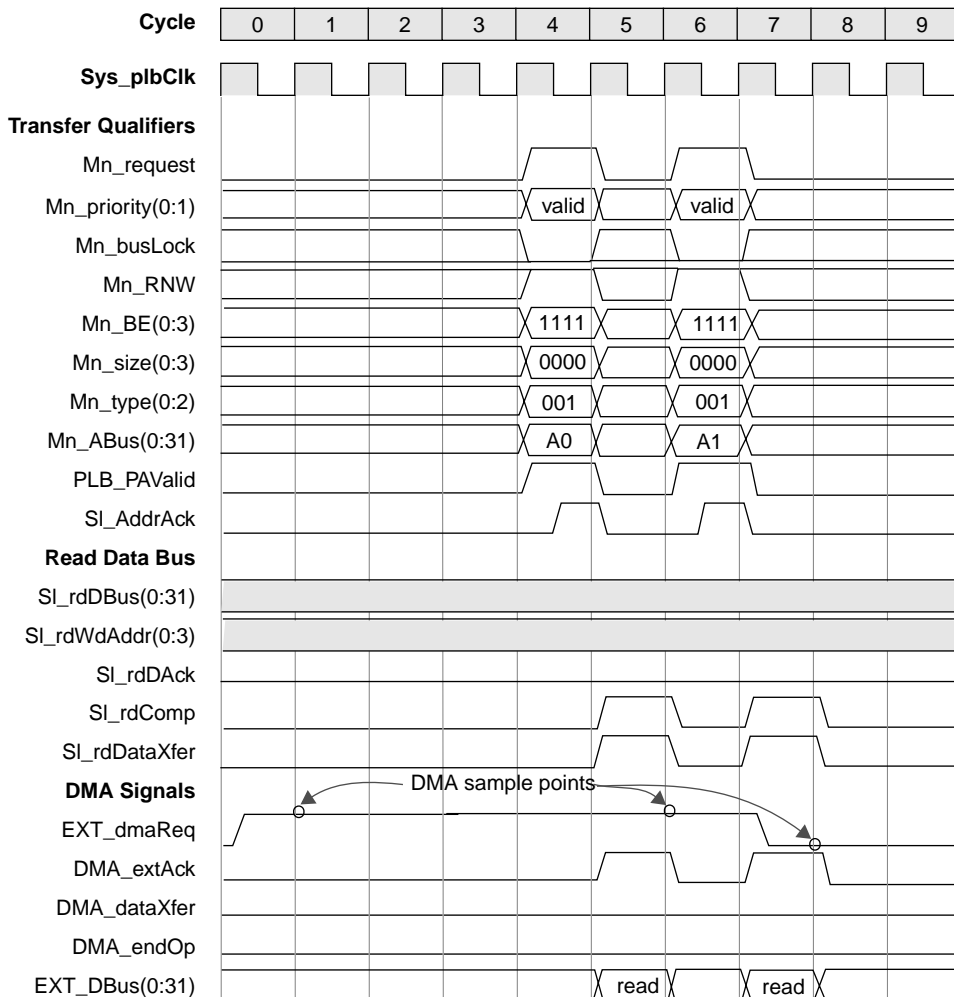


Figure 2-35. DMA Flyby Memory Read Transfer

2.5.6.2 DMA Flyby Memory Write Transfer

Figure 2-36 shows DMA flyby memory write transfer. The transfer begins by the DMA slave peripheral asserting the EXT_dmaReq. The DMA controller responds by requesting a PLB write transfer. The PLB SBC decodes the address and transfer type on the PLB and acknowledges the request by asserting SI_addrAck and SI_wrComp. Next, the DMA asserts DMA_extAck and the PLB SBC simultaneously begins a memory write to the requested address. Once the slave bus memory write is complete the PLB SBC asserts SI_wrDataXfer and the DMA negates DMA_extAck. If the DMA slave peripheral request is asserted at either DMA sample point, another transfer is performed.

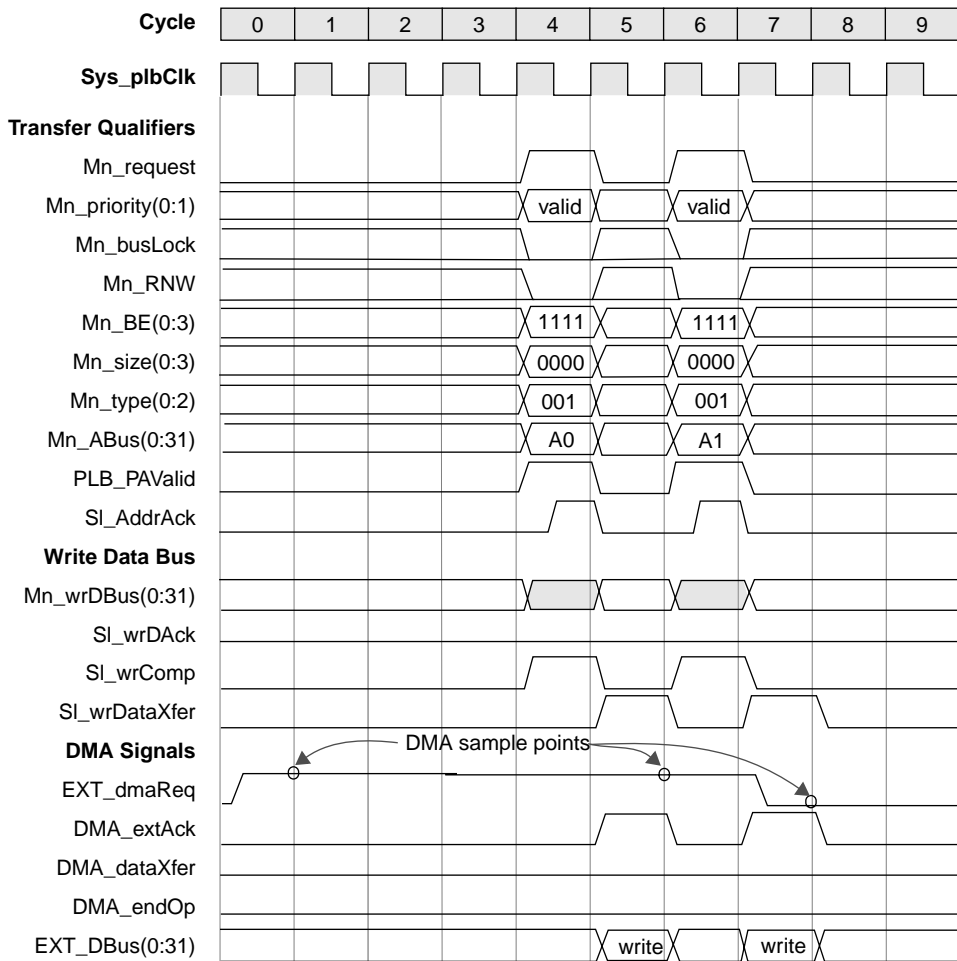


Figure 2-36. DMA Flyby Memory Write Transfer

2.5.6.3 DMA Flyby Burst Memory Read Transfer

Figure 2-37 shows DMA flyby burst memory read transfer. The transfer begins by the DMA slave peripheral asserting the EXT_dmaReq signal. The DMA controller responds by requesting a PLB read transfer. The PLB slave decodes the address and transfer type on the PLB and acknowledges the request by asserting SI_addrAck. Next, the DMA asserts DMA_extAck and the PLB SBC begins a memory read from the requested address. Once the slave bus memory read is complete the PLB SBC asserts SI_rdDataXfer and samples the DMA_flyByBurst signal to determine whether or not to continue the burst transfer. The burst transfer continues until the PLB slave samples the DMA_flyByBurst signal negated and DMA negates DMA_extAck. Note that the PLB_rdComp signal is asserted in the clock cycle following the address acknowledge freeing up the PLB bus prior to the completion of the FlyBy transfer.

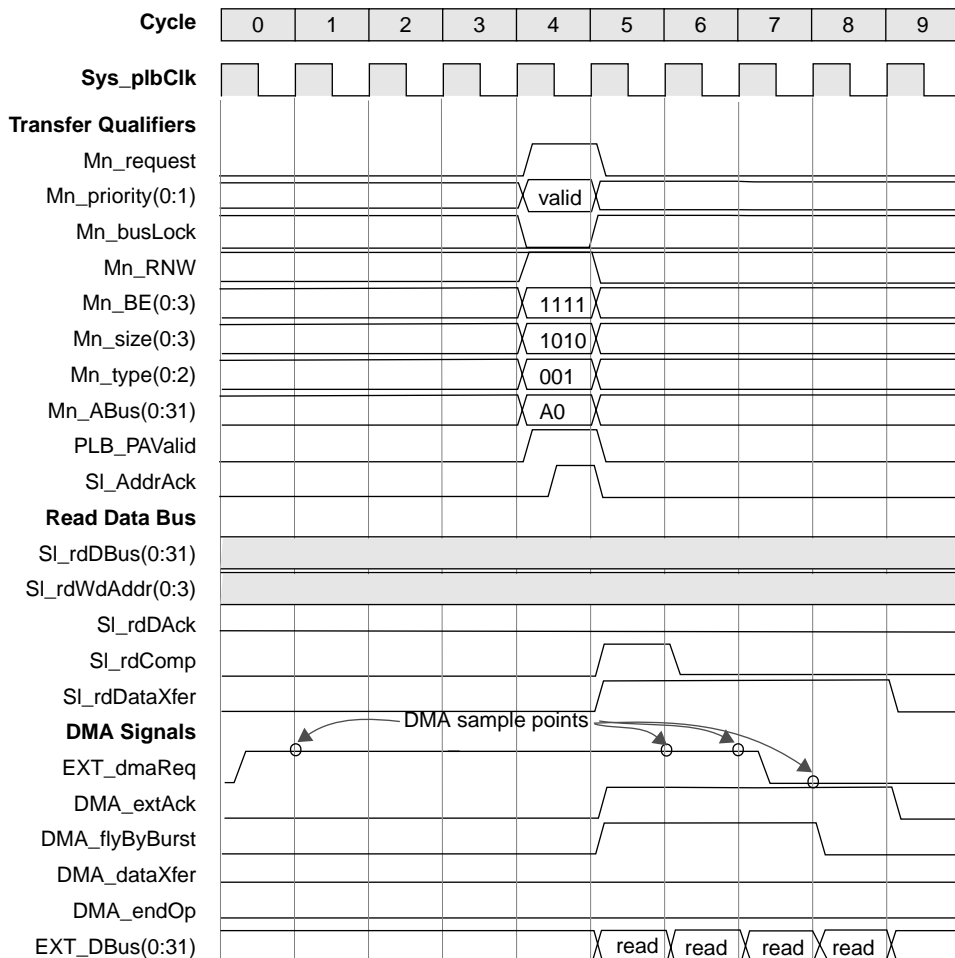


Figure 2-37. DMA Flyby Burst Memory Read Transfer

2.5.6.4 DMA Buffered Memory to Peripheral Transfer

Figure 2-38 shows DMA buffered memory to peripheral transfer. The transfer begins by the DMA slave peripheral asserting the EXT_dmaReq. The DMA responds by requesting a PLB read transfer. The PLB SBC acknowledges the request by asserting SI_addrAck. Memory is read and the PLB SBC asserts SI_rdComp and provides the read data with SI_rdDack. The DMA buffers this data internally and requests a PLB peripheral write transfer. The PLB SBC acknowledges the request by asserting SI_addrAck, SI_wrDack and SI_wrComp. Next, the DMA asserts DMA_extAck, and the PLB SBC responds by driving the latched write data onto the external bus. The DMA controller then negates the DMA_extAck signal and asserts DMA_endOp to indicate that the transfer has been completed and the PLB SBC must stop driving data.

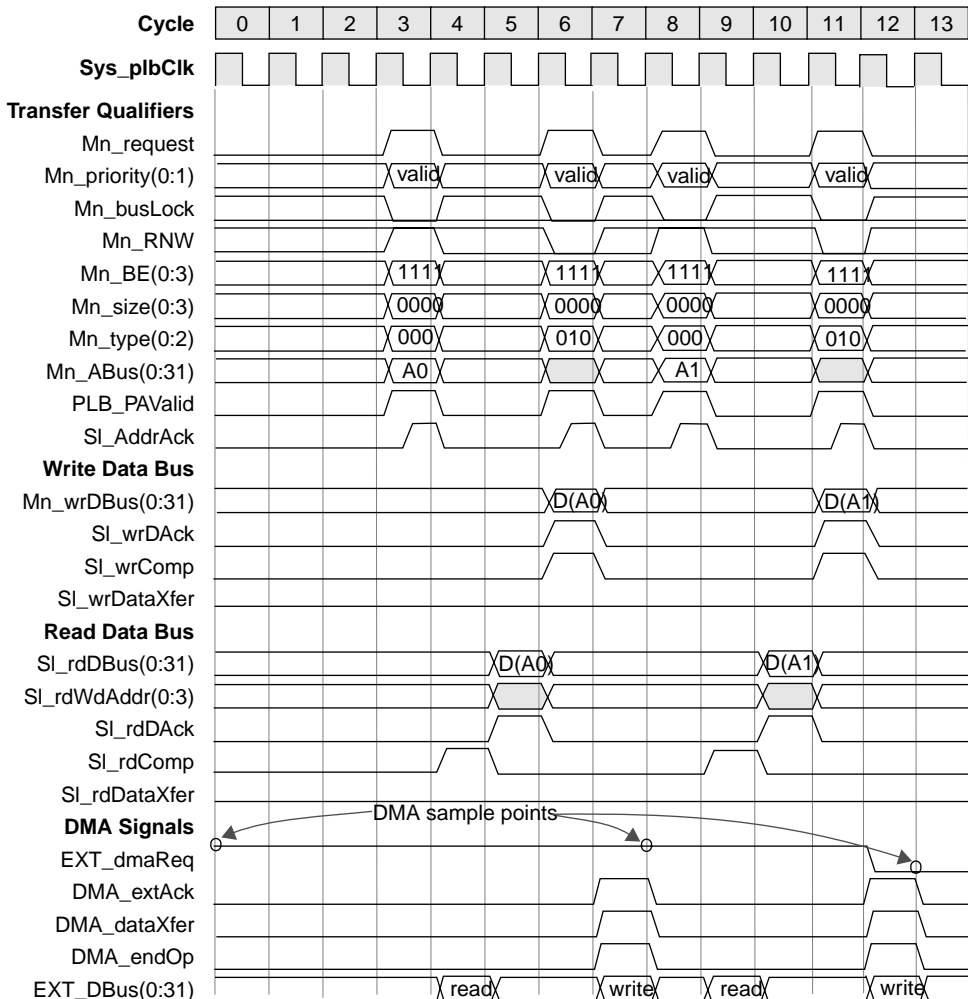


Figure 2-38. DMA Buffered Memory to Peripheral Transfer

2.5.6.5 DMA Buffered Peripheral to Memory Transfer

Figure 2-39 shows DMA buffered peripheral to memory transfer. The transfer begins by the DMA slave peripheral asserting the EXT_dmaReq. The DMA responds by requesting a PLB read transfer. The PLB SBC acknowledges the request by asserting SI_addrAck. Next, the DMA asserts DMA_extAck and the peripheral drives data onto the external bus. The DMA then asserts the DMA_dataXfer to indicate to the PLB SBC to latch the external data, and the DMA_endOp signal to complete the PLB transfer. The PLB SBC then asserts SI_rdComp and provides the data with SI_rdDack. The DMA buffers this data internally and requests a PLB write transfer. The PLB SBC acknowledges the request by asserting SI_addrAck, SI_wrDack, and SI_wrComp. If the DMA slave peripheral request was asserted at the falling edge of DMA_extAck another transfer is performed.

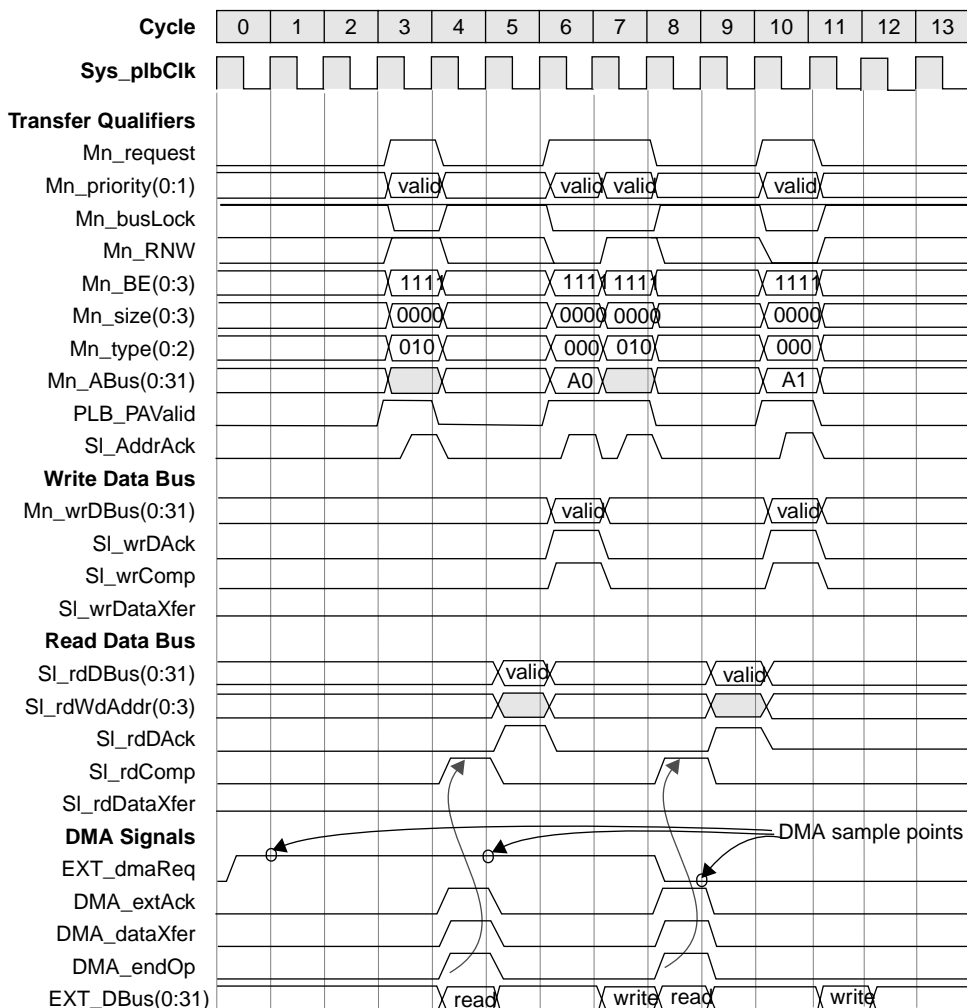


Figure 2-39. DMA Buffered Peripheral to Memory Transfer

2.5.6.6 DMA PLB Slave Buffered Memory To Memory Transfer

Figure 2-40 shows DMA PLB slave buffered memory to memory transfer. The DMA controller begins the transfer by requesting a locked PLB read transfer. The PLB SBC decodes the address and transfer type on the PLB and acknowledges the request by asserting `SI_addrAck`. In response to the assertion of `SI_addrAck`, the DMA controller begins the last portion of the memory to memory operation by requesting a PLB write transfer. The PLB SBC performs the memory read from the requested address and stores the data in its local buffer and completes the PLB transfers by asserting `SI_rdComp`, and `SI_wrComp`, as shown. The PLB SBC then writes the contents of its local buffer out to memory to complete the memory to memory operation.

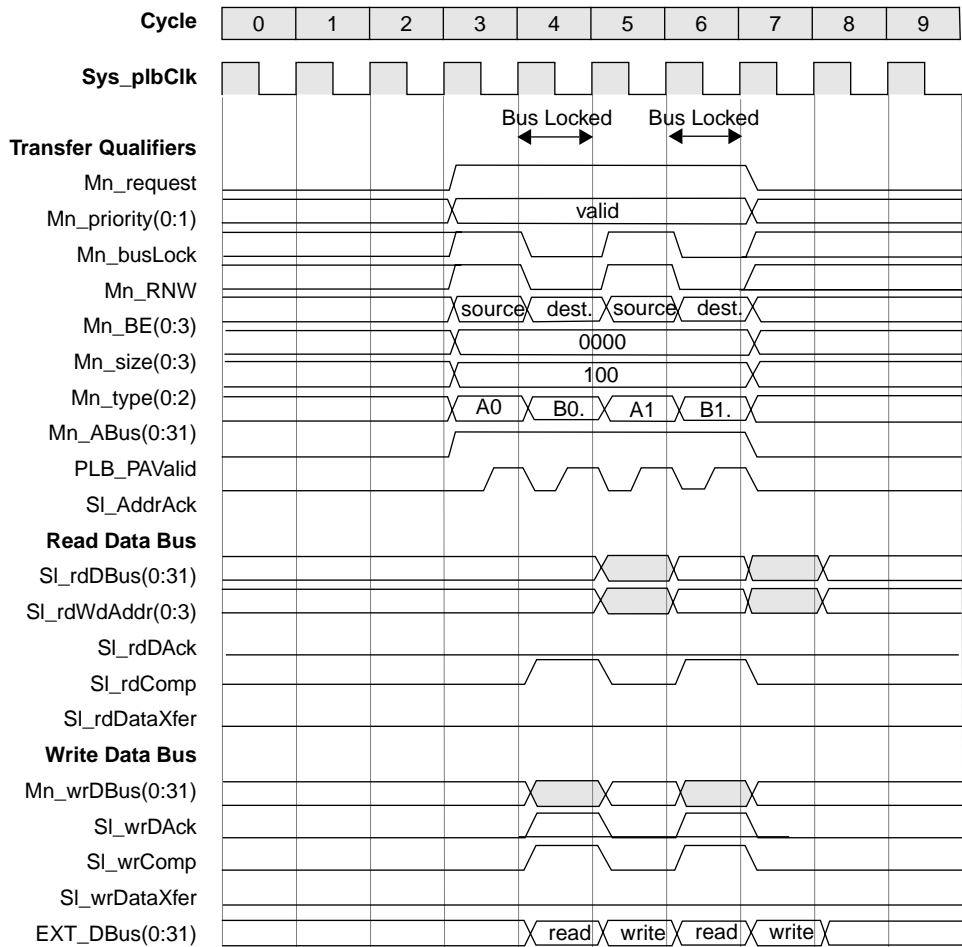


Figure 2-40. DMA PLB Slave Buffered Memory to Memory Transfer

2.5.6.7 DMA PLB Slave Buffered Peripheral To Memory Transfer

Figure 2-41 shows DMA PLB slave buffered peripheral to memory transfer. The transfer begins by the DMA slave peripheral asserting the EXT_dmaReq signal. The DMA controller responds by requesting a PLB buffered peripheral write transfer. The PLB SBC decodes the address and transfer type on the PLB and acknowledges the DMA request by asserting SI_addrAck and SI_wrComp. Next, the DMA controller asserts the DMA_extAck signal and the peripheral provides data. The DMA then asserts, the DMA_dataXfer signal to indicate to the PLB SBC to latch the data into its local buffer, and the DMA_endOp signal to indicate the last clock cycle of data hold time. The DMA controller negates DMA_extAck and samples the peripheral request to determine if another transfer is requested. The PLB SBC then performs a memory write of the data.

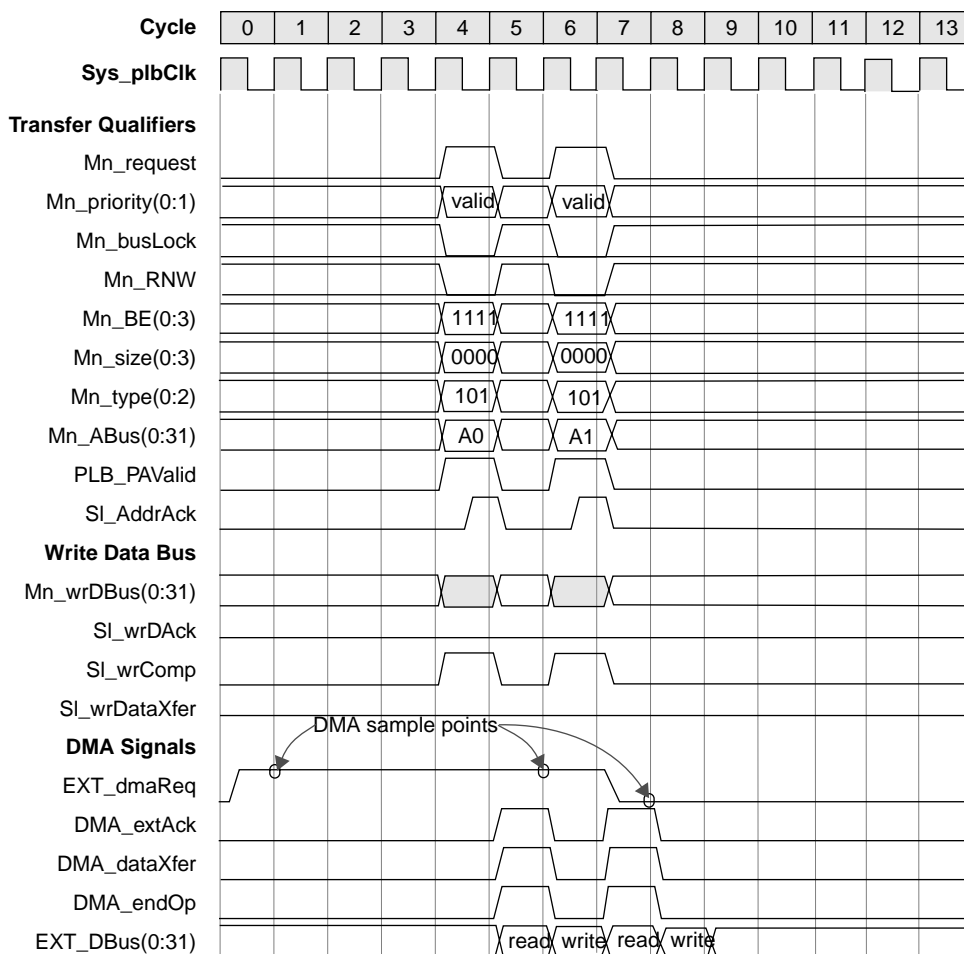


Figure 2-41. DMA PLB Slave Buffered Peripheral to Memory Transfer

2.5.6.8 DMA PLB Slave Buffered Memory To Peripheral Transfer

Figure 2-42 shows DMA buffered memory to peripheral transfer. The transfer begins by the DMA slave peripheral asserting the EXT_dmaReq signal. The DMA controller responds by requesting a PLB read transfer. The PLB SBC acknowledges the request by asserting SI_addrAck. The PLB SBC then performs a memory read from the requested address and stores the data in its local buffer. The PLB SBC completes the PLB operation by asserting SI_rdComp (and optionally SI_rdDack). The PBL SBC also asserts SI_rdDataXfer to indicate to the DMA that the data stored in its local buffer is being driven on the slave bus. The DMA asserts DMA_extAck in response. The DMA then negates DMA_extAck and samples the peripheral request to see if another transfer is requested.

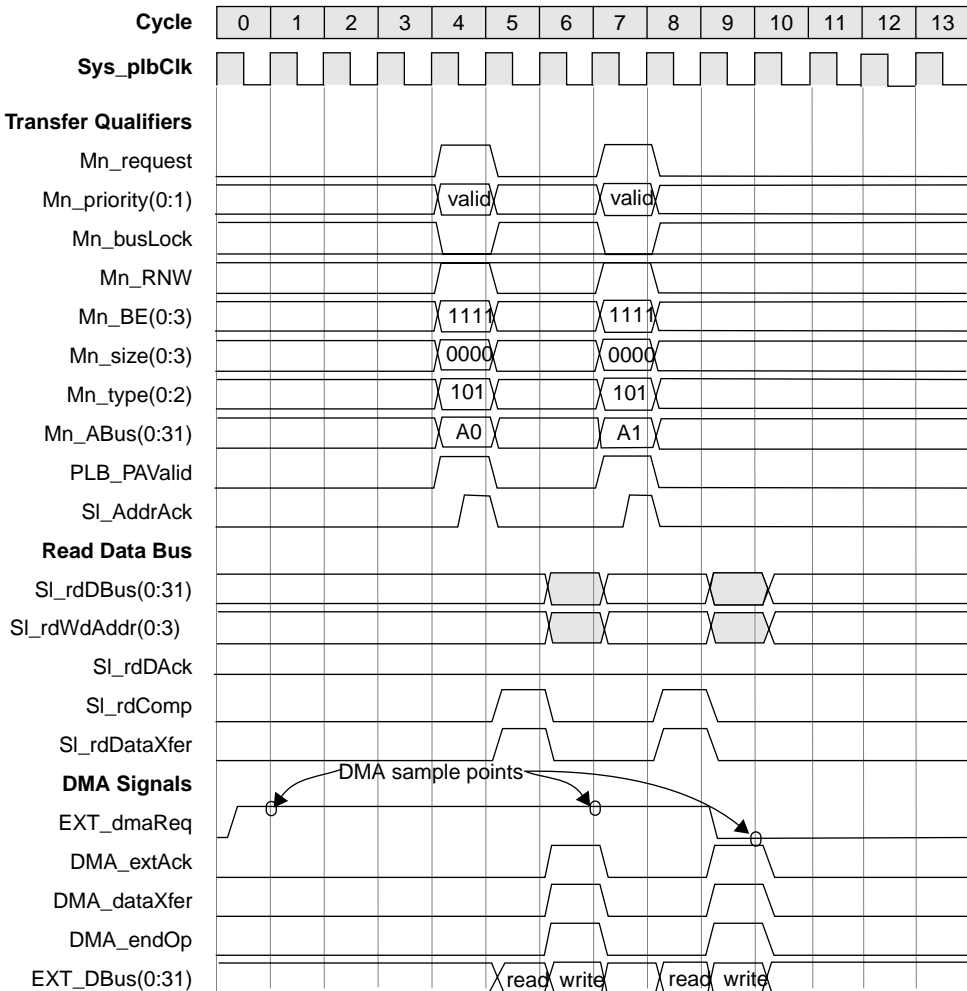


Figure 2-42. DMA PLB Slave Buffered Memory To Peripheral Transfer

Chapter 3. On-Chip Peripheral Bus

The on-chip peripheral bus (OPB) is designed for easy connection of on-chip peripheral devices. It provides a common design point for various on-chip peripherals. The OPB is a fully synchronous bus which functions independently at a separate level of bus hierarchy. It is not intended to connect directly to the processor core. The processor core can access the peripheral on this bus through the OPB bridge unit which is a separate macro. See the OPB bridge user's manual for more information. This document is organized as follows:

- OPB Overview
- OPB Signals
- OPB Interfaces
- OPB Operations

3.1 OPB Overview

The on-chip peripheral bus has the following features:

- A 32 bit address bus and a 32 bit data bus
- Fully synchronous, edge triggered protocol
- Provides support for 8-bit, 16-bit and 32-bit peripherals
- Uses a distributed multiplexer method of attachment instead of tristate drivers, to ease manufacturing test. Address and data buses may be implemented in distributed AND-OR gates or as a dotted bus
- Dynamic bus sizing; byte, halfword, and fullword transfers
- Byte and halfword duplication for byte and halfword transfers
- Single cycle transfer of data between OPB bus master and OPB slaves
- Sequential address (Burst) protocol support
- Devices on the OPB may be memory mapped, act as DMA peripherals, or support both transfer methods
- A 16-cycle fixed bus timeout provided by the OPB arbiter
- OPB slave is capable of disabling the fixed timeout counter to suspend bus timeout error
- Support for multiple OPB bus masters
- Bus parking for reduced latency
- OPB masters may lock the OPB bus arbitration
- OPB slaves capable of requesting retry to break possible arbitration deadlock
- Bus arbitration overlapped with last cycle of bus transfers

- Flexible architecture extendable to support 64-bit transfers

3.1.1 Physical Implementation

Figure 3-1 shows a physical implementation of the OPB. Since the OPB supports multiple master devices, the address bus and data bus are implemented as a distributed multiplexer. This design will enable future peripherals to be added to the chip without changing the I/O on either the OPB arbiter or the other existing peripherals. By specifying the bus qualifiers as I/O for each peripheral (select for the ABus, DBusEn for the DBus), the bus can be implemented in a variety of ways, that is, as a distributed ring mux (shown below), using centralized AND/OR's or multiplexers, using transceiver modules and a "dotted" bus, etc. The optimal design for each implementation will vary, depending on the number of devices attached to the OPB and timing and routing constraints.

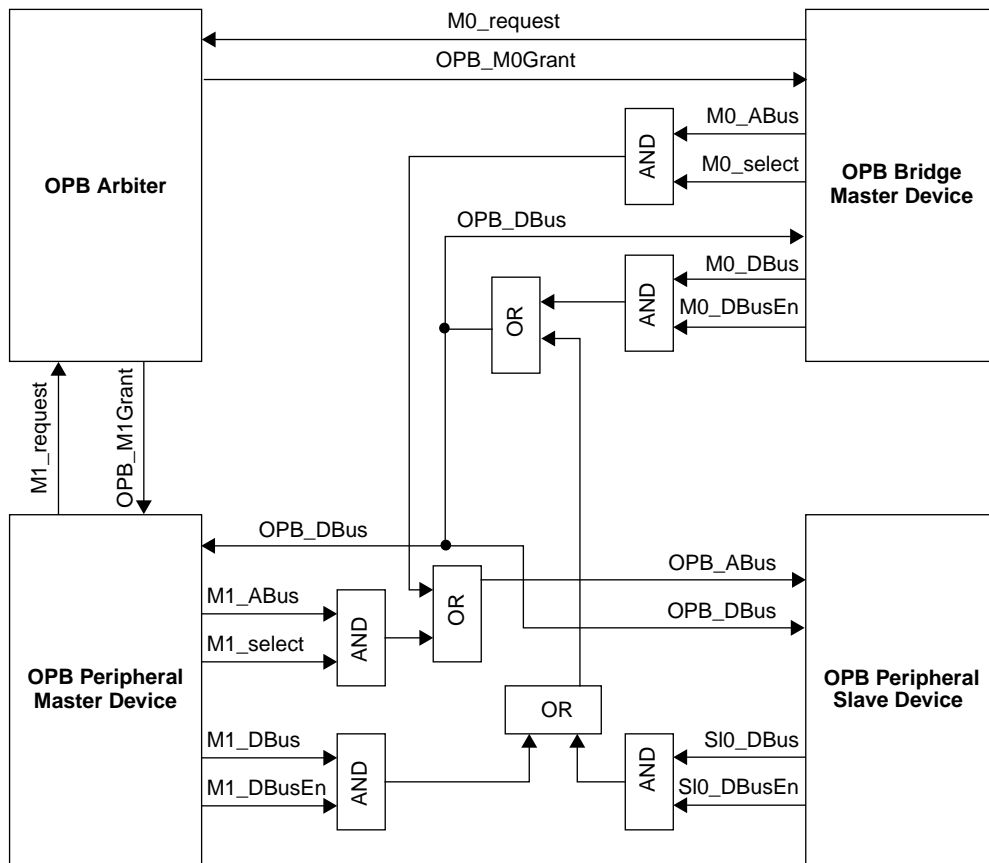


Figure 3-1. Physical Implementation of the OPB Bus

Control signals from OPB masters and slaves to and from the OPB arbiter and the peripherals will be similarly OR'ed together, and then sent to each device. Bus arbitration signals such as Mn_request and OPB_MnGrant are directly connected between the OPB arbiter and each OPB master device.

The following three tables show OPB output signal connection. These tables show the OPB bus logic used to combine signals from each master and slave device, to distribute to other master and slave devices.

Table 3-1 describes master output connection.

Table 3-1. Master Output Connection

Signal	To	Bus Logic	Gated By (AND)
Mn_request	Mn_request	Direct	Ungated
Mn_busLock	OPB_busLock	OR	Ungated
Mn_select	OPB_select	OR	Ungated
Mn_RNW	OPB_RNW	AND-OR	Mn_select
Mn_hwXfer	OPB_hwXfer	AND-OR	Mn_select
Mn_fwXfer	OPB_fwXfer	AND-OR	Mn_select
Mn_seqAddr	OPB_seqAddr	AND-OR	Mn_select
Mn_ABus	OPB_ABus	AND-OR	Mn_select
Mn_DBus	OPB_DBus	AND-OR	Mn_DBusEn
Mn_DBusEn	Qualifies DBus		Ungated
SIn_DMAReq	SIn_DMAReq	Direct	Ungated

Table 3-2 describes slave output connection.

Table 3-2. Slave Output Connection

Signal	To	Bus Logic	Gated By (AND)
SIn_xferAck	OPB_xferAck	OR	Ungated
SIn_hwAck	OPB_hwAck4	OR	Ungated
SIn_fwAck	OPB_fwAck	OR	Ungated
SIn_errAck	OPB_errAck	OR	Ungated
SIn_retry	OPB_retry	OR	Ungated
SIn_ToutSup	OPB_ToutSup	OR	Ungated
SIn_DBus	OPB_DBus	AND-OR	SIn_DBusEn
SIn_DBusEn	Qualifies DBus	N/A	Ungated
SIn_DMAREq	SIn_DMAREq	Direct	Ungated

Table 3-3 describes arbiter output connection.

Table 3-3. Arbiter Output Connection

Signal	To	Bus Logic
OPB_MnGrant	OPB_MnGrant	Direct
OPB_timeout	OPB_timeout	Direct

3.2 OPB Signals

OPB signals can be grouped under the following categories:

- Arbitration Signals
- Bus Signals
- Data Transfer Control Signals
- Optional DMA Peripheral Support Signals

3.2.1 Signal Naming Conventions

The implementation of the OPB consists of an OPB arbiter, and OPB master and slave devices, connected at the chip level by bus logic (AND and OR gates). Slaves which are connected to the OPB use the following naming convention:

- Signals which are outputs of the OPB bus logic and inputs to the master and slave devices are prefixed with `OPB_`. There is only one output of the bus logic for each one of these signals and it is received as an input by each relevant device attached on the OPB. For example, `OPB_RNW` is an output of the OPB bus logic and is an input to each slave attached to the OPB.
- Signals which are outputs of the OPB slaves and inputs to the OPB bus logic are prefixed with `Sln_`. Each slave has its own output which is an input to the OPB bus logic, where it is logically OR'ed together to form a single signal. The slaves must ensure that these signals are driven to a logic '0' when they are not involved in a transfer on the OPB (with the exception of `Sln_DBus`, which need not be driven to a "0" when inactive, but which is qualified with `Sln_DBusEn`). For example, `Sln_xferAck` is an output of each slave attached to the OPB, there are multiple `Sln_xferAck` inputs to the OPB bus logic, and they are OR'ed together to form `OPB_xferAck`, which is output to all OPB masters.

Each master is attached directly to the OPB bus logic with its own address, read data, and write data buses and control signals, and these signals use the following naming convention:

- Signals which are driven by a master as an input to the OPB bus logic are pre-fixed with `Mn_`. For example, `Mn_request` signal when implemented would result in `M0_request`, `M1_request`, etc., up to the maximum number of masters supported (implementation dependent).
- Signals which are driven by the OPB arbiter to a specific master have a prefix `OPB_Mn` to indicate that this signal is from the OPB arbiter to a specific master (i.e. `OPB_MnGrant`). The OPB arbiter provides an output for this signal for each master attached to the bus. For example, the `OPB_MnGrant` signal, when implemented would result in `OPB_M0Grant`, `OPB_M1Grant`, etc., up to the maximum number of masters supported (implementation dependant).

- Signals which are driven by the OPB bus logic to all master and slave devices are prefixed with OPB_. There is only one output of the OPB bus logic for each one of these signals and it is received as an input by each relevant device attached on the OPB. For example, OPB_select is an output of the OPB bus logic and is an input to each slave device attached to the OPB.

Table 3-4 provides a summary of all OPB input/output signals in alphabetical order, the interfaces under which they are grouped, followed by a brief description and page reference for detailed functional description.

Table 3-4. Summary of OPB Signals

Signal Name	Interface	I/O	Description	Page
DMA_SInAck	DMA	I	DMA slave acknowledge	3-13
Mn_request	Master	O	Master bus request	3-8
Mn_busLock	M/A	O	Master bus arbitration lock	3-8
Mn_select	Master	O	Master select	3-10
Mn_RNW	Master	O	Master read not write	3-11
Mn_hwXfer	Master	O	Master halfword transfer	3-11
Mn_fwXfer	Master	O	Master fullword transfer	3-11
Mn_seqAddr	Master	O	Master sequential address	3-11
Mn_ABus(0:31)	Master	I	Master address bus	3-10
Mn_DBus(0:31)	Master	O	Master data bus	3-12
Mn_DBusEn	Master	O	Master data bus enable	3-12
OPB_busLock	Arbiter	I	OPB bus arbitration lock	3-8
OPB_MnGrant	M/A	I	OPB master bus grant	3-8
OPB_timeout	M/A	I	OPB timeout error	3-9
OPB_select	S/A	I	OPB select	3-10
OPB_RNW	Slave	I	OPB read not write	3-11
OPB_hwXfer	Slave	I	OPB halfword transfer	3-11
OPB_fwXfer	Slave	I	OPB fullword transfer	3-11
OPB_seqAddr	Slave	I	OPB sequential address	3-11
OPB_ABus(0:31)	M/S	I	OPB address bus	3-10
OPB_DBus(0:31)	M/S	I	OPB data bus	3-12

Table 3-4. Summary of OPB Signals (cont.)

Signal Name	Interface	I/O	Description	Page
OPB_xferAck	M/A	I	OPB transfer acknowledge	3-12
OPB_hwAck	Master	I	OPB halfword acknowledge	3-12
OPB_fwAck	Master	I	OPB fullword acknowledge	3-12
OPB_errAck	Master	I	OPB error acknowledge	3-13
OPB_retry	Master	I	OPB bus cycle retry	3-9
OPB_ToutSup	Arbiter	I	OPB timeout suppress	3-13
SIn_DMAREq	DMA	O	Slave DMA request	3-13
SIn_xferAck	Slave	O	Slave transfer acknowledge	3-12
SIn_hwAck	Slave	O	Slave halfword acknowledge	3-12
SIn_fwAck	Slave	O	Slave fullword acknowledge	3-12
SIn_errAck	Slave	O	Slave error acknowledge	3-13
SIn_retry	Slave	O	Slave bus cycle retry	3-9
SIn_ToutSup	Slave	O	Slave timeout suppress	3-13
SIn_DBus(0:31)	Slave	O	Slave data bus	3-10
SIn_DBusEn	Slave	O	Slave data bus enable	3-12

3.2.2 Arbitration Signals

OPB bus arbitration among requesting master devices is performed by the OPB arbiter. Each master connects directly to the arbiter through Mn_request and OPB_MnGrant signals. The arbiter also receives OPB_busLock, OPB_select, and OPB_xferAck to monitor bus activity for arbitration. The OPB arbiter defines two valid types of arbitration cycles:

1. Idle

OPB_select and OPB_busLock are deasserted, indicating no data transfer in progress.

2. Overlapped arbitration cycle

OPB_XferAck is asserted, indicating the final cycle in a data transfer, and OPB_busLock is not asserted. Arbitration in this cycle allows another master to begin a transfer in the following cycle, avoiding the need for a “dead” cycle on the bus.

Grants are issued to masters (requesting or parked) only during valid arbitration cycles. Locking masters receive grants during valid arbitration cycles in response to requests, while the bus is locked.

3.2.2.1 Mn_request (Master Bus Request)

The Mn_request signal is asserted by an OPB master to request control of the bus, which is granted by the OPB arbiter via the OPB_MnGrant signal. For single transfers, Mn_request should normally be deasserted during the first or only cycle during which the bus is to be used by the device. If an OPB master requires continuous data transfer cycles, it can continue to assert Mn_request, and then deassert Mn_request during the first or only cycle of the last data transfer. Continuous assertion of Mn_request does not assure uninterrupted access to the OPB; the bus lock signal is provided to accomplish this.

3.2.2.2 OPB_busLock, Mn_busLock(OPB Bus Arbitration Lock)

The Mn_busLock signal may be asserted by an OPB master during any data transfer cycle for which it controls the bus (Mn_select is asserted). Mn_busLock may be asserted simultaneously with Mn_select, or in any following cycle in which Mn_select is asserted. The busLock signal may be deasserted at any time, and will normally be deasserted during the final data transfer cycle of a master's sequence of transfers, to allow for overlapped bus arbitration.

The OPB is "locked" whenever OPB_busLock is active, and will arbitrate among requesting masters during valid arbitration cycles only when OPB_busLock is inactive. While locked, the OPB Arbiter will continue to grant the bus to the OPB master device asserting lock, without regard to the priority of the device or other devices with concurrent requests for the bus.

The bus master which asserts Mn_busLock may proceed to the next data transfer cycle without bus arbitration. Lock has the effect of freezing the arbiter in its current state, i.e., granted to the locking master. Thus, Mn_request and OPB_MnGrant have no effect on bus arbitration while the bus is locked -- although a master will typically continue to assert Mn_request, and OPB_MnGrant will continue to be asserted during valid arbitration cycles. The master could deassert Mn_select, relinquishing control of the bus, but would still be "granted" the bus by the arbiter as long as Mn_busLock remains asserted. Deassertion of Mn_busLock results in bus arbitration during the next valid bus arbitration cycle, which may be the same cycle in which Mn_busLock is deasserted. By deasserting Mn_busLock prior to the final data transfer cycle, the asserting master potentially allows another master to proceed with a separate data transfer cycle with no intervening "dead" cycle on the OPB. See the timing diagrams for examples.

3.2.2.3 OPB_MnGrant (OPB Master Bus Grant)

The OPB_MnGrant signal is asserted by the OPB Arbiter to grant control of the bus to a master device requesting it. The master may begin to drive signals on the OPB in the cycle following the assertion of OPB_MnGrant. All OPB masters should examine their bus grant signals at the rise of the OPB Clock, and may not proceed to initiate a bus cycle unless it is asserted (or unless they have locked the bus). The OPB_MnGrant signal will only be asserted during a valid arbitration cycle, as defined above.

If a master has locked the bus via OPB_busLock, that master retains control of the OPB and no other master requests will be granted. The locking master's grant signal will be asserted in response to its request during valid arbitration cycles, but the assertion of request and grant have no effect on bus arbitration for the duration of the bus lock condition.

The OPB supports bus parking. If no requests are pending during a valid OPB arbitration cycle, the OPB Arbiter may “park” on one master, asserting that master's grant signal. This allows the parked master to proceed with a data transfer cycle without incurring the delay of performing an arbitration cycle, if no other master is requesting. A parked master may proceed to initiate a bus transfer cycle by asserting Mn_select, if its grant signal was asserted in the previous cycle, without the delay of asserting request and awaiting a valid grant.

The enabling of the bus parking feature, and determination of which master the OPB Arbiter will park on, are implementation dependant.

3.2.2.4 OPB_timeout (OPB Timeout Error)

The OPB_timeout signal is an output of the OPB Arbiter. OPB_timeout is an input to all master devices on the OPB, and is used to indicate that a timeout error has occurred. This signal will be asserted in the 16th cycle following the assertion of OPB_select if there is no response from a slave (OPB_xferAck or OPB_retry), and if ToutSup is not asserted by an addressed slave device to suppress the timeout. Upon assertion of OPB_timeout, the master device which initiated the transfer cycle must terminate the transfer by deasserting Mn_select signal in the cycle following the assertion of OPB_timeout. If OPB_busLock is not asserted, the OPB Arbiter will perform a bus arbitration in the cycle in which OPB_select is deasserted. If OPB_busLock is asserted, the requesting master retains control of the OPB, but must still deassert Mn_select following the assertion of OPB_timeout for at least one cycle.

If OPB_xferAck or OPB_retry are asserted in the 16th cycle following select, coincident to the assertion of OPB_timeout, the master device should ignore OPB_timeout, and respond to the slave's OPB_xferAck or OPB_retry signal.

3.2.2.5 OPB_retry, Sln_retry(OPB Bus Cycle Retry)

The Bus Cycle Retry signal is asserted by an OPB slave to indicate that it is unable to perform the requested transfer at this time. The primary use of this signal is to permit resolution of a deadlock condition which may occur as a result of system implementations that include buses which operate independently.

An OPB slave will assert the Sln_retry signal instead of the Sln_xferAck signal when a situation requiring it is detected. It must remain asserted until the slave becomes deselected as a result of the OPB_select signal being deasserted. Sln_retry will cause the requesting master to terminate the transfer by deasserting OPB_select, and remove its request from the Arbiter. Both the master's Mn_select and Mn_request must remain deasserted for one cycle, during which the OPB Arbiter will re-arbitrate the bus.

A slave asserting Sln_retry must not assert Sln_xferAck.

Sln_retry must be asserted within 16 cycles of OPB_select to avoid a timeout, unless Sln_ToutSup is asserted. If OPB_retry and OPB_timeout are received simultaneously by a master device (i.e., Sln_retry was asserted in the 16th cycle following OPB_select), the master should ignore OPB_timeout and act on OPB_retry as appropriate

3.2.3 Bus Signals

3.2.3.1 OPB_ABus(O:31), Mn_ABus(0:31) (OPB Address Bus)

The OPB_ABus is used by bus masters to select a unique OPB slave attached to the OPB. The 32 lines of the address bus form a binary number which represents an address. This address will specify a one to one mapping of device functions, peripheral registers, or storage functions contained within devices which are attached to the bus.

The most significant bit of the address bus will be carried by bit 0 and the least significant bit of the address will be carried by bit 31. The most significant byte of a halfword or fullword will be the byte which corresponds to the smallest binary address.

The Mn_ABus signals from each OPB Master are AND'ed with that master's Mn_select, and the resulting buses from all OPB masters are then OR'ed together to form OPB_ABus. Thus, a master device may continue to drive data onto Mn_ABus when its select is not asserted.

See section 3.1.1, "Physical Implementation," on p. 3-2 for details on address and data bus connections.

3.2.3.2 OPB_DBus(0:31), Mn_DBus(0:31), Sln_DBus(0:31)(OPB Data Bus)

The OPB_DBus is used to transfer data between OPB peripherals. The data bus consists of 32 signals. Bit 0 is the most significant bit, and bit 31 is the least significant bit. When subdivided into bytes, Bits 0-7 represent the most significant byte on the bus.

When a peripheral is acting in concert with the DMA controller as a device requesting a write to memory, the Data Bus In is driven by the DMA peripheral. This is described in more detail in section 3.4.6, "Optional DMA Peripheral Cycle," on p. 3-52.

The Mn_DBus and Sln_DBus signals from each OPB device are AND'ed with that device's Mn_DBusEn or Sln_DBusEn, respectively, and the resulting buses from all OPB devices are then OR'ed together to form OPB_DBus. Thus, a device may continue to drive data onto its DBus when its DBusEn is not asserted.

See section 3.1.1, "Physical Implementation," on p. 3-2 for details on address and data bus connections.

3.2.4 Data Transfer Control Signals

3.2.4.1 OPB_select, Mn_select (OPB Select)

The Mn_select is driven by an OPB master in the cycle following the assertion of that master's OPB_MnGrant signal to assume control of the bus and to indicate that a valid data

transfer cycle is in progress. The Mn_select signal qualifies all master control signals and is the enable for Mn_ABus[0:31], Mn_RNW, Mn_fwXfer, Mn_hwXfer, and Mn_seqAddr. Mn_select will continue to be driven until the master receives OPB_xferAck, OPB_retry, or OPB_timeout.

A master who has assumed control of the bus may terminate the transfer cycle and relinquish the bus at any time by deasserting Mn_select. All slaves are required to terminate the transfer in progress and reset their state machines if the select signal is deactivated. If the select is deactivated in the cycle in which the slave would have activated the Sln_xferAck or Sln_retry signal, then the slave must deactivate the Sln_xferAck or Sln_retry signal in this cycle.

3.2.4.2 OPB_RNW, Mn_RNW (OPB Read Not Write)

The OPB_RNW signal indicates the direction of a data transfer. The signal must be valid any time that Select is active. If the signal is high, the request is for the OPB slave to supply data to be read into the master. If the signal is low, the request is for the OPB slave to accept write data from the master to the slave. For a write operation, the first data to be transferred must be placed on Mn_DBus when Mn_select is asserted.

3.2.4.3 OPB_fwXfer, Mn_fwXfer, OPB_hwXfer, Mn_hwXfer (OPB Transfer Size)

The transfer Size signals are asserted by the bus master to indicate the size of the requested transfer. They are used in conjunction with the OPB_fwAck and OPB_hwAck signals from the slave to implement Dynamic Bus Sizing on the OPB.

Table 3-5. OPB_fwXfer and OPB_hwXfer Encoding

OPB_fwXfer	OPB_hwXfer	Transfer Size
0	0	Byte
0	1	Halfword
1	0	Reserved
1	1	Fullword

3.2.4.4 OPB_seqAddr, Mn_seqAddr (OPB Sequential Address)

To reduce access latency for sequential addresses, the OPB_seqAddr signal is provided. This signal is asserted by the bus master to indicate that the transfer being performed will be followed with a transfer to the next sequential address. The slave which receives this signal may assume that there will be no intervening bus operations to addresses other than the next sequential address

This signal is always used in conjunction with the bus arbitration lock in order to guarantee that there are no intervening bus operations that might occur to non-sequential addresses.

This signal can help the bus slave to avoid address decode cycle, to improve data transfer performance.

If OPB slave device ignores this signal, the data transfer proceeds normally.

3.2.4.5 Mn_DBusEn, Sln_DBusEn (Master Data Bus Enable)

Mn_DBusEn and Sln_DBusEn signals are used to enable a master or slave device's data onto the OPB data bus during write and read transfers, respectively. The Mn_DBus and Sln_DBus bus are AND'ed with these signals prior to being OR'ed together with other devices' data buses to form OPB_DBus. Master and slave devices may thus continuously drive their data output buses, and only drive the enable signals during valid transfer cycles. See section 3.1.1, "Physical Implementation," on p. 3-2 for details on address and data bus connections.

3.2.4.6 OPB_xferAck, Sln_xferAck (OPB Transfer Acknowledge)

Transfer Acknowledge signal is asserted by the addressed slave to indicate the completion of a transfer between the OPB master and the OPB slave. In the case of write operations, this means that the slave has accepted the data which presently appears on the data bus, or will do so at the end of this cycle. In the case of read operations, this means that the slave has placed the data to be transferred to the OPB master on the data bus or will drive the data on the data bus prior to the end of this cycle. Transfer Acknowledge qualifies the device width control signals Sln_hwAck and Sln_fwAck, and the Error Acknowledge signal Sln_errAck. Sln_xferAck must be asserted within 16 cycles of OPB_select to prevent a timeout (unless Sln_ToutSup is asserted).

Sln_xferAck should not be asserted if Sln_retry is asserted.

If Sln_xferAck is asserted in the same cycle in which OPB_timeout is asserted, the requesting master should ignore the OPB_timeout signal and complete the data transfer.

3.2.4.7 OPB_fwAck, Sln_fwAck, OPB_hwAck, Sln_hwAck (OPB Transfer Size Acknowledge)

The Transfer Size Acknowledge signals are asserted by a bus slave to indicate its device width (i.e., which bits of the data bus it is utilizing). All byte devices are attached to DBus[0:7]; halfword devices are attached to DBus[0:15], and fullword devices are attached to the full DBus[0:31]. These signals are used in conjunction with the OPB_fwXfer and OPB_hwXfer signals from the master to implement Dynamic Bus Sizing on the OPB.

Sln_fwAck and Sln_hwAck may be asserted immediately upon a slave device's decode of its address during a transfer cycle (OPB_select asserted). They *must* be valid when Sln_xferAck is asserted. Table 3-6 shows the encoding for the Transfer Size Acknowledge signals.

Table 3-6. OPB_fwAck and OPB_hwAck Encoding

OPB_fwAck	OPB_hwAck	Slave Device Width
0	0	Byte
0	1	Halfword

Table 3-6. OPB_fwAck and OPB_hwAck Encoding (cont.)

OPB_fwAck	OPB_hwAck	Slave Device Width
1	0	Fullword
1	1	Reserved

3.2.4.8 OPB_errAck, SIn_errAck (OPB Error Acknowledge)

The Error Acknowledge signal is asserted by a bus slave to indicate that the slave encountered an error in performing the requested transfer. SIn_errAck may be asserted immediately upon a slave device's decode of its address during a transfer cycle (OPB_select asserted), or any time thereafter. It *must* be valid when SIn_xferAck is asserted.

3.2.4.9 OPB_toutSup, SIn_toutSup (Slave Time-out Suppress)

The Time-out Suppress signal is asserted by an OPB slave to indicate to the OPB Arbiter that the bus operation will be delayed for an extended period of time. This signal must be asserted within 16 cycles from the assertion of OPB_select to prevent a bus timeout. SIn_ToutSup will be used by the OPB Arbiter to disable the timeout counter and suppress the assertion of OPB_timeout. SIn_ToutSup must remain asserted until the slave can complete the requested operation.

3.2.5 Optional DMA Peripheral Support Signals

3.2.5.1 SIn_dmaReq (Slave DMA Request)

The DMA peripheral asserts this line to the DMA controller to indicate its readiness to transfer data. Before making the request, the DMA channel should have been programmed to indicate this device's width, that it is an internal device, the transfer direction, the target address, and the timing parameters associated with the DMA peripheral.

3.2.5.2 DMA_SInAck (DMA Slave Acknowledge)

After receiving a request from a DMA peripheral, the DMA channel arbitrates for control of the PLB. When the PLB grants the DMA channel the bus, it places the memory address on the bus. The DMA controller will have been programmed to indicate that the DMA device is internal. If the DMA transfer has been programmed to be a Read-From-Requesting device, the DMA device should drive the OPB_Dbus with the data; on writes, the data on the OPB_Dbus is latched into the device. In both cases, the data should be valid on the last cycle of DMA_SInAck. This can be controlled by the Peripheral Wait Time parameter in the DMA channel control register. This register indicates the number of cycles that ack should be asserted.

3.3 OPB Interfaces

The OPB I/O signals are grouped under the following interface categories depending on their function. See section 3.2, "OPB Signals," on p. 3-5 for detailed functional description.

- OPB Master Interface
- OPB Slave Interface
- OPB Arbiter Interface
- Optional DMA Interface

3.3.1 OPB Master Interface

Figure 3-2 shows all master interface signals. These signals are used to connect masters on the OPB bus. OPB master device can also act as OPB slave device for other OPB bus master request. See section 3.2, "OPB Signals," on p. 3-5 for detailed functional description.

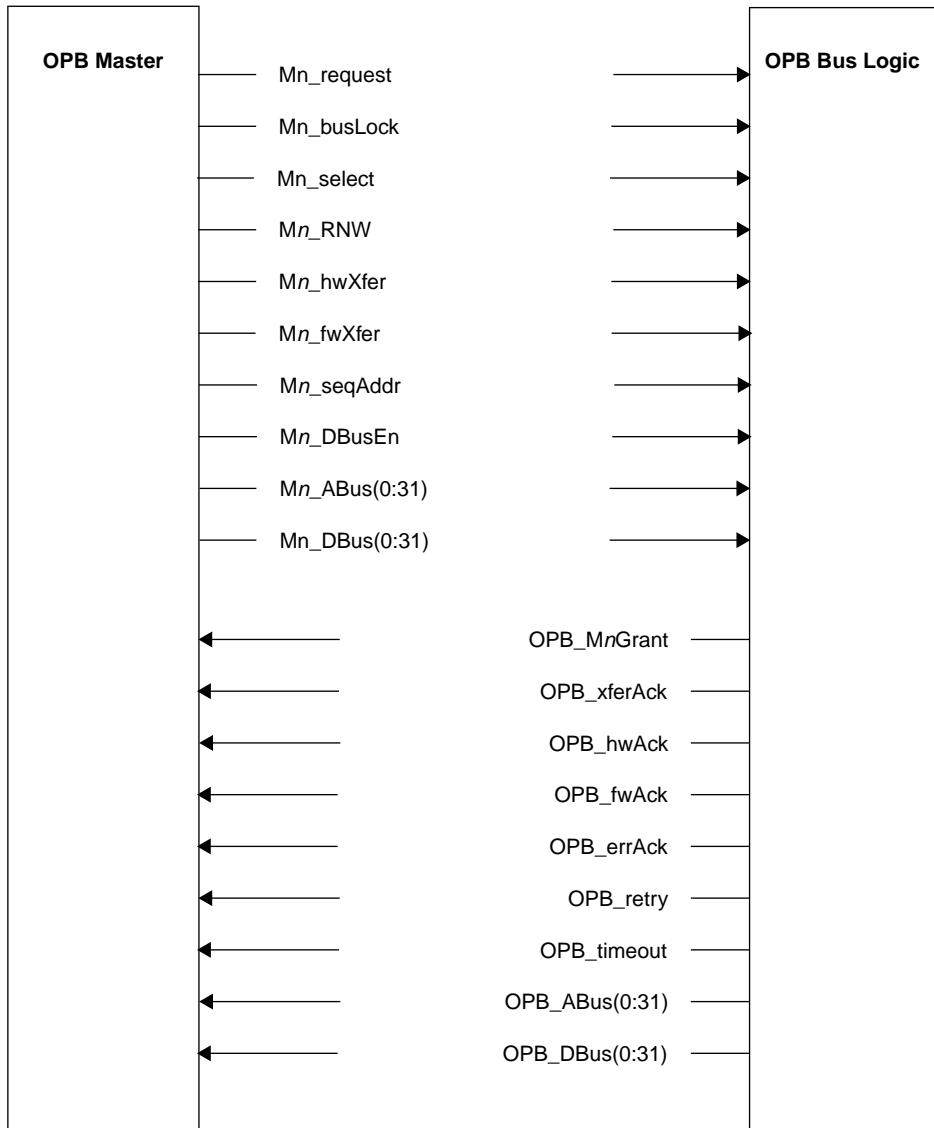


Figure 3-2. OPB Master Interface

3.3.2 OPB Slave Interface

Figure 3-3 shows all OPB slave interface signals. These signals are used to connect slave devices on the OPB bus. This diagram also describes a fullword device with all 32 bits of OPB_DBus and SIn_DBus connected. Slave devices may also be of byte (8bit) or halfword (16 bit) widths. See section 3.2, "OPB Signals," on p. 3-5 for detailed functional description

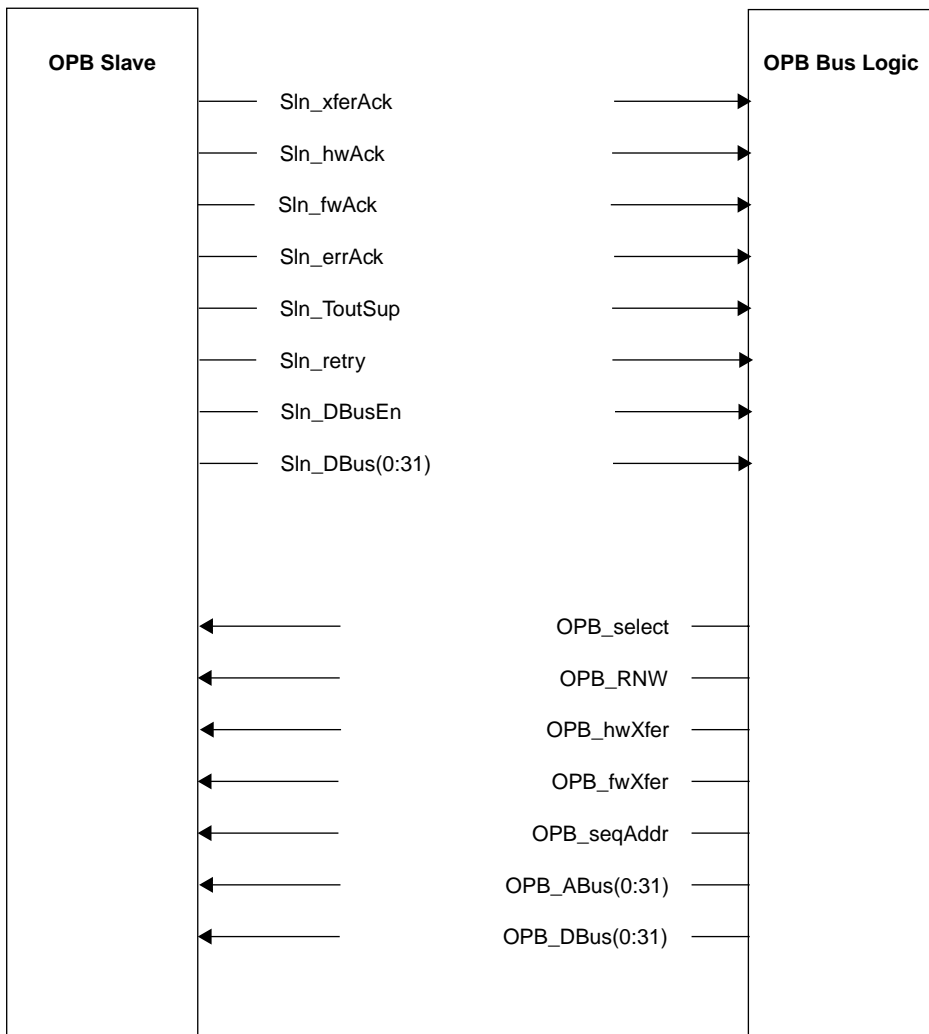


Figure 3-3. OPB Slave Interface

3.3.3 OPB Arbiter Interface

Figure 3-4 shows all OPB arbiter interface input/output signals. These signals are used to connect OPB bus arbiter to the OPB bus. Note that some slave interface signals are used to access the internal registers of the arbiter. Only those signals shown attached to the arbiter itself are necessary to implement the OPB arbiter function. See section 3.2, "OPB Signals," on p. 3-5 for detailed functional description

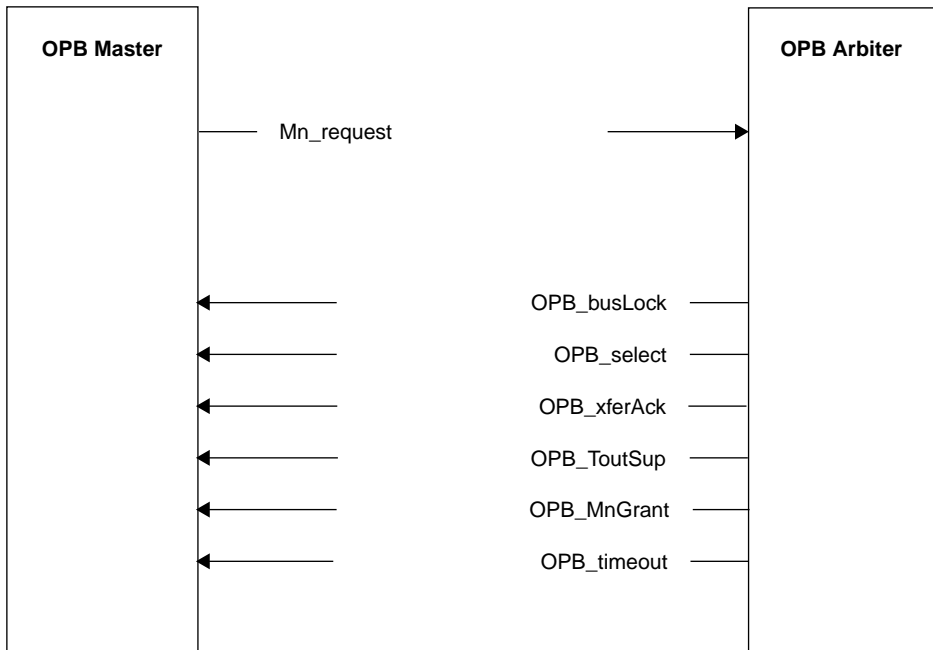


Figure 3-4. OPB Arbiter Interface

3.3.4 Optional DMA Interface

The optional DMA support consists of pairs of request and acknowledged signals as described below. In operation, a device will be connected to a particular DMA channel on the DMA controller by direct connection to the channels inputs.

Figure 3-5 shows all DMA input output signals. See section 3.2, "OPB Signals," on p. 3-5 for detailed functional description

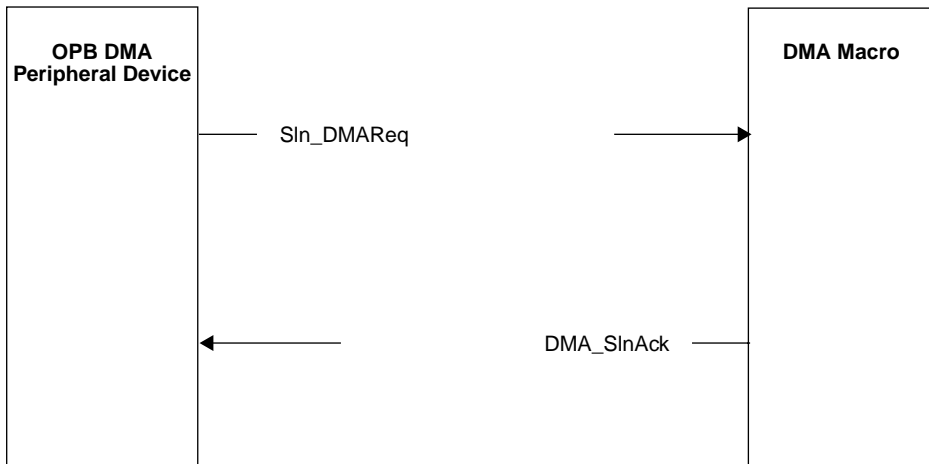


Figure 3-5. Optional DMA Interface

3.4 OPB Operations

This section discusses in detail the OPB operations which include:

- OPB Bus Arbitration Protocol
- Data Transfer Protocol
- Dynamic Bus Sizing
- Optional DMA Peripheral Cycle

3.4.1 OPB Bus Arbitration Protocol

This section on bus arbitration discusses in detail the OPB basic bus arbitration, continuous bus request, bus lock signal, multiple bus request arbitration or overlapped bus arbitration, bus master priority, and reduced latency arbitration using bus parking.

3.4.1.1 OPB Basic Bus Arbitration

OPB bus arbitration proceeds by the following protocol:

1. An OPB master asserts its bus request signal.
2. The OPB arbiter receives the request, and outputs an individual grant signal to each master according to its priority and the state of other requests.
3. An OPB master samples its grant signal at the rising edge of OPB clock. In the following cycle, the OPB master may initiate a data transfer between the master and a slave device by asserting its select signal.

The bus grant signal is only issued by the OPB arbiter during a valid bus arbitration cycle, defined as either:

- *Idle*, which means that the OPB_select and OPB_busLock are deasserted, indicating no data transfer is in progress, or
- *Overlapped arbitration cycle*, which means that the OPB_xferAck is asserted, indicating the final cycle in a data transfer, and OPB_busLock is not asserted. Arbitration in this cycle allows another master to begin a transfer in the following cycle, avoiding the need for a 'dead' cycle on the bus.

Figure 3-6 shows typical OPB bus arbitration cycle.

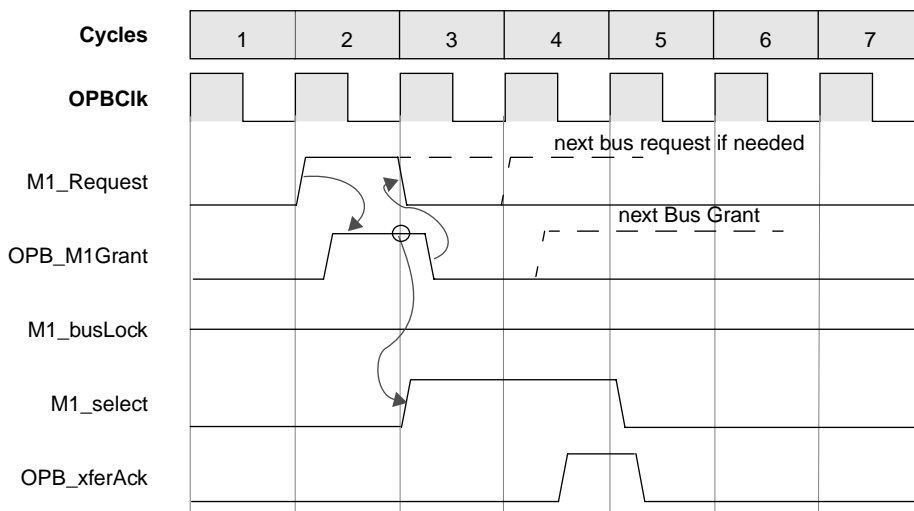


Figure 3-6. OPB Basic Bus Arbitration

3.4.1.2 OPB Bus Arbitration - Continuous Bus Request

An OPB master device need not deassert its request upon receipt of a bus grant signal if it has multiple bus transfer cycles to perform.

Figure 3-7 shows an OPB bus arbitration cycle in which an OPB master device asserts bus request continuously for four data transfer cycles. Bus grant is asserted during valid arbitration cycles.

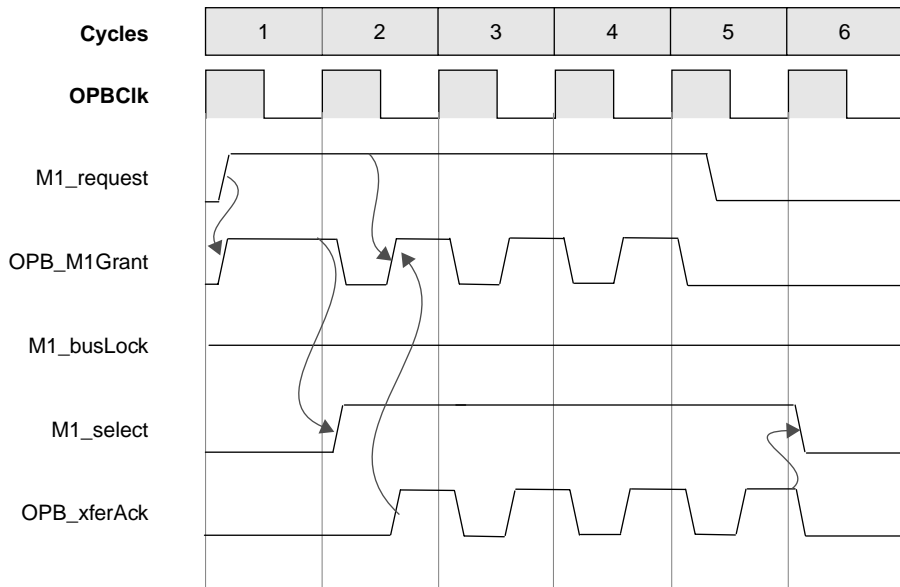


Figure 3-7. OPB Bus Arbitration - Continuous Bus Request

Note: Even if an OPB master device asserts request continuously, it will not necessarily receive a valid grant signal. Other OPB masters with higher bus priority may request the OPB, and will be granted according to OPB arbiter priority (see section 3.4.1.5, "OPB Bus Master Priority," on p. 3-24). If an OPB master device needs a non-interruptible sequence of bus cycles, it can use the busLock signal for this purpose.

3.4.1.3 OPB Bus Arbitration - BusLock Signal

If an OPB master asserts the busLock signal upon assuming control of the bus, the OPB arbiter will continue to grant the OPB to the master which locked the bus. Grant signals will be generated if the master asserts its request signal, during valid arbitration cycles. Bus request and grant signals have no effect on bus arbitration, and the master which asserted busLock will retain control of the bus until busLock is deasserted for at least one complete cycle.

Figure 3-8 shows a typical OPB arbitration cycle utilizing busLock. The master device asserts busLock immediately upon assuming control of the bus, and retains control of the bus until it deasserts busLock. Bus grants will be asserted as shown only if the master request is asserted.

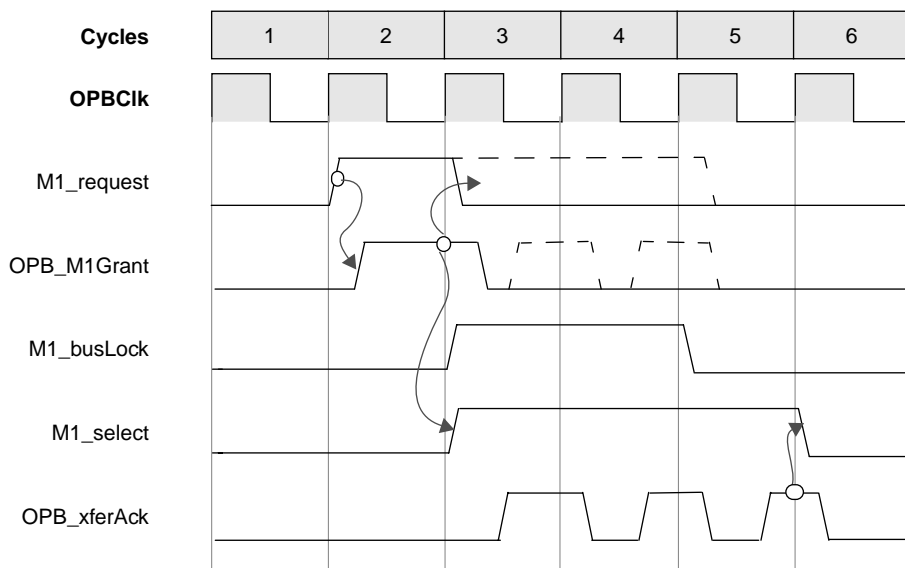


Figure 3-8. OPB Bus Arbitration - BusLock Signal

In the above example, OPB master 1 requires three non-interruptible cycles of data transfer. OPB slave device has one cycle data transfer latency. M1_busLock signal is asserted along with M1_select. The OPB master may proceed with data transfer cycles while asserting busLock without engaging in bus arbitration, and without regard to the state of the request and grant signals. The OPB arbiter will detect the busLock signal, and will continue to grant the bus to the current master, regardless of other (higher priority) requests.

Note: In the above example, cycle 2 and 5 are the arbitration cycles and cycle 3 and 4 are bus locked.

3.4.1.4 OPB Multiple Bus Master Arbitration

Figure 3-9 shows multiple bus request or overlapped bus arbitration. In the following example, master 1 and 2 simultaneously request the OPB master, master 1 having higher priority.

This overlapped bus arbitration allows for efficient utilization of OPB bandwidth. For additional detailed discussion, see section 3.4.2, "Data Transfer Protocol," on p. 3-26.

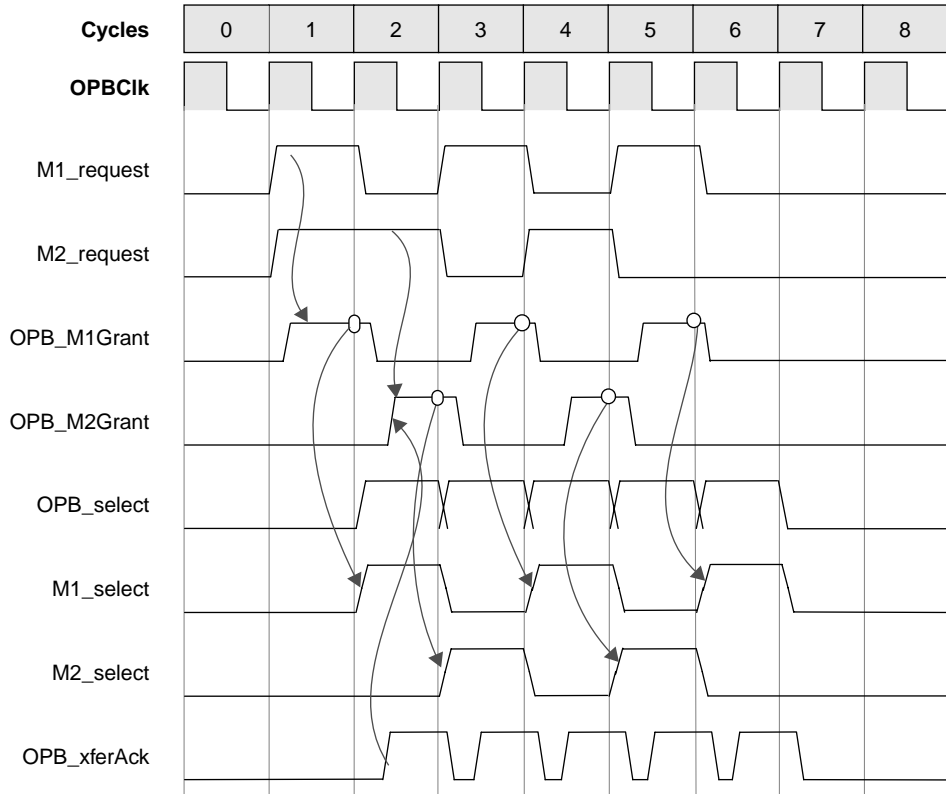


Figure 3-9. OPB Multiple Bus Request Arbitration

3.4.1.5 OPB Bus Master Priority

The OPB architecture requires an arbiter that includes a timeout counter, to resolve competing requests for OPB resources by OPB master devices.

The following sections describe several possible implementations of an OPB arbiter

Fixed Priority

Priority is set in hardware within the arbiter. The system designer assigns relative priorities to OPB master devices via the way they are attached to the arbiter. This is the simplest arbitration procedure, the easiest and least costly to implement, and the least flexible.

Programmable Bus Priority

Bus arbitration priority is determined by the priority fields of a register which is accessible by software. The relative priority of each master attached to the OPB can be specified by each application. To ensure an orderly boot-up, the fields of the priority register are uniquely set upon system reset to a default value. Thus, there is a default priority among masters determined by the order in which they are attached to the arbiter by the system designer.

This arbitration procedure is more complex than that of a fixed priority, and requires access to the arbiter's resources (as a slave device on the OPB, or through some other addressable interface). This procedure is appropriate for situations where the same system will be used in different applications, or in situations in which the peripherals attached to the OPB will experience different workloads and require different priorities.

Self-modifying Bus Priority

In applications where peripherals of equal priority and comparable levels of utilization are attached to the OPB, a fair procedure of allocating priorities among peripherals may be appropriate. One example would be a self-modifying priority, whereby the relative priorities of each master are altered following every complete bus arbitration (each request/grant transaction). Several priority allocation algorithms are commonly used, including 'Round-Robin,' 'Least Recently Used (LRU),' and a selection from among several fixed priorities. The minimum requirement is that a self-modifying priority be provided which guarantees that no requesting master can be completely locked out from the OPB.

3.4.1.6 OPB Bus Parking

The OPB architecture provides the ability to park on one master. The parked master will continue to receive a grant signal during valid arbitration cycles, when no request is asserted by any master. This allows the parked master to access the bus without delay due to an arbitration cycle, thus reducing latency. Figure 3-10 shows reduced latency arbitration using bus parking.

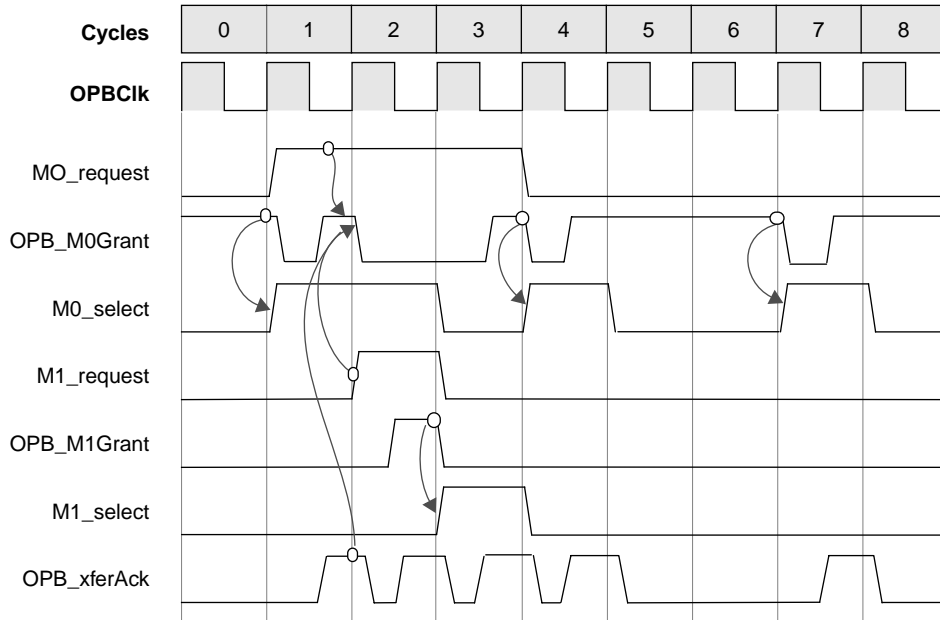


Figure 3-10. Reduced Latency Arbitration using Bus Parking

In the above example, the bus is parked on master 0, master 1 having the higher priority. The OPB is parked on master zero in cycle 0. Since no master is requesting, master 0 is granted the bus. Master 0 is then able to assert select signal in cycle 1 to assume control of the bus and begin a data transfer cycle, as well as issue a request for subsequent cycles. It avoids the necessity of an arbitration cycle due to the fact that the bus was parked on master 0, and its grant was asserted in the previous cycle. Master 1 in turn requests the bus in cycle 2, and is granted because of its higher priority. The grant to master 0 is removed. Having been granted in the previous arbitration cycle, master 1 now asserts select in cycle 3. Note that this is the normal OPB arbitration sequence. Master 1 removes its request, and the lower priority master 0 is again granted the bus. Master 0 in turn asserts select in cycle 4. The OPB is parked on master 0 in cycles 5 and 6, which means that the grant signal for master 0 is asserted, since no other bus masters are requesting. Finally master 0 asserts select and begins a data transfer cycle in cycle 7, having been granted the bus in the previous cycle via bus parking. Master 0 need not assert its request for this transfer, as it requires the bus for only one cycle.

3.4.2 Data Transfer Protocol

This section on data transfer discusses in detail the basic data transfer, overlapped bus arbitration, continuous bus request, bus lock operation, sequential address signal operation, slave retry operation, bus timeout error, and timeout error suppression.

3.4.2.1 OPB Basic Data Transfer

Figure 3-11 shows typical OPB data transfer cycle. In the following example, fullword master device 1 reads data from fullword slave device 2. Note that slave device 2 has a two-cycle latency. When the OPB master device requires access to OPB, it asserts its request signal. The OPB arbiter will assert the master's grant signal according to bus arbitration protocol, and during a valid bus arbitration cycle. The requesting OPB master device assumes OPB ownership by asserting its select signal, in the cycle following that in which it samples its grant at the rising edge of OPB Clock. The slave completes the transfer by asserting xferAck, which causes the master to latch data from the data bus on read transfers, and deassert select.

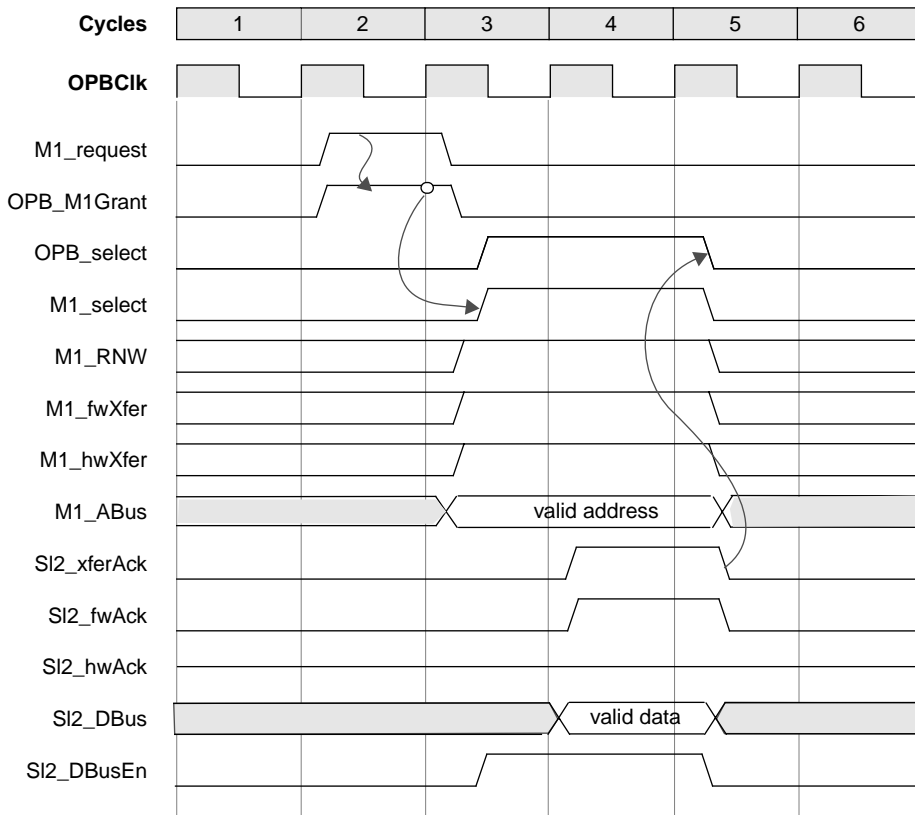


Figure 3-11. OPB Basic Data Transfer

3.4.2.1.1 Fullword - Fullword Read and Write Operation 1

Figure 3-12 shows fullword - fullword read and write operation with slave 3 having one cycle latency. OPB master 1 reads data from slave 3 and writes data to slave 3.

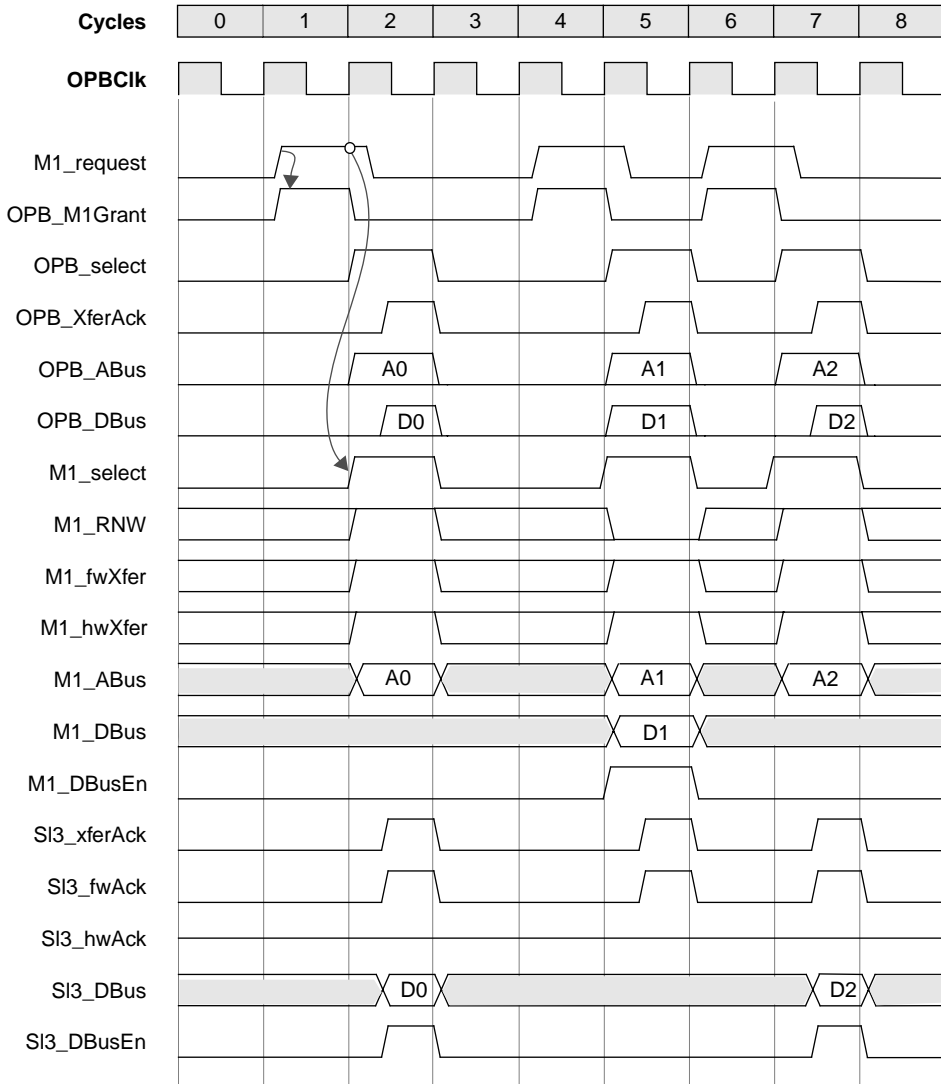


Figure 3-12. Fullword - Fullword Read and Write Operation 1

3.4.2.1.2 Fullword - Fullword Read and Write Operation 2

Figure 3-13 shows the fullword - fullword read and write operation 2 with slave 3 having two cycle latency. Master 1 reads data from slave 3 and writes data to slave 3.

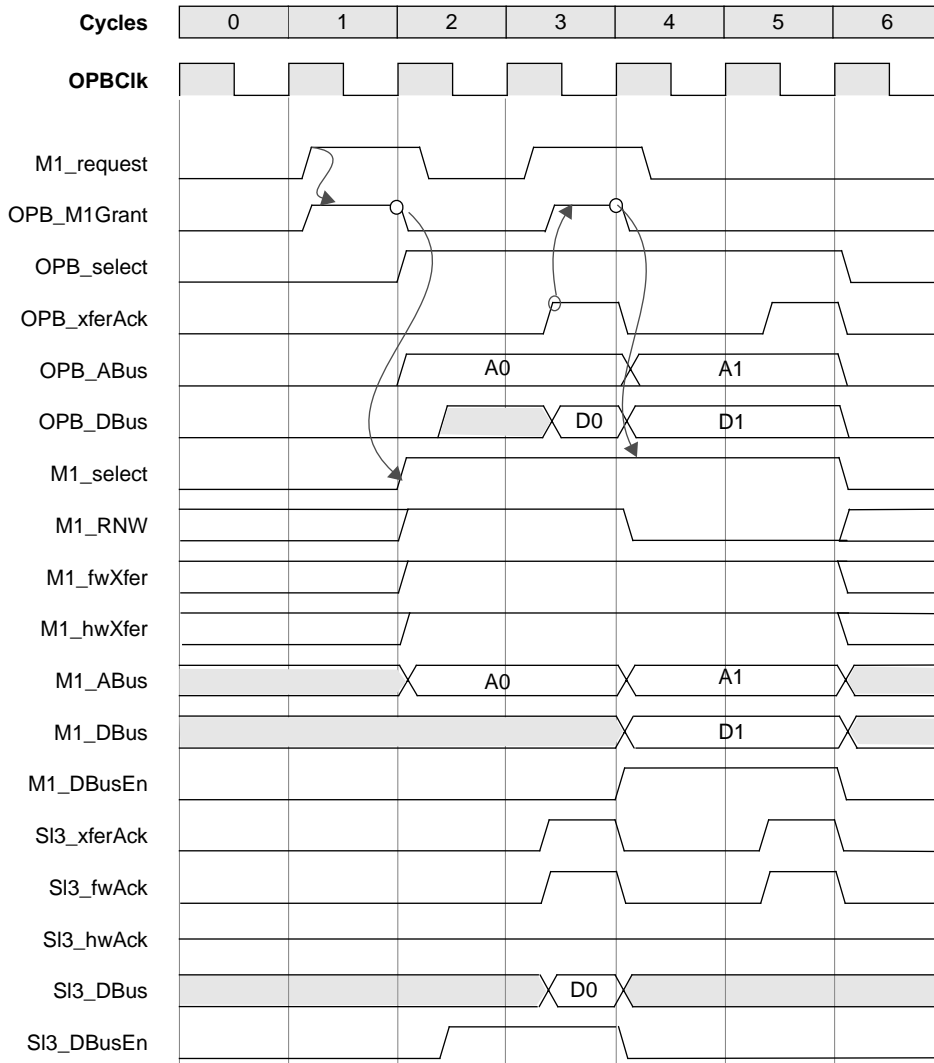


Figure 3-13. Fullword - Fullword Read and Write Operation 2

3.4.2.2 Overlapped Bus Arbitration

Figure 3-14 shows a more general case of OPB data transfer. In the following example, two OPB masters, master 1 and 2, require the OPB and assert requests in the same cycle. Master 1 is a fullword device, and requests a read operation from slave 3. Master 2 is a fullword device and also requests a read operation from slave 3. Slave 3 is a fullword device with a two-cycle latency. OPB Master 1 has high bus priority.

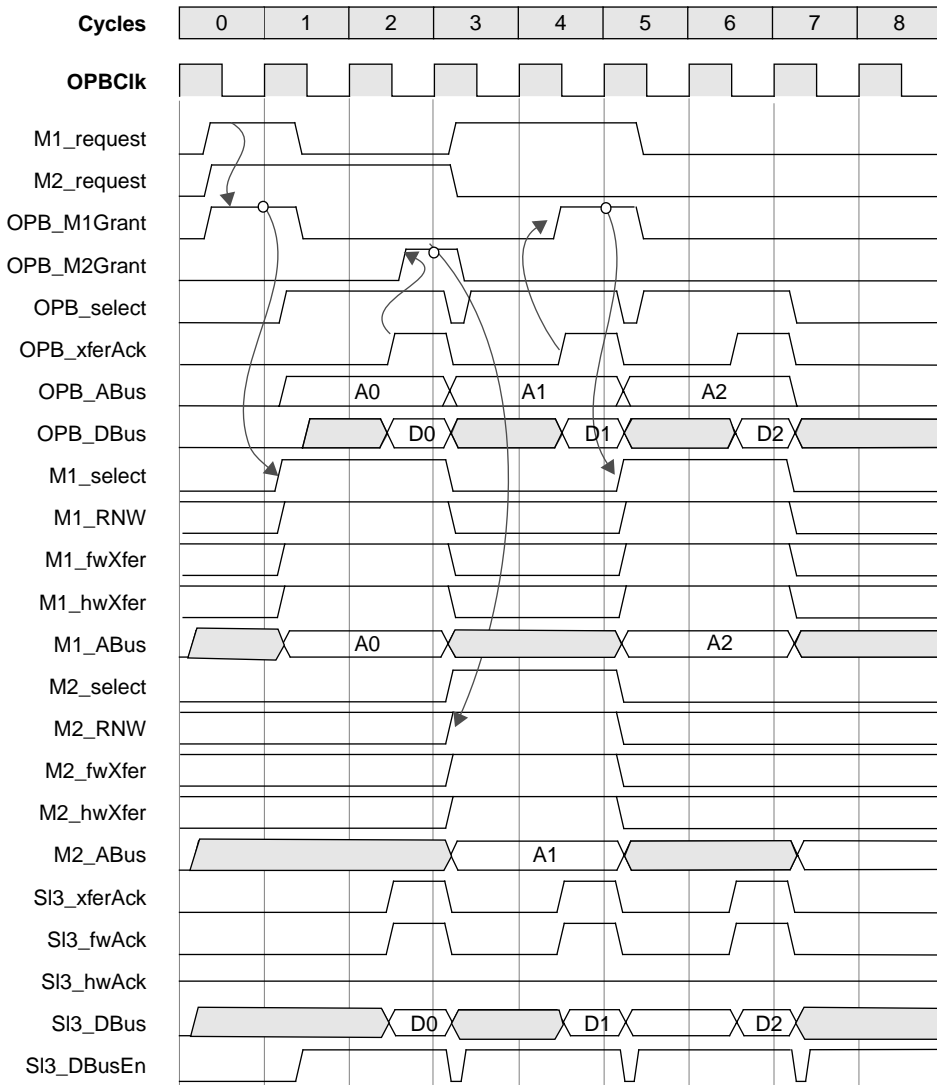


Figure 3-14. OPB Data Transfer

In the above example, the data transfer operation proceeds as follows:

- Master 1 and 2 assert requests in cycle 0. OPB arbiter grants to master 1 due to its higher priority by asserting OPB_M1Grant.
- Upon sampling OPB_M1Grant on the rising edge, master device 1 initiates a data transfer in cycle 1 by asserting M1_select, and negates M1_request. The OPB arbiter negates OPB_M1Grant. OPB_M2Grant is not asserted since this is not a valid arbitration cycle (bus is busy but not in final transfer cycle).
- Slave 3 acknowledges in cycle 2, indicating completion of the data transfer, by asserting SI3_xferAck. Master 1 latches data on OPB_DBus at the end of this cycle, and relinquishes control of the bus by deasserting M1_select, which also gates off all its control signals. The OPB arbiter asserts OPB_M2Grant upon the assertion of OPB_xferAck, overlapping arbitration with the data transfer.
- Master 2 then samples OPB_M2Grant at the rising edge in cycle 3, and initiates data transfer by asserting M2_select, and negates M2_request. The OPB arbiter negates OPB_M2Grant.
- Slave 3 acknowledges in cycle 4, indicating completion of the data transfer, by asserting SI3_xferAck. Master 2 latches data on OPB_DBus at the end of this cycle, and relinquishes control of the bus by deasserting select, which also gates off all of its control signals. The OPB arbiter asserts OPB_M1Grant upon the assertion of OPB_xferAck, overlapping arbitration with the data transfer.
- Upon sampling OPB_M1Grant on the rising edge in cycle 5, master 1 initiates a data transfer by asserting M1_select, and negates M1_request. The OPB arbiter negates OPB_M1Grant.
- Finally slave 3 acknowledges in cycle 6, indicating completion of the data transfer, by asserting its SI3_xferAck signal. Master 1 latches data on OPB_DBus at the end of this cycle, and relinquishes control of the bus by deasserting M1_select, which also gates off all its control signals.

Under this protocol, bus arbitration and data transfer are overlapped. This allows OPB to transfer data efficiently, avoiding dedicated bus arbitration cycles.

3.4.2.3 Continuous Bus Request

Figure 3-15 shows an OPB data transfer cycle when the OPB master continuously asserts its bus request signal. In the following example, OPB master 2 requests four consecutive data transfers and OPB master 1 requests one. Masters 1 and 2 are fullword devices, and both request read operations from slave device 3. Slave device 3 is a fullword device and has one cycle latency. OPB master 1 has higher bus priority than master 2. Data transfer of OPB master 2 is interrupted by OPB master 1 at cycle 3.

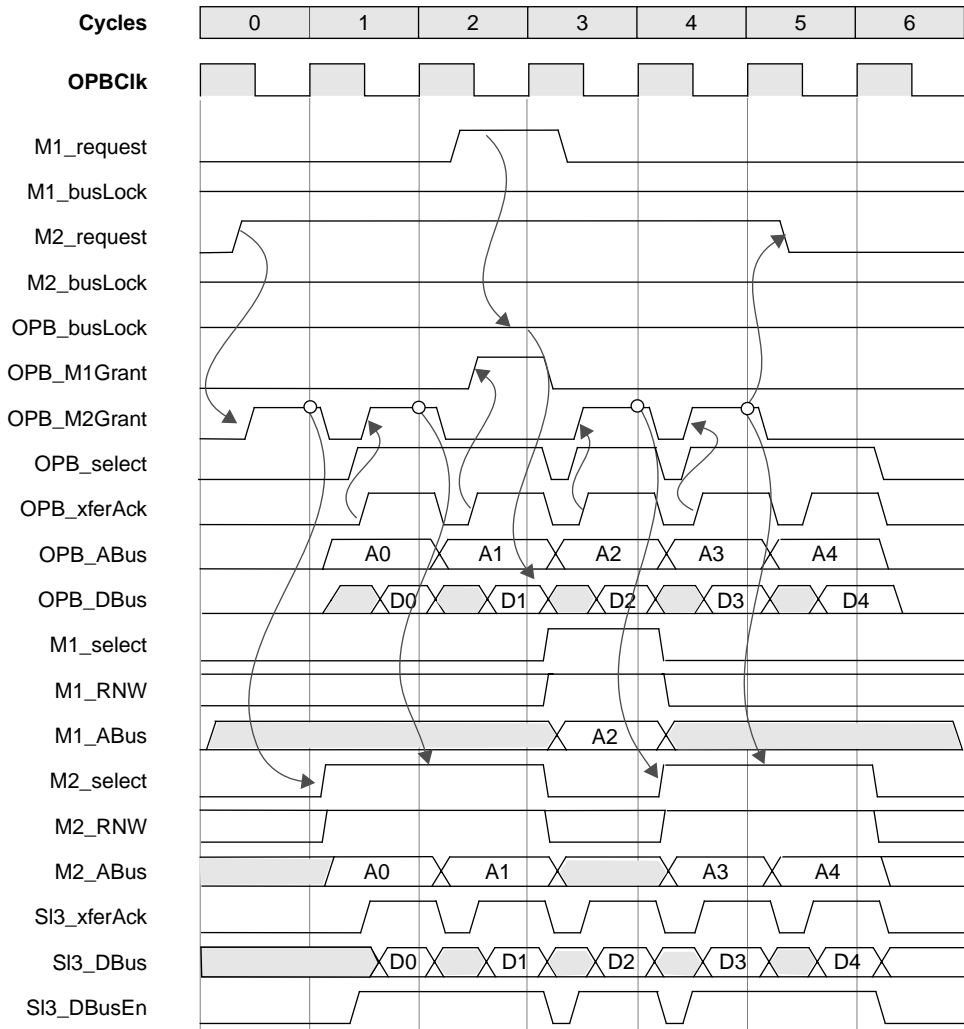


Figure 3-15. Continuous Bus Request

3.4.2.4 Bus Lock Operation

The OPB provides OPB_busLock signal for continuous data transfer cycles with no interruption. Figure 3-16 shows a data transfer operation with the busLock signal. In this example, OPB master 1 reads data from slave 3 four times in a continuous sequence without interruption. Master 1 and 2 are fullword devices that request to read data from slave 3. Slave 3 is fullword device with 1 cycle latency. Master 1 has priority. BusLock signal will normally be negated at the beginning of last data transfer cycle (in this case, cycle 4). Otherwise, OPB requires one bus arbitration cycle.

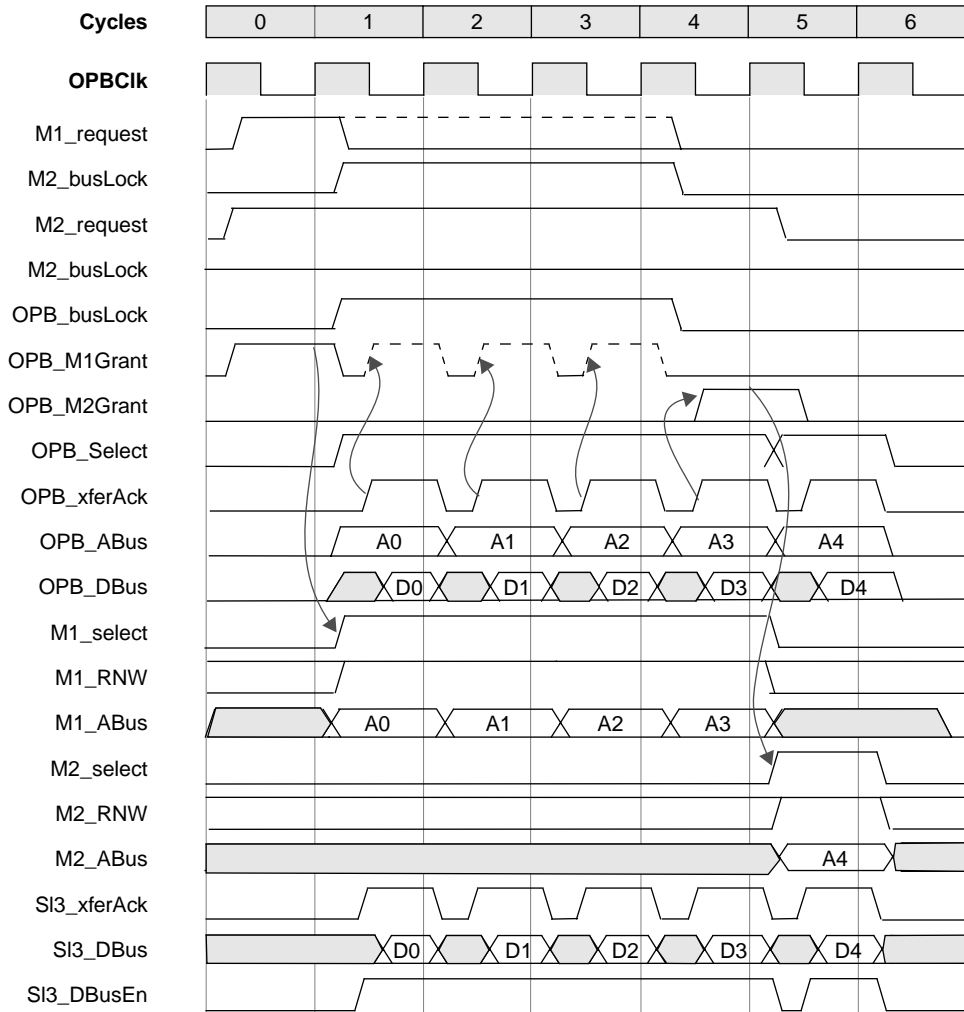


Figure 3-16. Bus Lock Data Transfer Cycle

Figure 3-17 demonstrates the case in which an extra arbitration cycle is necessary due to busLock not being deasserted during the last data transfer cycle. Device conditions are the same as for the previous figure. In the following example, master 1 asserts busLock signal during its data transfer and continues to assert busLock through the last transfer cycle. A grant is generated at cycle 4 if the master asserts request. Even if the master does not assert request, there will be no arbitration cycle there, due to busLock being asserted, which prevents the normal OPB overlapped arbitration. The OPB requires one additional cycle for arbitration, to grant master 2 at cycle 5.

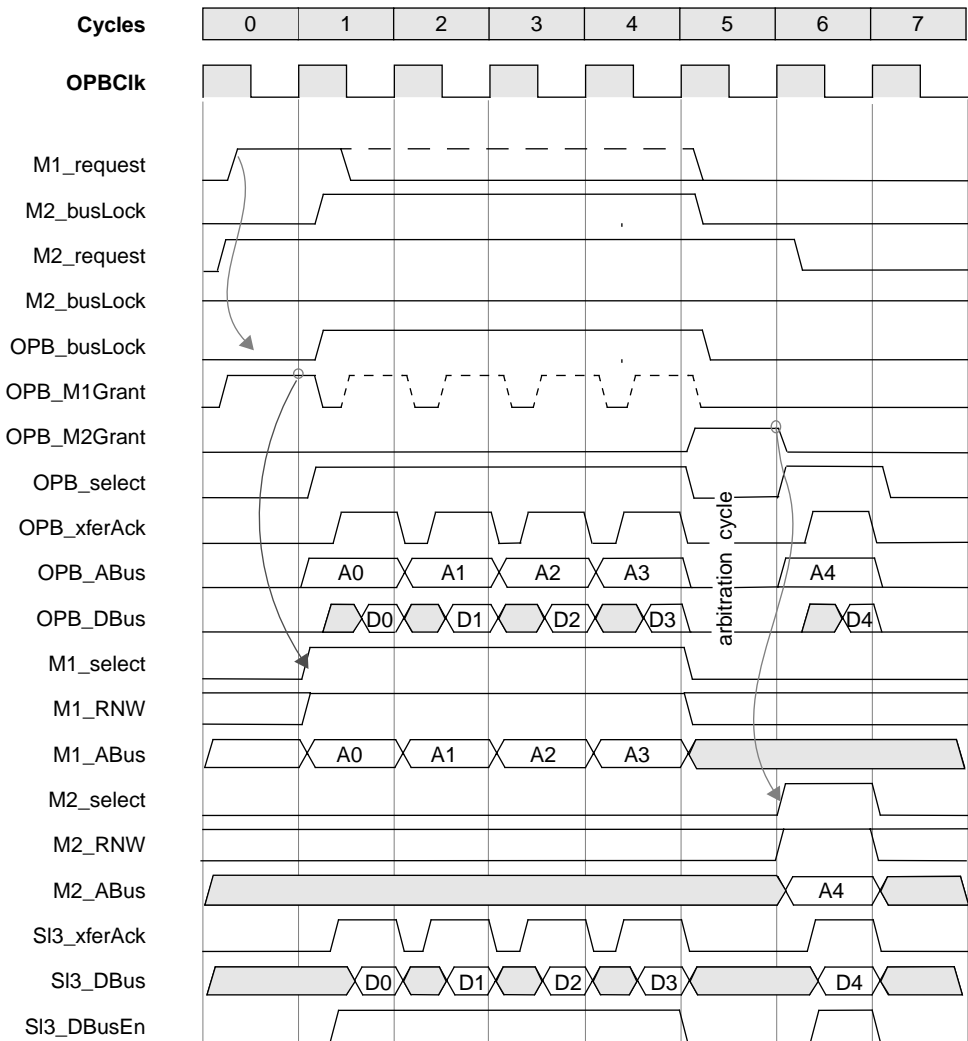


Figure 3-17. Bus Lock Signal Penalty Case

3.4.2.5 Sequential Address Signal Operation

The OPB provides the seqAddr signal for multiple data transfers to sequential addresses to the same slave. This signal is asserted by the OPB master to indicate that the transfer being performed will be followed with a transfer to the next sequential address. This signal is always used in conjunction with the bus arbitration lock in order to guarantee that there are no intervening bus operations that might occur to non-sequential addresses. Master 1 will read data from slave 3 four times, using SeqAddr and busLock signals. Slave 3 normally has a 2 cycle latency, but can achieve 1 cycle latency in “burst mode” by avoiding subsequent address decode cycles. Data transfer requires only 5 cycles (2+1+1+1) even if this slave normally requires two cycles to access and provide data. Without OPB_seqAddr, 8 cycles would be required.

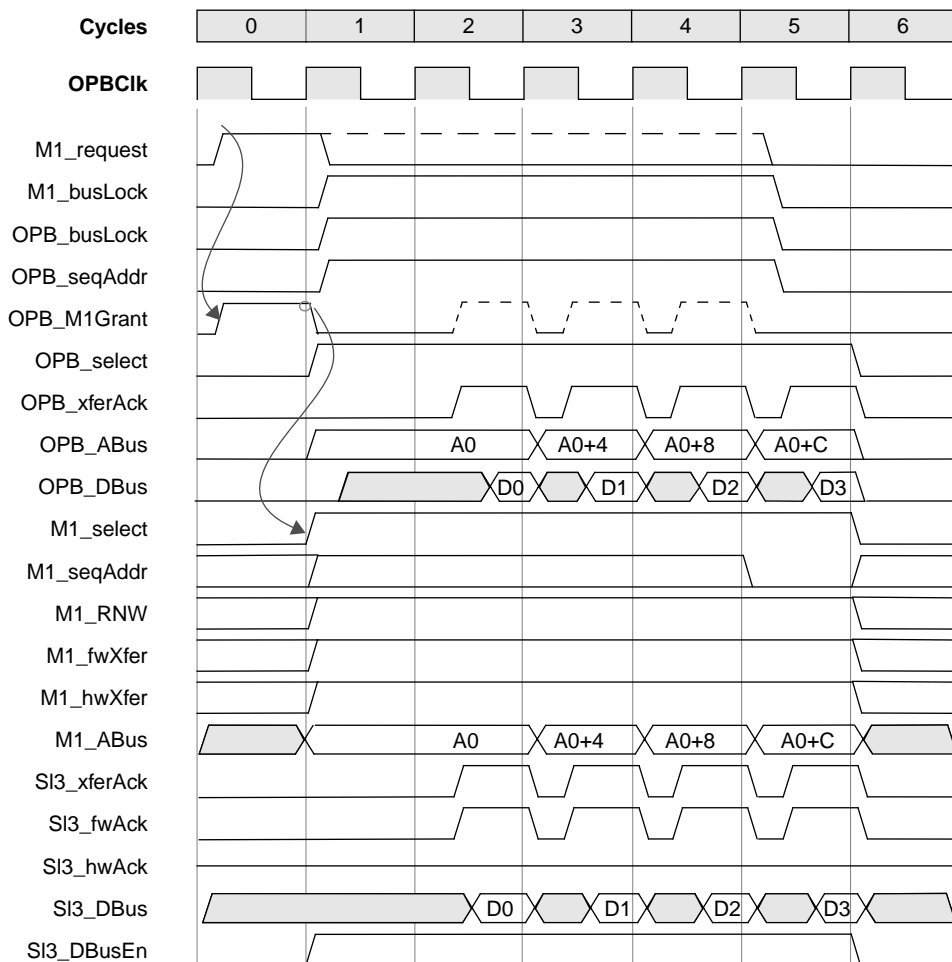


Figure 3-18. Sequential Address Signal Operation

3.4.3 Slave Re-try Operation

To alleviate deadlock scenarios on the bus, the OPB provides the `Sln_retry` signal. The retry signal forces the requesting master to abandon a pending data transfer, and allow the OPB arbiter to re-arbitrate the bus.

The retry signal is asserted by an OPB bus slave to indicate that there exists a condition which precludes the slave from performing the indicated bus operation at this time.

A bus slave will assert the `Sln_retry` signal instead of the `Sln_xferAck` signal when a situation requiring it is detected. It will remain asserted until the bus slave becomes deselected as a result of the 'select' signal being deasserted.

The bus master will respond to this signal by immediately terminating the transfer in progress and relinquishing control of the bus. This is accomplished by the master deasserting `Mn_select` and, if asserted, `OPB_busLock` in the cycle following the detection of the `OPB_retry` signal. The bus master requesting the transfer is also required to deassert `Mn_request` for one cycle following the detection of `OPB_retry` to allow for resolution of the deadlock. Following this one-cycle request back-off, the master may then proceed to request the transfer again, or request another transfer.

The OPB arbiter will re-arbitrate the bus in the cycle in which `Mn_select` and `Mn_busLock` are deasserted by the master. In this cycle, the master that was just retried is required to drop his request and this gives the slave which retried the operation a chance to access the OPB in order to resolve the deadlock condition.

The retry signal, and the requirement that retried masters terminate their transfer and back off requests for one cycle, is insufficient to guarantee that all deadlock scenarios will be alleviated. Depending on the number and type of master devices connected to the OPB, and their relative arbitration priorities, it is possible that deadlock conditions will arise which the retry will not solve. It does, however, provide master devices on the OPB with sufficient information to detect a deadlock condition, and to take appropriate actions to resolve it.

System designers should also be aware that a retry operation can break a locked series of transfers, due to the requirement that the master deassert `Mn_busLock` for one cycle if it is active. This is necessary to allow for an arbitration cycle on the OPB. This raises the possibility, however, that a retry operation could interrupt an atomic transfer, and a different master could win the next arbitration cycle. In this case, nothing prevents the new master from accessing data to/from the same address as the locked transfers.

In the following example, OPB master 2 and slave 2 are the same unit, a bridge to an external bus. That bus has initiated a write to OPB slave 3, to which it is committed and which is using all its internal resources. OPB master 1 has a higher priority on the OPB, and requests a read from slave 2. Slave 2, however, cannot respond to the read request until it performs its write to slave 3. This condition is a deadlock. It would be inappropriate for slave 2 to merely ignore master 1's request and cause a time-out, or to respond with `Sl2_errAck`, since this is not an error condition. Slave 2 thus issues `Sl2_retry`. This causes master 1 to terminate the current cycle by removing its select, and back off the bus for one cycle by

removing its request. This causes an arbitration cycle on the OPB. The master side of the external bus bridge, master 2, then wins arbitration, and performs a write to slave 3. Master 1 then resumes its request, receives a grant, and issues its read to slave 2, which is now free to respond.

Figure 3-19 shows retry signal operation.

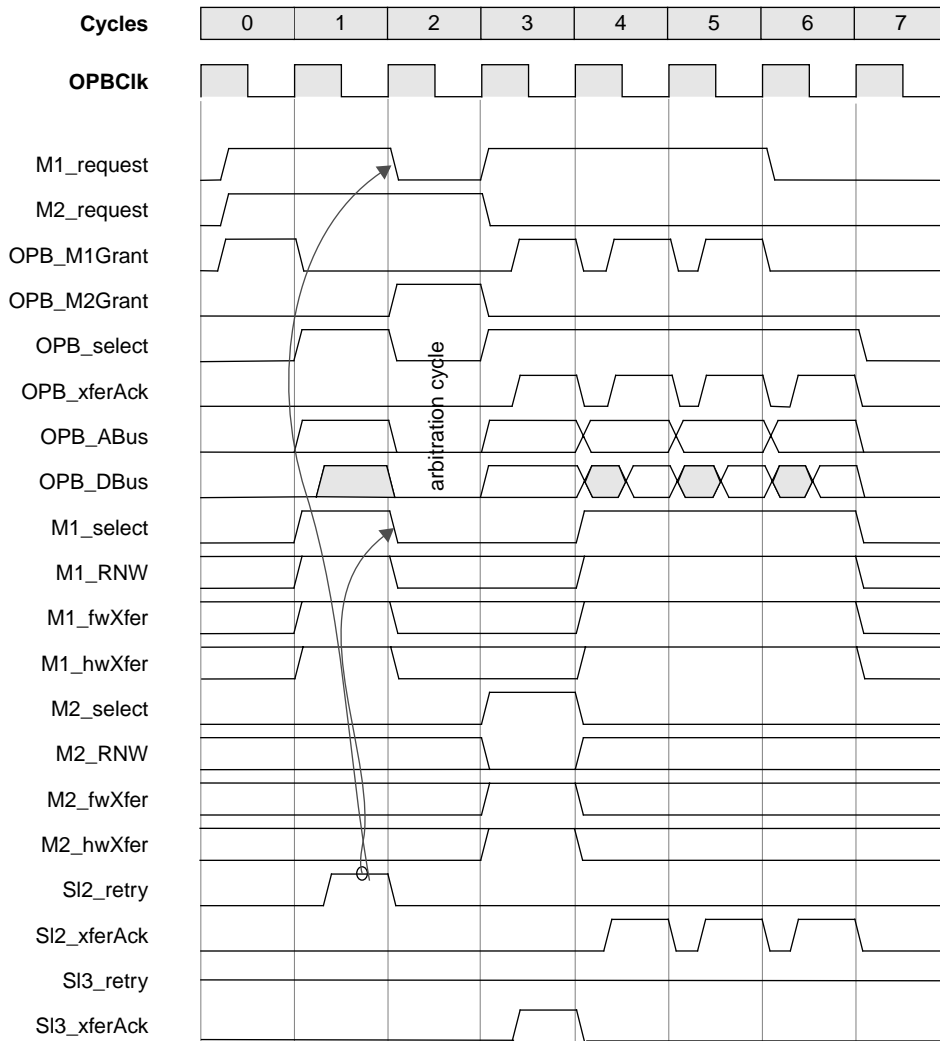


Figure 3-19. Retry Signal Operation

3.4.4 Bus TimeOut Error

An OPB timeout error will be detected by the OPB arbiter. All slave devices on the OPB must respond to OPB_select within 16 cycles from the assertion of OPB_select by asserting Sln_xferAck or Sln_retry signals. If OPB_xferAck or OPB_retry signal is not asserted, then the OPB arbiter will assert OPB_timeout signal in the 16th cycle. A slave device which requires more than 16 cycles to complete its transfer may inhibit the timeout counter by asserting Sln_ToutSup.

OPB_timeout signal is driven from the OPB arbiter to each OPB master attached to the OPB. This signal is used to indicate to the OPB master that a Timeout error has occurred and that the master should terminate the transfer. The master must drop its select signal in the cycle following the cycle in which the OPB_timeout signal is detected as being active.

If the bus is not locked, the arbiter will re-arbitrate in the cycle in which the select is deactivated and will proceed to grant a new master of the bus. If the bus is locked, the master is still required to deactivate the select signal for one cycle and then may proceed in the following cycle to re-activate the select signal to perform another transfer.

Note that specifying the timeout counter as fixed at sixteen cycles requires that a slave device respond within 16 cycles by either activating its Sln_xferAck or Sln_retry signal or by activating the Sln_ToutSup signal.

The slave may assert Sln_ToutSup any time up to and including within the 16th cycle following the assertion of OPB_select. When OPB_ToutSup is asserted, the OPB arbiter will suppress OPB_timeout signal and suspend the timeout counter in the OPB arbiter.

If OPB_xferAck and OPB_timeout are active in the same cycle (i.e., the slave responds in the 16th cycle following OPB_select), the master should accept OPB_xferAck, completing the transfer, and ignore the OPB_timeout signal.

If OPB_retry and OPB_timeout are active in the same cycle (i.e., the slave responds in the 16th cycle following OPB_select), the master should accept OPB_retry and ignore OPB_timeout. In this case the effect will be similar, as both signals require the master to relinquish control of the OPB by removing its select signal in the following cycle. OPB_retry, however, also requires that the master remove its request in the following cycle, and may initiate other, system-dependant activity on the part of the master to alleviate system deadlock.

3.4.4.1 OPB Bus Timeout Error Condition

Figure 3-20 shows OPB timeout error situation. Master 1 and 2, and slave 3 and 4 are full-word devices. Master 1 will read data from slave 3, master 2 will read data from slave 4. In this case master 1 has higher priority than master 2.

If OPB slave 3 does not respond to the OPB master 1's request within the 16 cycles, then master 1 terminates this data transfer cycle by negating M1_select, upon receipt of OPB_timeout from the arbiter.

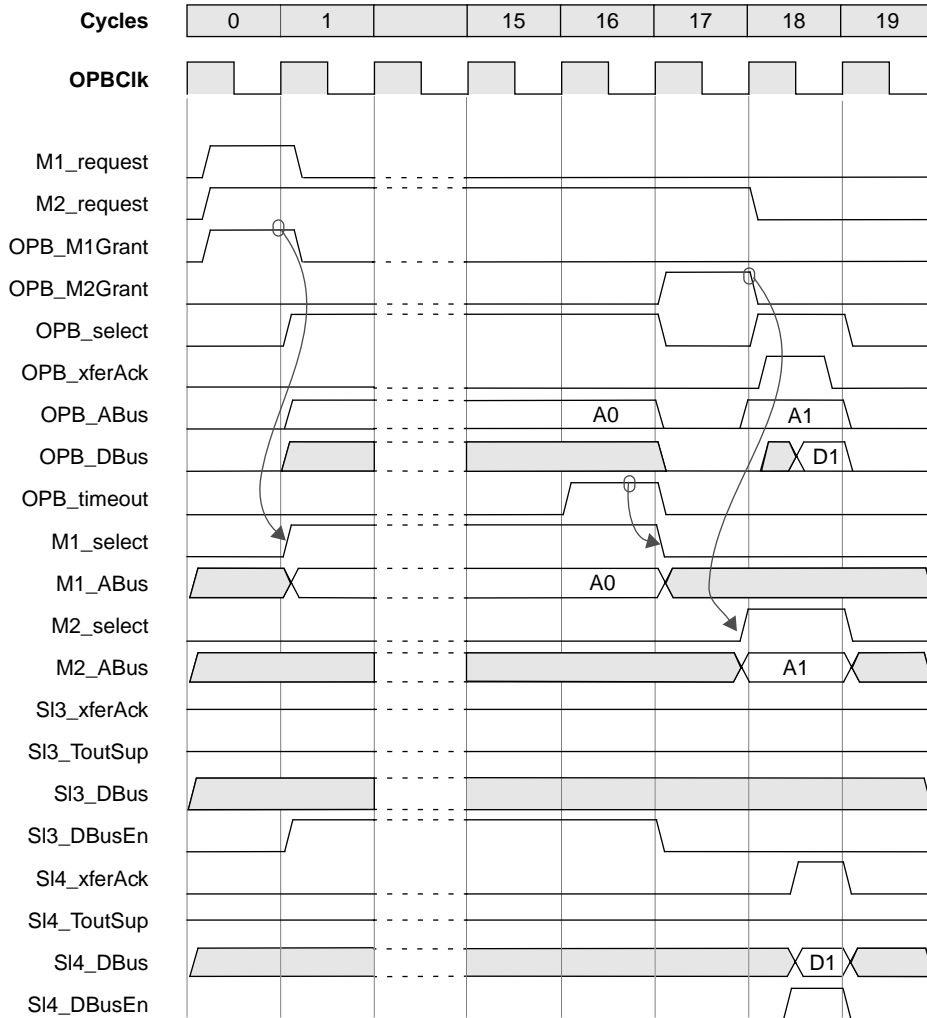


Figure 3-20. Bus Timeout Error Condition

3.4.4.2 OPB Timeout Error Suppression

Figure 3-21 shows an OPB slave suppressing TimeOut error with the ToutSup signal. Master 1 and 2, and slave 3 and 4 are fullword devices. Master 1 will read data from slave 3, master 2 will read data from slave 4. In this case master 1 has higher priority than master 2. The OPB master 1 still does not have a response from the OPB slave 3 by the TimeOut cycle, but slave device 3 asserts its ToutSup Signal. The OPB slave suppresses OPB_timeout from the arbiter, and master 1 waits for a response from the slave, after which OPB transactions proceed normally.

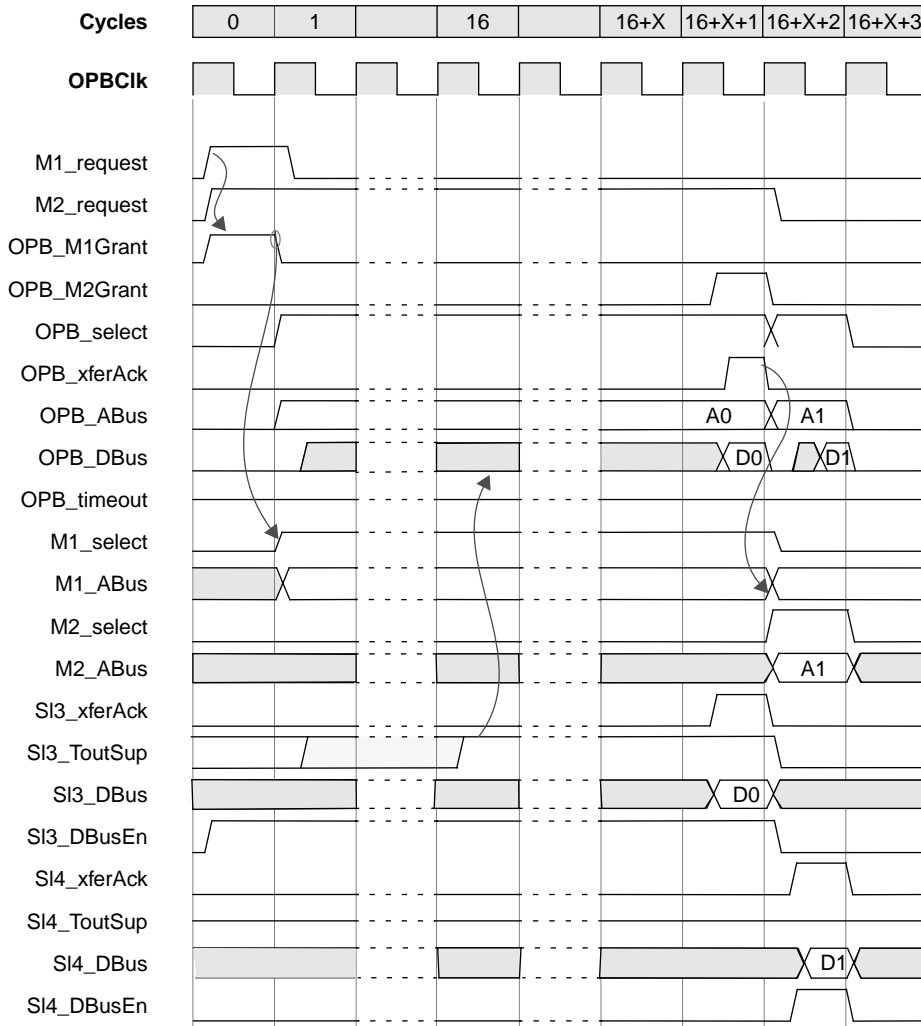


Figure 3-21. Timeout Error Suppression

3.4.5 Dynamic Bus Sizing

Dynamic bus sizing is a feature of the on-chip peripheral bus that permits devices having different data bus widths to inter-operate efficiently. Dynamic bus sizing is performed on a transfer by transfer basis. Bus sizing operation is controlled by the following four signals on the bus:

- Fullword transfer
- Halfword transfer
- Fullword acknowledge
- Halfword acknowledge

When the OPB master performs an operation that is wider than the selected OPB slave, the master must split the operation into two or more smaller transfers. Table 3-7 summarizes the operation of the bus when performing transfers between OPB masters and slaves of varying widths.

Table 3-7. Summary of Dynamic Bus Sizing

Addr (30, 31)	Transfer Size	Bus By 0	Bus By 1	Bus By 2	Bus By 3	Acknowledge
(0, 0)	Byte	B0				By, Fw, Hw Ack
(0, 1)	Byte	B1	B1			By, Fw, Hw Ack
(1, 0)	Byte	B2		B2		By, Fw, Hw Ack
(1, 1)	Byte	B3	B3		B3	By, Fw, Hw Ack
(0, 0)	Halfword	B0	B1			Fw, Hw Ack
(0, 0) (0, 1)	Halfword Byte *	B0 B1	B1 B1			By Ack By Ack
(1, 0)	Halfword	B2	B3	B2	B3	Fw, Hw Ack
(1, 0) (1, 1)	Halfword Byte *	B2 B3	B3 B3	B2	B3 B3	By Ack By Ack
(0, 0)	Fullword	B0	B1	B2	B3	Fw, Ack
(0, 0) (0, 1) (1, 0) (1, 1)	Fullword Byte * Halfword Byte *	B0 B1 B2 B3	B1 B1 B3 B3	B2 B3	B3 B3 B3	By Ack By Ack By Ack By Ack
(0, 0) (1, 0)	Fullword Halfword *	B0 B2	B1 B3	B2 B2	B3 B3	By Ack By Ack

Note 1: Transfer size is indicated by the fullword transfer and halfword transfer signals. Fullword transfers are indicated by both being asserted. Halfword transfers are

indicated by only halfword transfer being asserted. Byte transfers are indicated by neither being asserted. The condition of fullword transfer asserted and halfword transfer not asserted is reserved.

- Note 2:** The bus byte used portion of this table shows which byte within a fullword is being transferred on which byte of the data bus. Blank boxes in this section of the chart indicate “don’t care” portions of the data bus.
- Note 3:** Slave type is indicated by the fullword acknowledge and the halfword acknowledge signals. Fullword slaves assert fullword acknowledge, halfword slaves assert halfword acknowledge, and byte slaves assert neither signal. The condition of fullword acknowledge and halfword acknowledge simultaneously asserted is reserved. It is permissible for a particular OPB slave to function as different types of slaves on different transfers.
- Note 4:** Transfers indicated by a * are generated by the OPB master in order to transfer an operand that is wider than the OPB slave being addressed.

3.4.5.1 Data Alignment

Figure 3-22 shows attachment of bus devices of varying widths. All devices are attached "Left Justified" to the bus, that is, byte devices are attached to bits 00 thru 07, halfword devices are attached to bit 00 thru 16, and fullword devices are attached to all 32 bits.

Byte, halfword, and fullword transfers to a fullword device are transferred memory aligned using all 32 bits of the data bus. Byte and halfword transfers to a halfword device are transferred memory aligned using the high order 16 bits of the data bus. Byte transfers to byte devices are transferred using the high order byte of the bus. When a transfer is requested which is wider than the device, the OPB uses dynamic bus sizing to transfer the data at the device width.

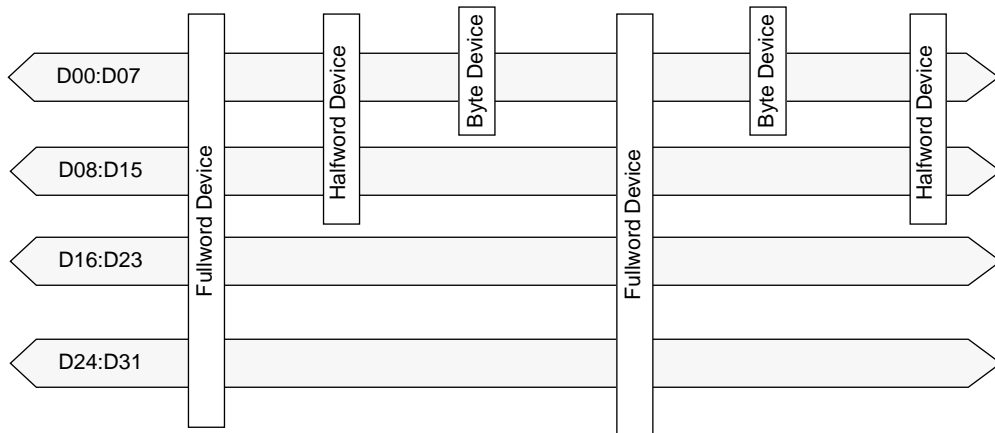


Figure 3-22. Attachment Of Bus Devices Of Varying Width

3.4.5.2 Data Transfer with Dynamic Bus Sizing

The sections that follow demonstrate representative combinations of device widths which give rise to data transfer cycles utilizing dynamic bus sizing.

3.4.5.2.1 Fullword - Halfword Read and Write Operation

Figure 3-23 shows fullword - halfword read and write operation. Master 1 which is fullword device will read and write data to slave 3. Slave 3 is halfword device and has one cycle latency. Note that in cycle 7, master device 1 will drive the data bus with the second halfword. It will be replicated across both halfword positions, so the full 32 bit data bus will be driven with valid data.

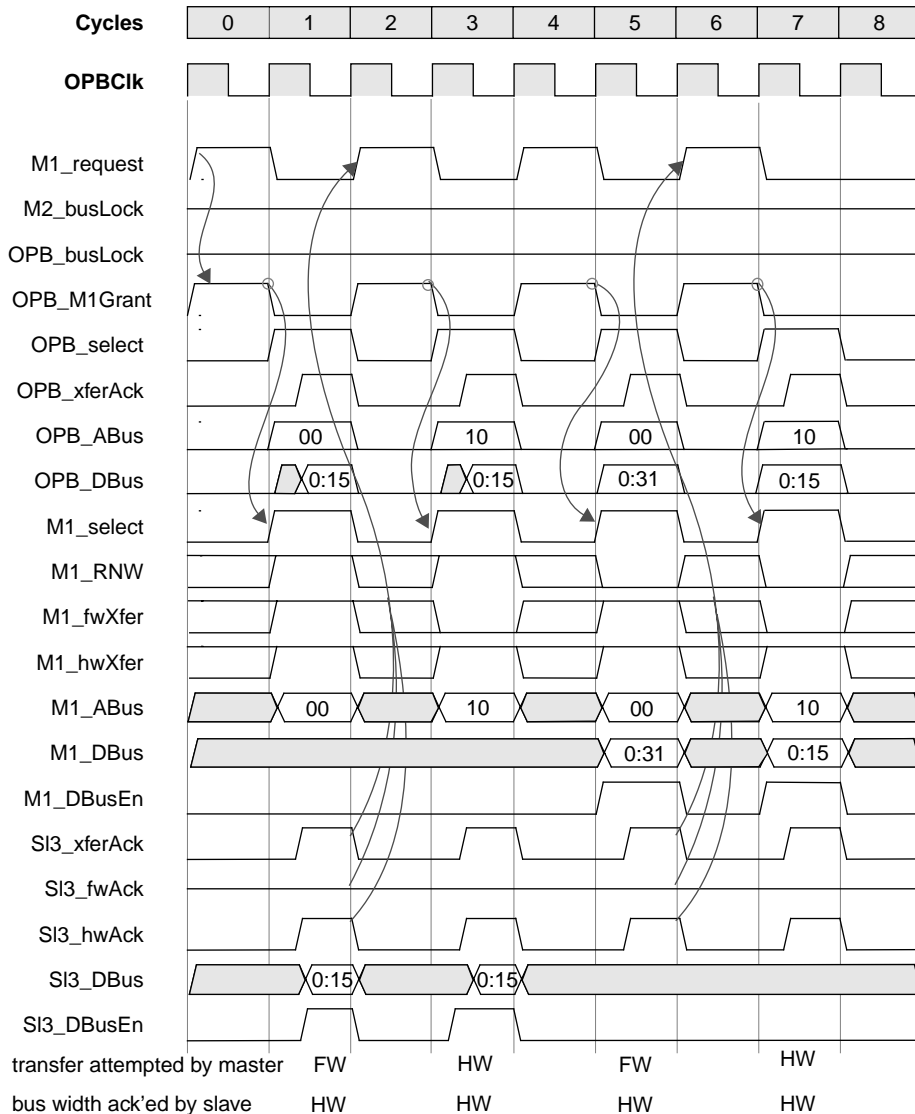


Figure 3-23. Fullword - Halfword Read and Write Operation

3.4.5.2.2 Fullword - Byte Read Operation

Figure 3-24 shows fullword - byte read operation. Master 1 is fullword device, slave 2 is byte device with 1 cycle latency. Master 1 requests to read fullword data from slave 2. Master 1 generates the three additional cycles (3-5) to comply with the dynamic bus sizing requirement. Note that each of these transfers is separately arbitrated, and it is possible that the sequence could be interrupted by other, higher priority, master devices.

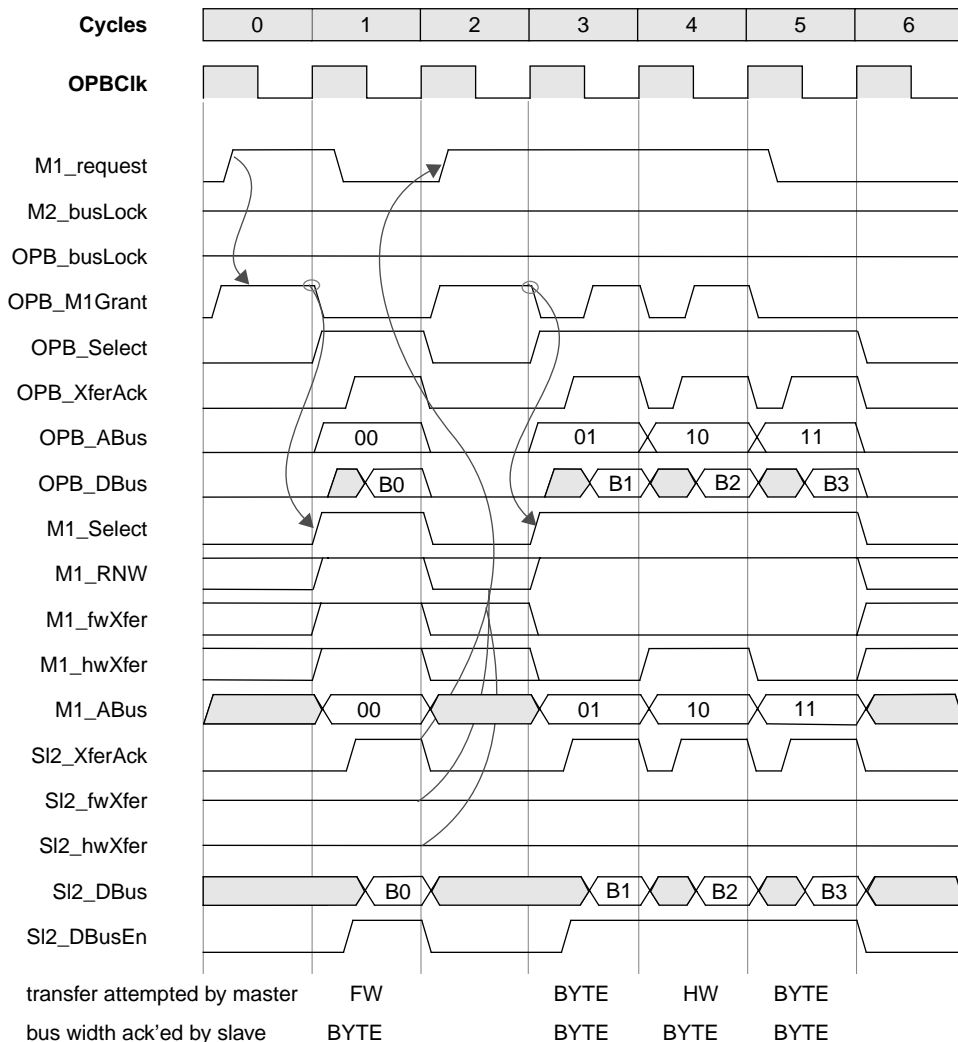


Figure 3-24. Fullword - Byte Read Operation

3.4.5.2.3 Dynamic Bus Sizing and Overlapped Arbitration

Figure 3-25 shows a representative sequence of OPB data transfers, where one master performs dynamic bus sizing and the other does not. Arbitration is overlapped between the two masters. In this case, OPB master 1 will read and write data to OPB slave 3 and OPB master 2 will read data from OPB slave 4. OPB master 1 and 2, and slave 3, are fullword devices and slave 4 is a halfword device.

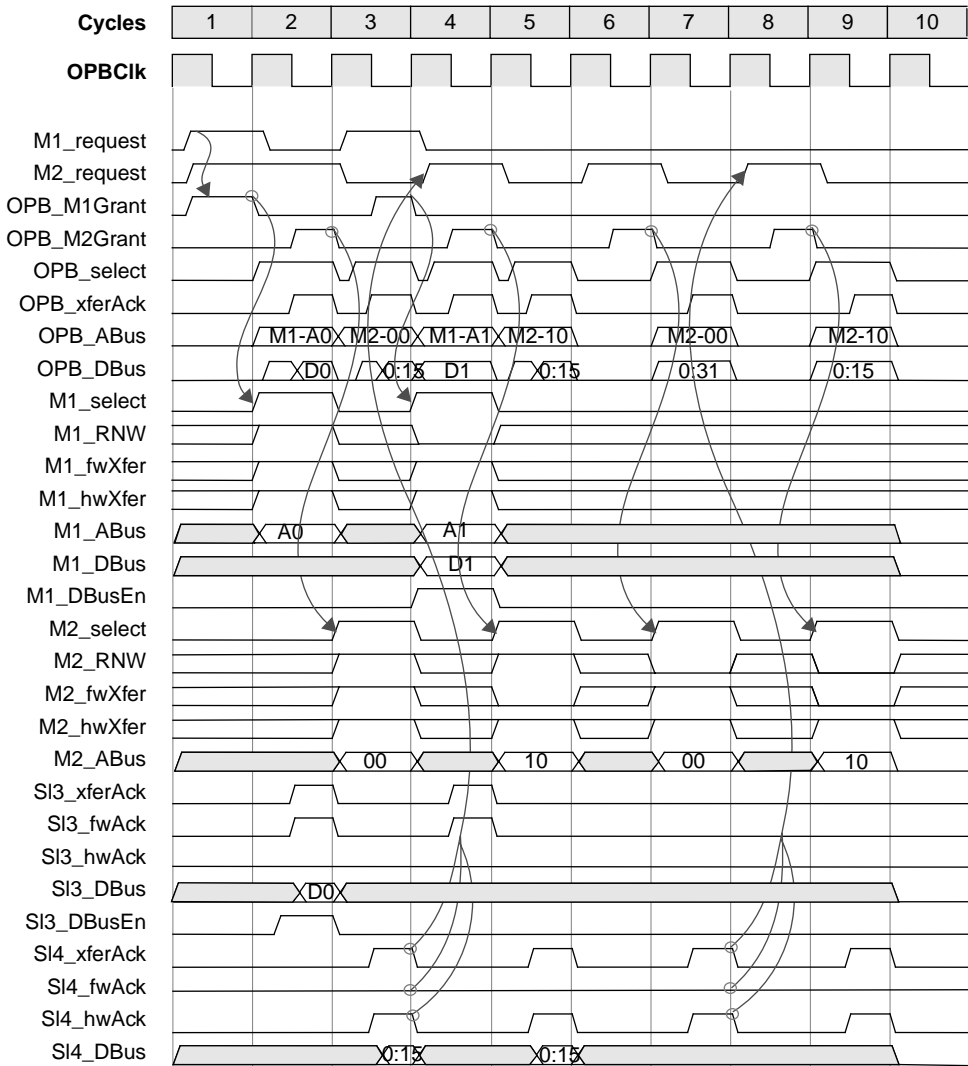


Figure 3-25. Overlapped Arbitration

3.4.5.2.4 Locked Dynamic Bus Sizing With Interruption

Figure 3-26 shows a case where a master asserts busLock to lock the bus for the duration of the dynamic bus sizing sequence. The data transfer, including dynamic bus sizing, proceeds as in the previous case. However, the master asserted busLock. This precludes another master (M2) of higher priority from interrupting the sequence in cycle 3. Master 1 deasserts busLock in cycle 5, the last of its data transfer cycles. This allows the OPB to arbitrate in cycle 5, granting the bus to master 2, which assumes ownership of the bus by asserting M2_select in cycle 6 (the full details of master 2's data transfer omitted).

Even using busLock in the manner described below, a master device cannot guarantee an uninterrupted dynamic bus sizing operation. If another, higher priority, master asserts a request during the first access (a valid arbitration cycle), it could obtain (and lock) the bus. The first master device will not know the bus width of the addressed slave until it receives acknowledge, and thus only then will it realize that additional data transfer cycles are necessary.

OPB masters that require uninterrupted data transfer, including dynamic bus sizing cycles, may achieve this by asserting busLock for every cycle, even for nominally single-cycle full-word and halfword transfers. This approach defeats the overlapped arbitration capability of the OPB, and reduces overall bandwidth, as it requires idle cycles on the bus for arbitration for those cases where slaves are able to respond to the transfer size requested.

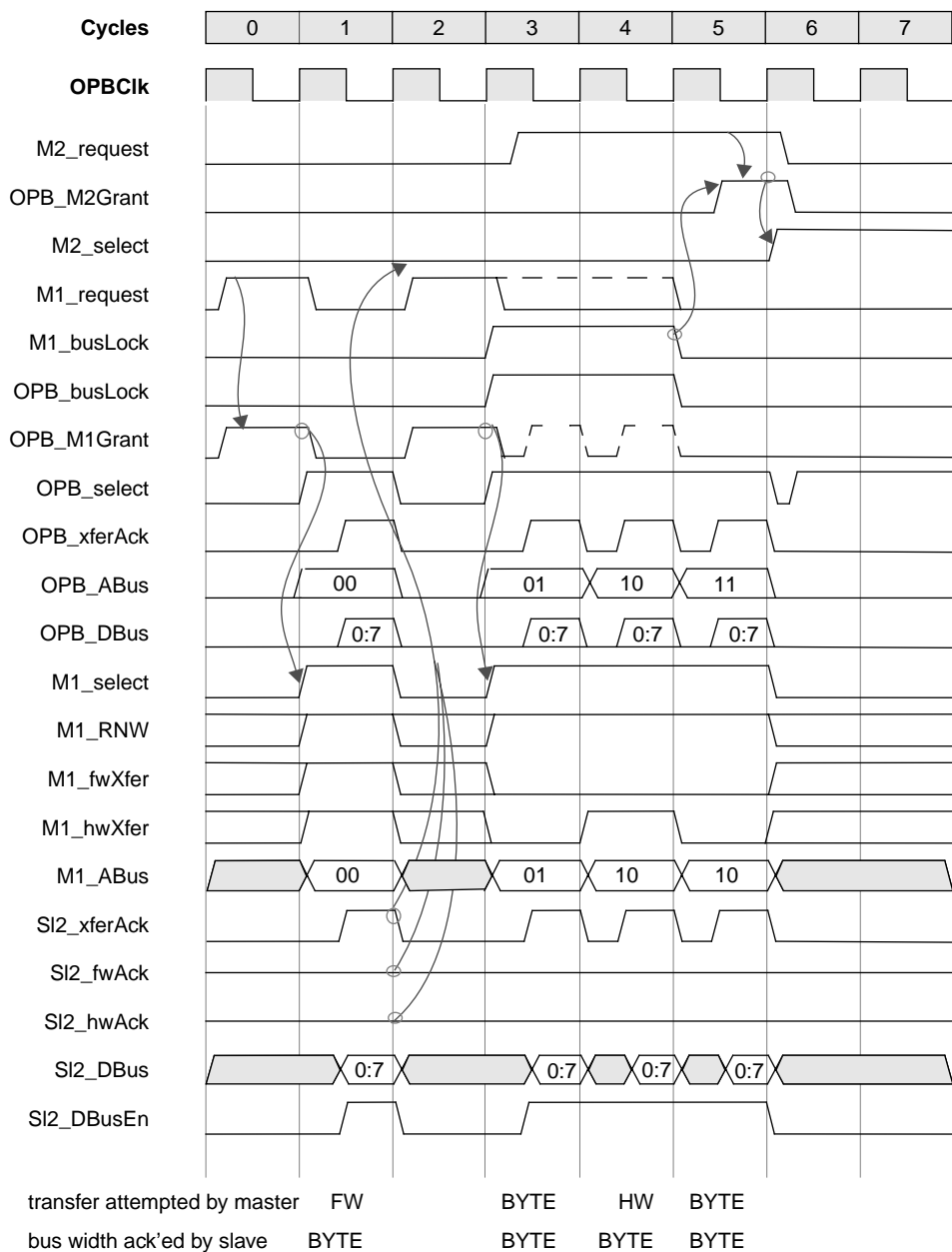


Figure 3-26. Dynamic Bus Sizing with BusLock Signal

3.4.5.2.5 Locked Dynamic Bus Sizing With No Interruption

Figure 3-27 shows locked dynamic bus sizing with no interruption. In this case, master 1 and 2 are fullword devices, slave 3 is halfword device and slave 4 is fullword device. Both slave devices have 1 cycle latency. Master 1 requests to read data from slave 3 and master 2 requests to read data from slave 4.

In this case, the data transfers at cycles 2 and 6 are generated by master 1 for dynamic bus sizing (fullword reads from halfword slave device). When dynamic bus sizing is required, this procedure is effective, as it provides dynamic bus sizing without interruption and the following data transfer can take advantage of overlapped bus arbitration.

However, if dynamic bus sizing is not required, this procedure requires a dedicated arbitration cycle for the next data transfer. This is the case for master 2, which asserted busLock, but did not require dynamic bus sizing cycles. The OPB arbiter generates a grant (if master 2 is requesting) at cycles 3 and 7. Whether or not master 2 requests and receives grants in these cycles, the fact that it has locked the bus prevents arbitration cycles there. The OPB arbiter thus uses cycles 4 and 8 for bus arbitration.

The system designer implementing master devices on the OPB will need to consider which approach to dynamic bus sizing is appropriate for each application.

Figure 3-28 and Figure 3-29 demonstrate representative data transfer cycles utilizing dynamic bus sizing, where BusLock is asserted immediately (i.e., the master device is “aware” of the bus width of the addressed slaves, anticipates the need for dynamic bus sizing, and wishes to complete the entire data transfer without interruption).

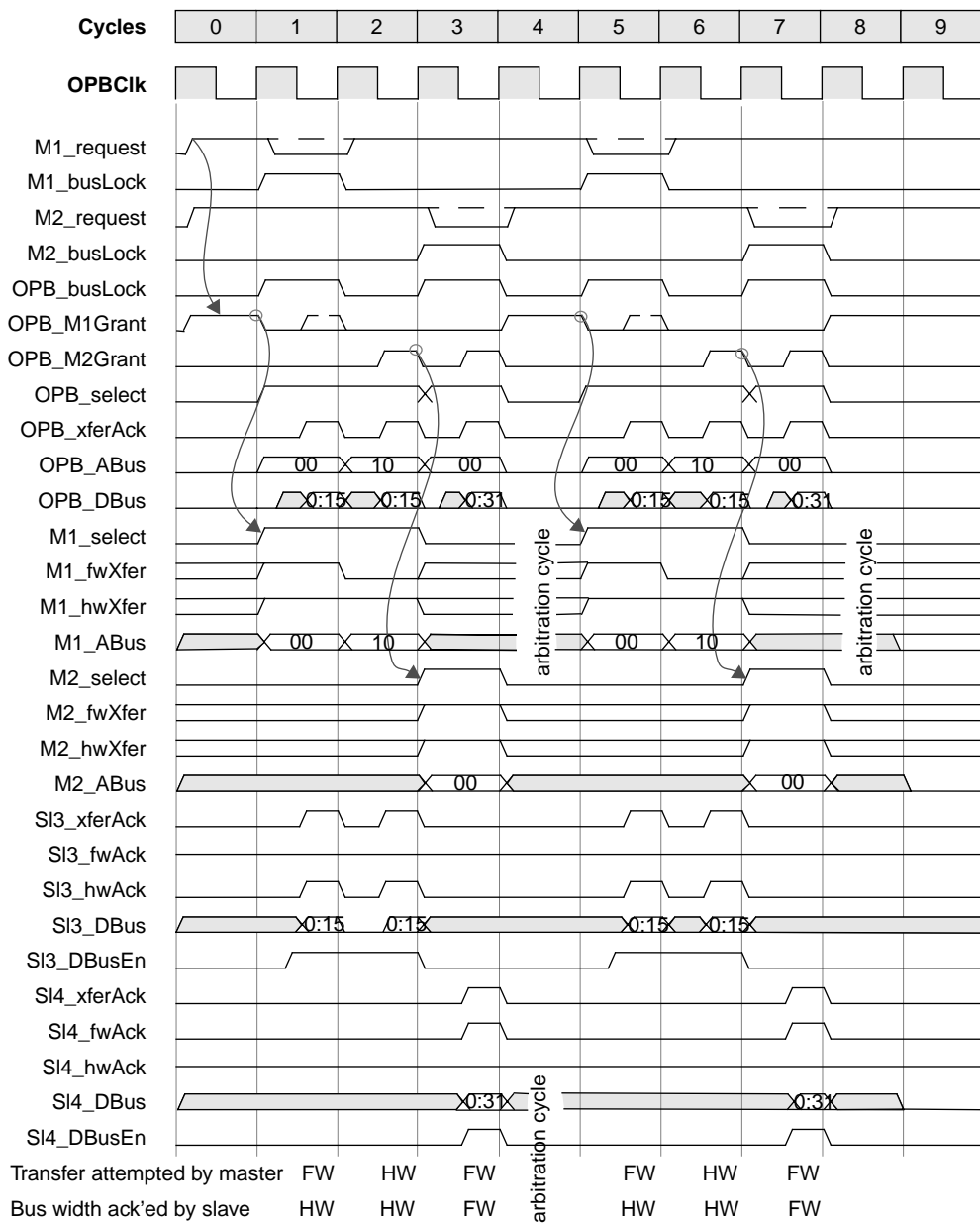


Figure 3-27. Dynamic Bus Sizing Without Interruption

3.4.5.2.6 Fullword - Byte Read and Write Operation

Figure 3-28 shows fullword - byte read and write operation with BusLock. Master 1 which is a fullword device will read and write data to slave 2. Slave 2 is a byte device with one cycle latency.

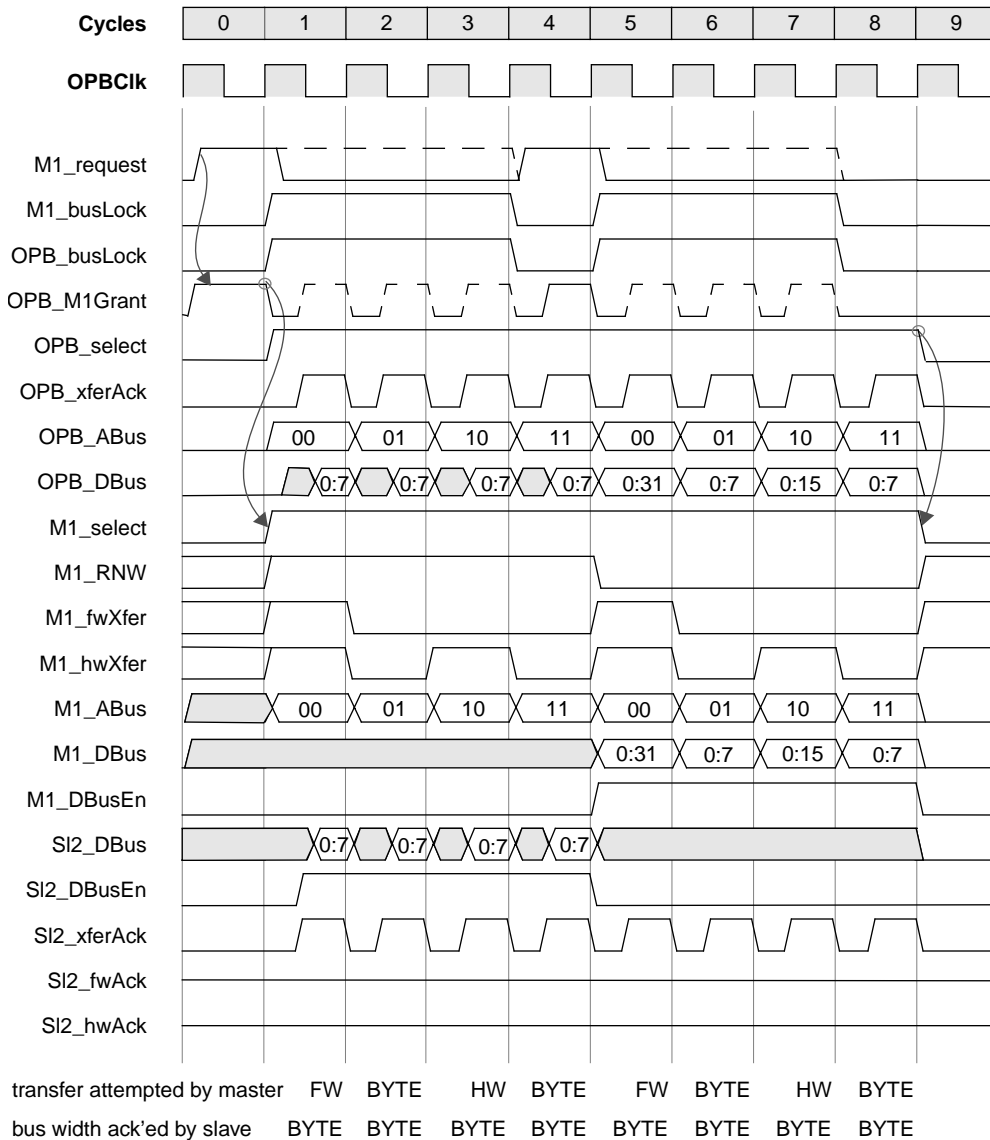


Figure 3-28. Fullword - Byte, Read and Write Operation

3.4.5.2.7 Halword - Byte, Read and Write Operation

Figure 3-29 shows halfword - byte read and write operation with BusLock. Master 1 which is a halfword device will read and write data to slave 2. Slave 2 is a byte device with one cycle latency. The master device anticipates the bus width of the slave, and asserts BusLock immediately for guaranteed uninterrupted access to the bus for the duration of the data transfer.

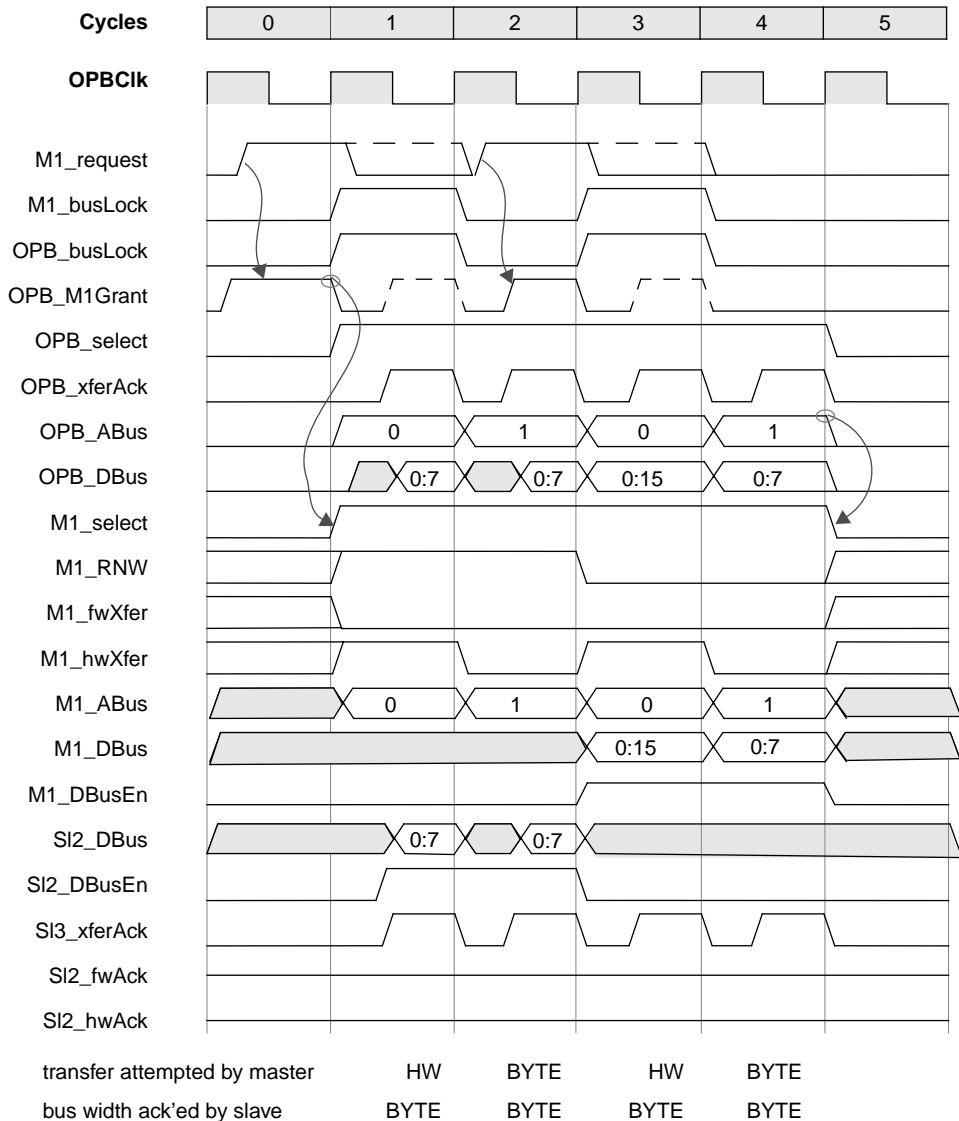


Figure 3-29. Halfword - Byte, Read and Write Operation

3.4.6 Optional DMA Peripheral Cycle

Figure 3-30 shows optional DMA peripheral read cycle. In the following example, OPB device acts as DMA peripheral device. OPB device only outputs data to OPB_DBus when the DMA acknowledge signal is asserted.

Here, master 0 is assumed to be the OPB bridge, which interfaces the OPB to the PLB or some other system resource for communication with the DMA controller. This aspect of DMA operation is specific to the system architecture, and is beyond the scope of this document. See OPB bridge user's manual for more information. The OPB architecture merely allows for the utilization of the OPB by DMA peripherals under direct control of the DMA, and using the OPB data bus.

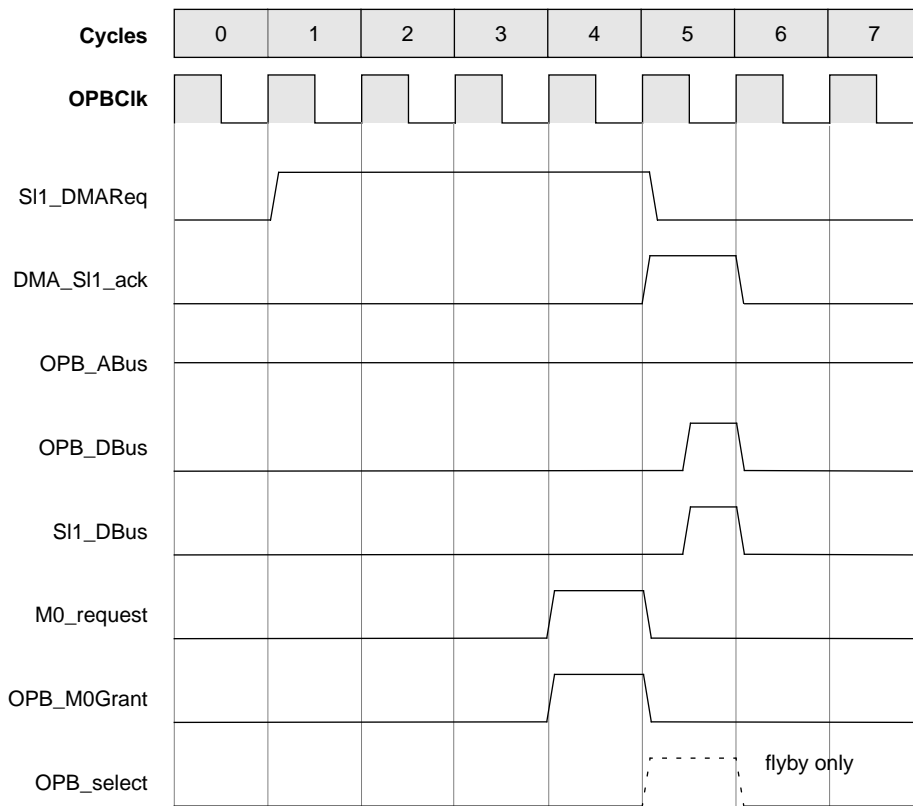


Figure 3-30. Optional DMA Peripheral Cycle

Chapter 4. Device Control Register Bus

The device control register (DCR) bus is designed to transfer data between the CPU's general purpose registers (GPRs) and the DCR slave logic's device control registers (DCRs). The DCR bus removes configuration registers from memory address, reduces loading and improves bandwidth of the processor local bus. This document is organized as follows:

- DCR Overview
- DCR Signals
- DCR Interfaces
- DCR Operations

4.1 DCR Overview

The DCR bus is a fully synchronous bus. Therefore, it is assumed that in Core+ASIC environments where the CPU and the DCR slave logic are running at different clock speeds, the slower clock's rising edge always corresponds to a faster clock's rising edge. The DCR bus is typically implemented as a distributed mux across the chip such that each sub-unit not only has a path to place its own DCRs on the CPU's DCR read path, but also has a path which bypasses its DCRs and places another unit's DCRs on the CPU's DCR read path.

The DCRs are on-chip registers that exist architecturally outside the processor core. They are accessed by using the move to device control register (**mtdcr**) and move from device control register (**mfdcr**) instructions. They may control the use of on-chip peripherals, such as memory controllers, etc. All DCRs are privileged for both read and write.

Features of the device control register bus include:

- 10-bit address bus and 32-bit data bus
- 2-cycle minimum read or write transfers extendable by slave or master
- Handshake supports clocked asynchronous transfers
- Slaves may be clocked either faster or slower than master
- Single device control register bus master
- Distributed multiplexer architecture
- A simple but flexible interface

Figure 4-1 provides a block diagram of the device control register bus.

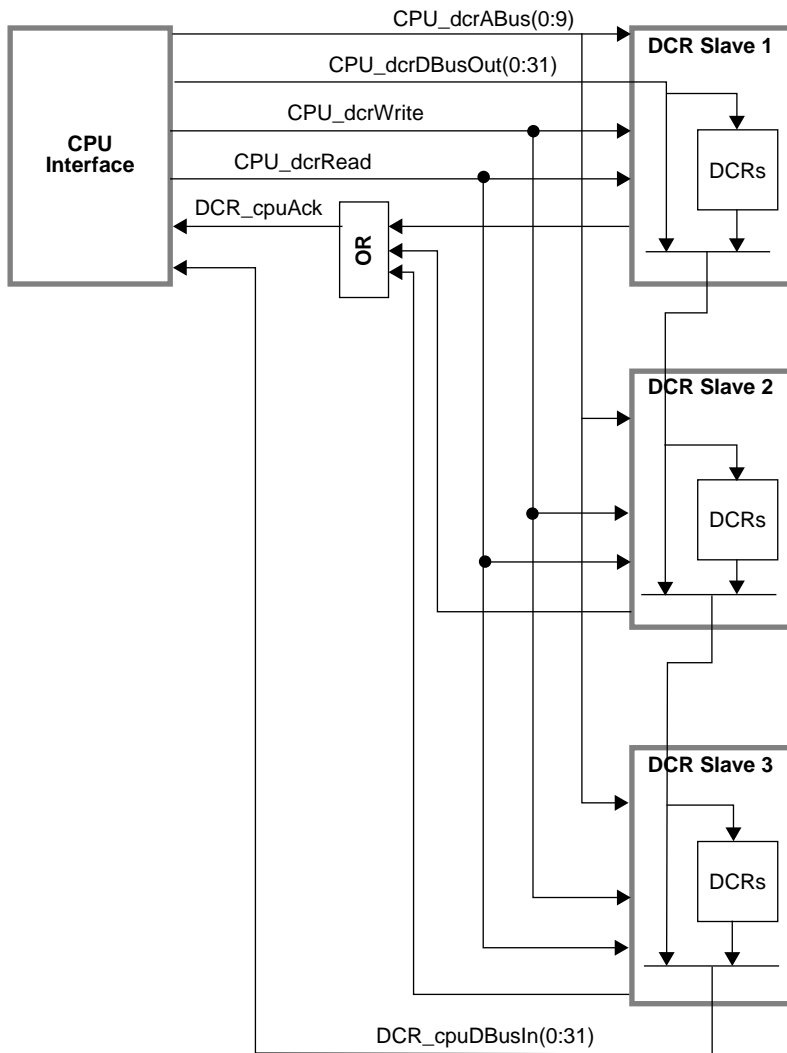


Figure 4-1. DCR Block Diagram

4.1.1 DCR Implementation

Figure 4-2 illustrates a logical implementation of the DCR bus interface. To ensure its reusability across several Core+ASIC environment, it is recommended that all DCR slave logic containing DCRs adhere to the implementation specified here. This implementation allows the DCR slave to run at different clock speeds than the processor core. The bypass mux should be implemented to keep the path delay to a minimum when not selected. This function can be implemented as a 2-1 mux or equivalent logic.

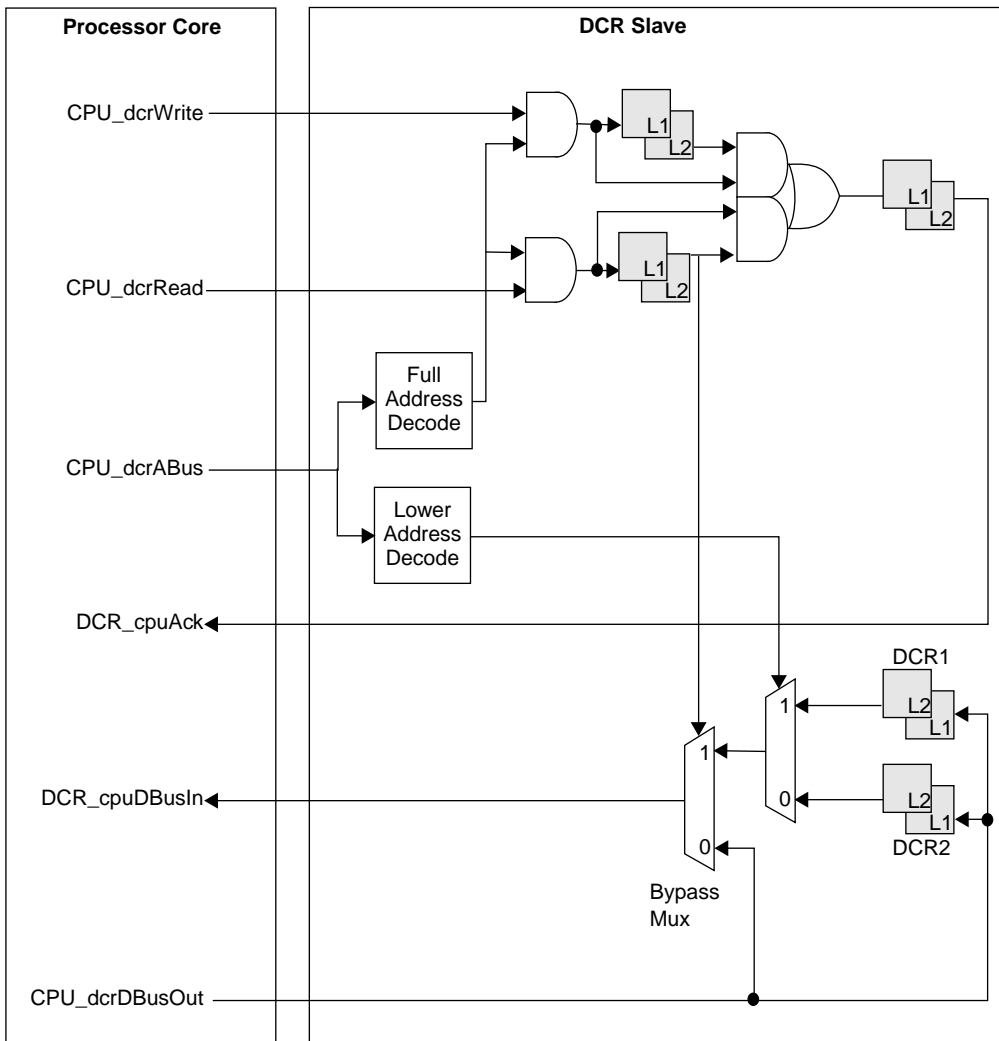


Figure 4-2. DCR Bus Implementation

Note: In the processor core, latches (not shown) are clocked by the CPU's clock while in the DCR bus, latches (shaded) are clocked by the DCR slave's clock.

In this implementation the address from the CPU, CPU_dcrABus, is qualified with CPU_dcrWrite and CPU_dcrRead and then latched, bringing these signals into the DCR slave clock domain. They are also routed around these latches to ensure that the DCR_cpuAck signal can be deactivated as soon as possible.

Additional logic can also be added to delay the assertion of DCR_cpuAck to allow extra time to access the DCR. If DCR_cpuAck is not asserted within 16 cycles, the CPU will time-out and simply execute the next instruction.

4.2 DCR Signals

Table 4-1 provides a summary of all DCR signals in alphabetical order, the interfaces under which they are grouped, followed by a brief description and page reference for detailed functional description.

Table 4-1. Summary of DCR Signals

Signal Name	Interface	I/O	Description	Page
CPU_dcrWrite	CPU	I	CPU DCR write	4-5
CPU_dcrRead	CPU	I	CPU DCR read	4-5
CPU_dcrABus(0:9)	CPU	I	CPU DCR address bus	4-5
CPU_dcrDBusOut(0:31)	CPU	I	CPU DCR data bus out	4-6
DCR_cpuDBusIn(0:31)	CPU	O	DCR CPU data bus in	4-6
DCR_cpuAck	CPU	O	DCR CPU acknowledge	4-6

4.2.1 CPU_dcrWrite (CPU DCR Write)

This is the control signal for a DCR write operation. This signal is asserted by the CPU to indicate that a move to device control register (mtdcr) operation is in progress and that the address on CPU_dcrABus(0:9) is valid. Once this signal is asserted, the CPU will wait a maximum of 16 cycles for the DCR_cpuAck signal to be asserted indicating that the mtdcr operation is complete. If DCR_cpuAck is not asserted within 16 cycles, the CPU will time-out and simply execute the next instruction. During reset this signal is held inactive.

4.2.2 CPU_dcrRead (CPU DCR Read)

This is the control signal for a DCR read operation. This signal is asserted by the CPU to indicate that a move from device control register (mfdcr) operation is in progress and that the address on CPU_dcrABus(0:9) is valid. Once this signal is asserted, the CPU will wait a maximum of 16 cycles for the DCR_cpuAck signal to be asserted indicating that the mfdcr operation is complete. If DCR_cpuAck is not asserted within 16 cycles, the CPU will time-out and simply execute the next instruction. During reset this signal is held inactive.

4.2.3 CPU_dcrABus(0:9) (CPU DCR Address Bus)

This is the DCR address bus. This bus is valid anytime CPU_dcrWrite, or CPU_dcrRead is asserted, and it is invalid at all other times. Hence, the DCR slave logic must always qualify this bus with CPU_dcrWrite/CPU_dcrRead. The DCR address will remain stable during the mtdcr/mfdcr operation.

4.2.4 CPU_dcrDBusOut(0:31) (CPU DCR Data Bus Out)

This is the CPU DCR write data bus. This bus is driven by the CPU with valid DCR data during a mtdcr operation and is valid anytime the CPU_dcrWrite signal is asserted. During reset, this bus is driven with zeroes by the CPU and may be used by the DCR slave logic to initialize its DCRs.

Note: From the DCR slave logic's perspective, this bus may be driven directly by the CPU or another's unit bypass, as illustrated in Figure 4-1 on page 4-2.

4.2.5 DCR_cpuAck (DCR CPU Acknowledge)

This is the DCR transfer acknowledge signal. This signal indicates that the DCR slave logic has recognized CPU_dcrWrite/CPU_dcrRead and the address on CPU_dcrABus(0:9). This signal must be asserted, once the DCR has been updated during a mtdcr operation, or once data has been placed on DCR_cpuDBusIn(0:31) during a mfdcr operation. DCR_cpuAck should be kept asserted as long as CPU_dcrWrite or CPU_dcrRead is asserted.

Furthermore, DCR_cpuAck should be negated in the cycle following the negation of CPU_dcrWrite or CPU_dcrRead.

Note: The CPU will wait for DCR_cpuAck to be negated for the current mtdcr/mfdcr operation, prior to starting a new mtdcr/mfdcr operation.

4.2.6 DCR_cpuDBusIn(0:31) (DCR CPU Data Bus In)

This is the CPU DCR read data bus. The DCR slave logic will drive this bus with the contents of its selected DCR during a mfdcr operation as long as the address on CPU_dcrAddr(0:9) matches the DCR's address and CPU_dcrRead is asserted. Data must be valid when DCR_cpuAck is asserted. When the DCR slave DCR is not selected and DCR_cpuAck is not active, the DCR slave logic will bypass its DCR and drive this bus with CPU_dcrDBusOut(0:31).

4.3 DCR Interfaces

The DCR I/O signals are grouped under the following interface categories depending on their function. For detailed functional description of the signals see Section 4.2, “DCR Signals,” on page 4-5.

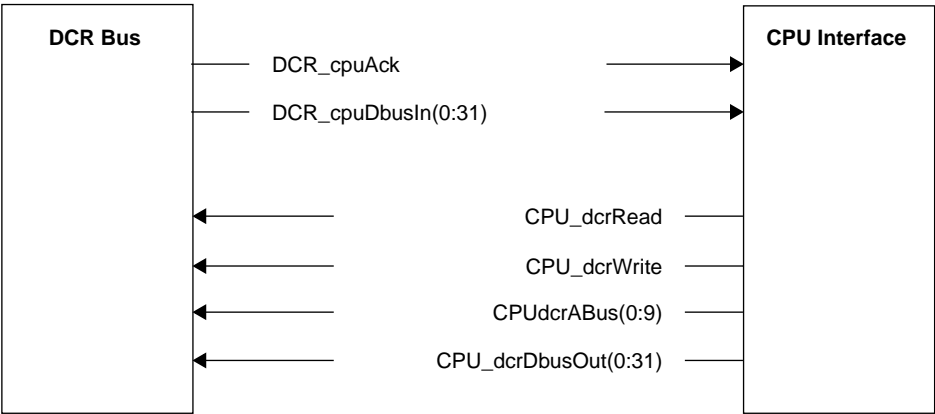


Figure 4-3. CPU Interface

4.4 DCR Operations

The DCR interface comprises an address bus, an input and output data bus, DCR write and DCR read signals, and an acknowledge signal.

The acknowledge is to indicate that a move-to operation is complete or that the move-from data is on the bus. It is interlocked with the move-to or move-from signal such that the transfer will take a minimum of three CPU cycles. This interlock mechanism allows the DCR interface to be connected to peripheral units that are clocked at different frequencies than the core. The rising edge of the slower clock (CPU or peripheral) must always correspond to the faster clock's rising edge. This implies that the clocks for the ASIC (containing the DCR) and the clocks for the core are derived from a common source. That common source can be multiplied or divided before being sent to the core or ASIC.

Address bus and outbound control signals are sent to each unit that requires a DCR interface. Data moves over an interface that forms a ring network. Each unit either passes along the unmodified data bus input, or, if it is the addressed unit being addressed, places its data on the data bus (**mfdcr** only).

DCR_cpuAck can be distributed in a similar way by logically ORing the acknowledge signal from a DCR unit with the acknowledge signal from the preceding DCR. Should this be timing prohibitive, all the acknowledge signals may be ORed together and connected to DCR_cpuAck. This is called combinatorial acknowledge.

The DCR logic must return DCR_cpuAck within 16 CPU cycles; otherwise the processor times out. No error occurs on a DCR operation time out. The CPU simply executes the next instruction.

This section on DCR operations basically describes the recommended bus loading and implementation. It also provides an example of the DCR bus implementation in 401 Core.

4.4.1 Recommended Bus Loading

- **CPU_dcrWrite**

Loading should be kept to a maximum of 2 standard loads. This is a multi-drop net and it is expected that the DCR Slave will buffer the signal before fanning out to internal logic.

- **CPU_dcrRead**

Loading should be kept to a maximum of 2 standard loads. This is a multi-drop net and it is expected that the DCR Slave will buffer the signal before fanning out to internal logic.

- **CPU_dcrABus(0:9)**

Loading should be kept to a maximum of 2 standard loads. This is a multi-drop net and it is expected that the DCR Slave will buffer the signals before fanning out to internal logic.

- **CPU_dcrDBusOut(0:31)**

Loading should be kept to a maximum of 5 standard loads. This bus is driven from the CPU or the previous DCR Slave in the DCR chain.

- **DCR_cpuAck**

This signal should be able to drive a minimum of 3 standard loads.

- **DCR_cpuDBusIn(0:31)**

This bus should be able to drive a minimum of 10 standard loads. If a DCR chain becomes long it may be broken up into more than one loop.

4.4.2 Recommended Implementation

The timing diagrams in this section provide examples of how the DCR bus interface operates in the following situation as demonstrated in Figure 4-2:

- CPU Same Speed as DCR Slave
- CPU Same Speed as DCR Slave with Longer DCR Access
- CPU Two Times Faster than DCR Slave
- CPU Slower than DCR Slave

4.4.2.1 CPU Same Speed as DCR Slave

Figure 4-4 describes the DCR bus in operation when the CPU clock is running at the same speed as DCR slave.

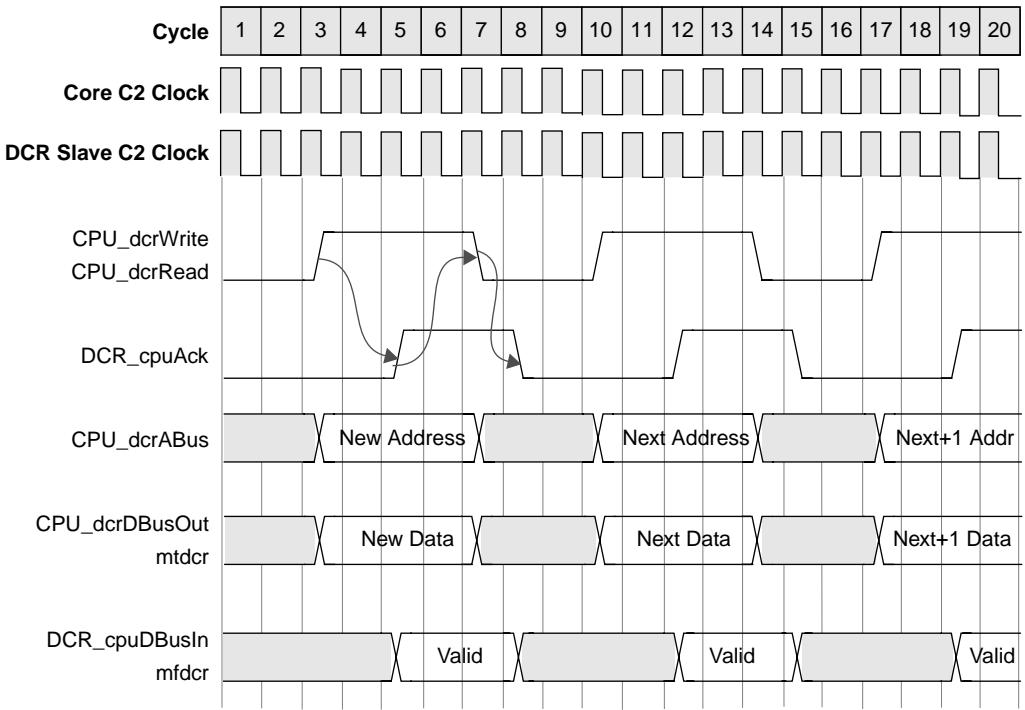


Figure 4-4. CPU Same Speed as DCR Slave

4.4.2.2 CPU Same Speed as DCR Slave with Longer DCR Access

The DCR slave can take longer to access the DCR data before responding with DCR_cpuAck. If DCR_cpuAck is not asserted within 16 cycles, the CPU will time-out and simply execute the next instruction.

Figure 4-5 describes DCR bus in operation when the CPU clock is running at the same speed as DCR slave with longer DCR access.

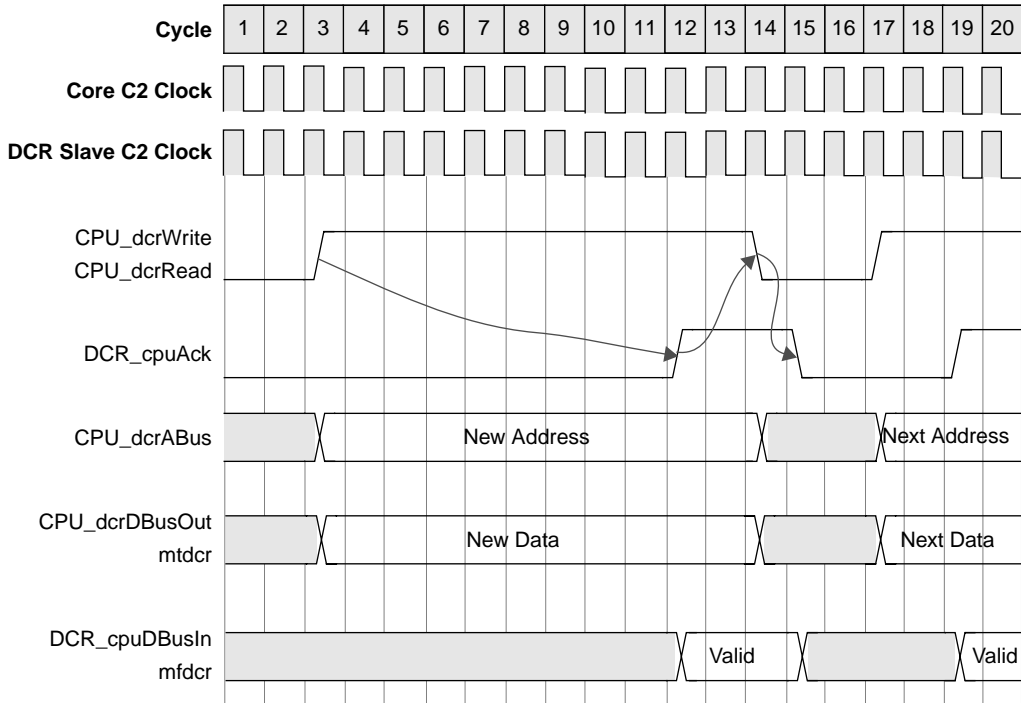


Figure 4-5. CPU Same Speed as DCR Slave with Longer DCR Access

4.4.2.3 CPU Two Times Faster than DCR Slave

The signals from the CPU are relative to the CPU clock and the signals from the DCR slave are relative to the DCR slave clock.

Figure 4-6 describes DCR bus in operation when the CPU clock is running two times faster than the DCR slave.

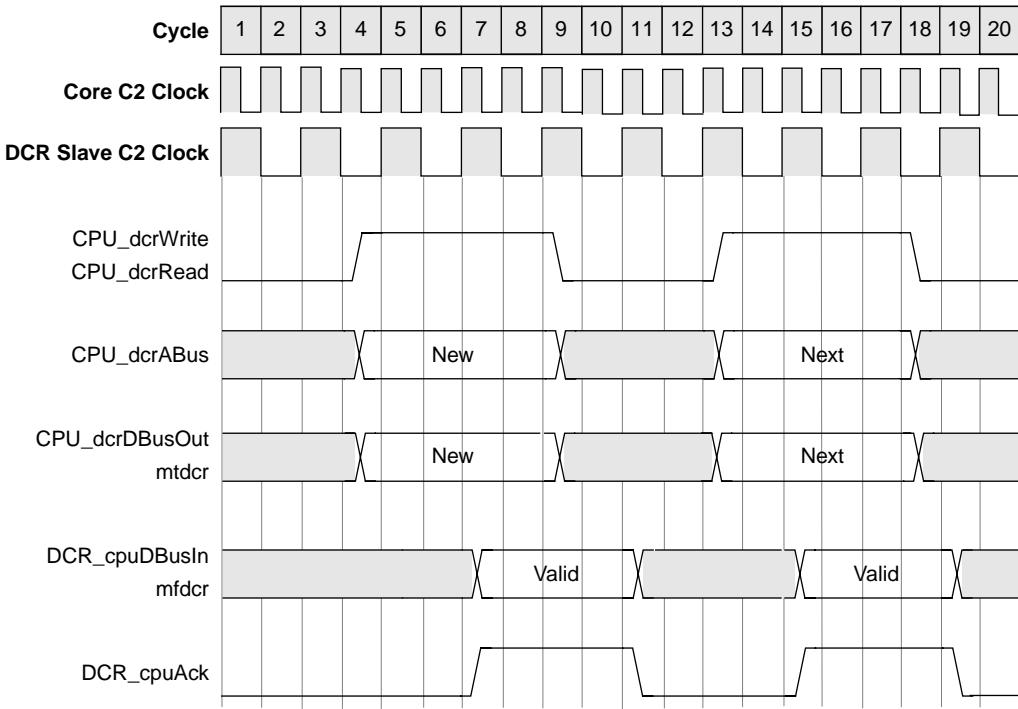


Figure 4-6. CPU Two Times Faster than DCR Slave

4.4.2.4 CPU Slower than DCR Slave

Figure 4-7 describes the dcr in operation when the CPU clock is running slower than the DCR slave. In fact, the DCR slave is running two times faster than the CPU clock.

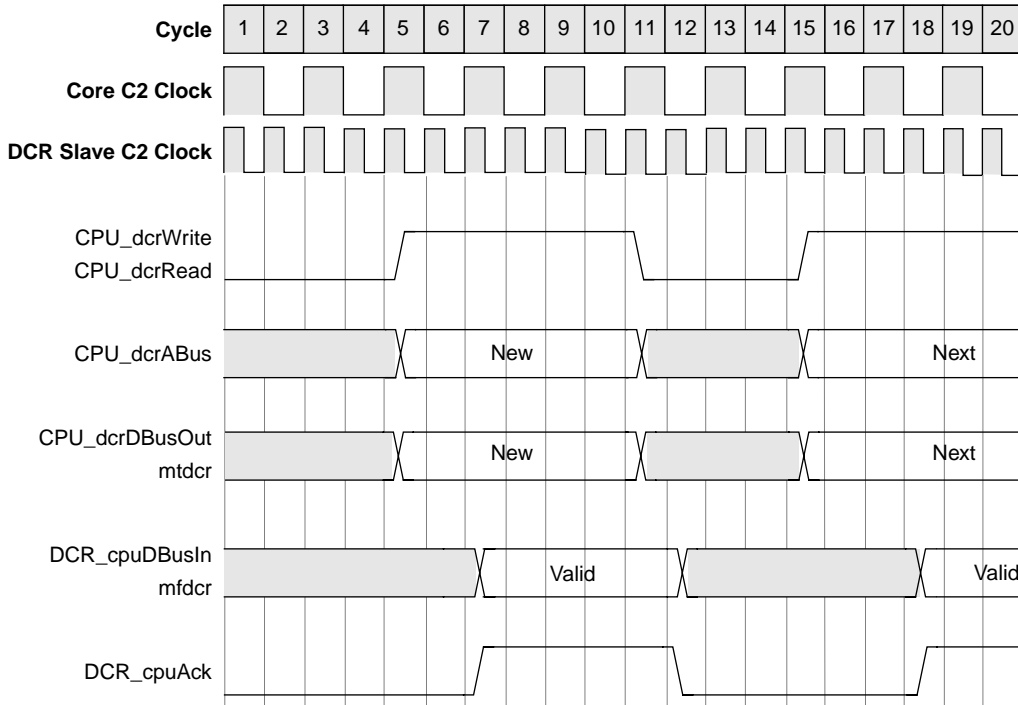


Figure 4-7. CPU Slower than DCR Slave

4.4.3 DCR Implementation in 401 Core

This section demonstrates how the device control register bus is implemented in 401 Core. The timing diagrams in this section provide examples of how the DCR bus interface operates in the following situation:

- CPU Same Speed as DCR Slave
- CPU Same Speed as DCR Slave With Longer DCR Access
- CPU Two Times Faster Than DCR Slave
- CPU Slower than DCR Slave

4.4.3.1 CPU Same Speed as DCR Slave

The address can not be used unless CPU_dcrWrite/CPU_dcrRead is valid in the next cycle. Once CPU_dcrWrite/CPU_dcrRead is active the **mtdcr/mfdcr** instruction will complete.

If data is captured at the end of cycle 4 during writes this gives 2 1/2 cycles of setup time for address and data. During reads this can give 2 cycles of setup time for data to the CPU if data is driven at the beginning of cycle 4. The setup time for the address would be 1 1/2 cycles.

Macros can design an optional mode to take advantage of the extra cycle of address and data setup time before CPU_dcrWrite/CPU_dcrRead is valid. The macro should also be designed to function without this feature. If this can not be offered as an optional mode then the macro must assume that the address and data are valid in the same cycle when CPU_dcrWrite/CPU_dcrRead is valid. DCR_cpuDBusIn can be driven once CPU_dcrRead is asserted and the DCR slave's DCR is selected. The data does not need to be valid until DCR_cpuAck is asserted by the DCR slave.

The first **mtdcr/mfdcr** instruction takes 6 cycles to execute. If mtdcr/mfdcr instructions are back-to-back the PPC401 core will add an extra dead cycle between instructions. Therefore the second mtdcr/mfdcr instruction will take 7 cycles.

Figure 4-8 describes CPU same speed as DCR slave.

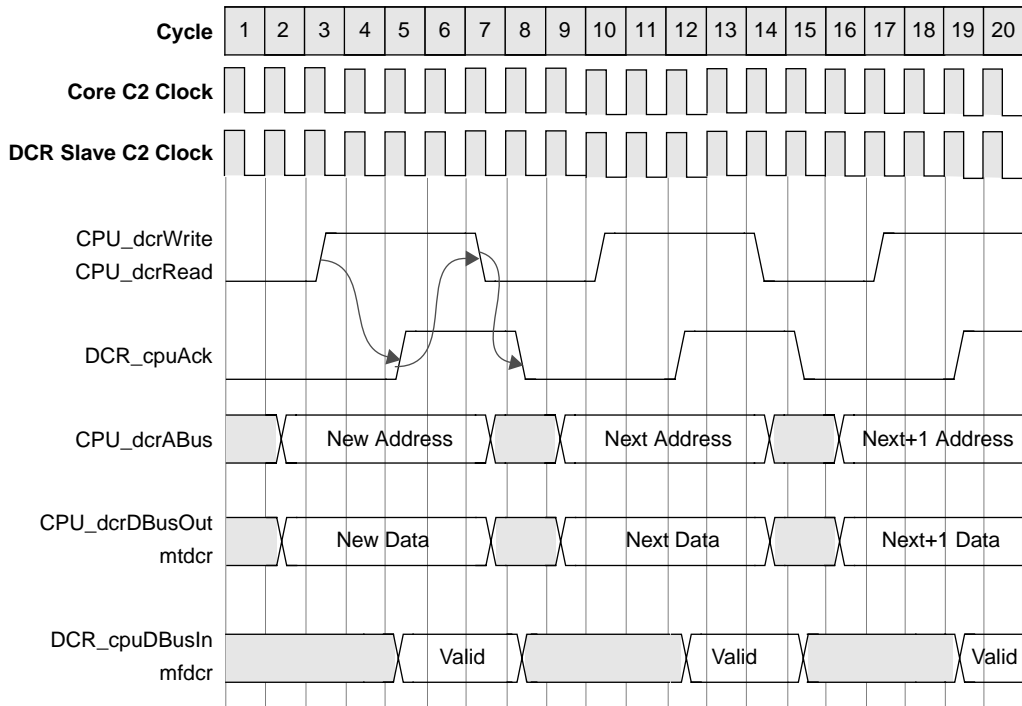


Figure 4-8. CPU Same Speed As DCR Slave

4.4.3.2 CPU Same Speed as DCR Slave With Longer DCR Access

The DCR slave can take longer to access the DCR data before responding with DCR_cpuAck. If DCR_cpuAck is not asserted within 16 cycles, the CPU will time-out and simply execute the next instruction.

Figure 4-9 describes CPU same speed as DCR slave with longer DCR access.

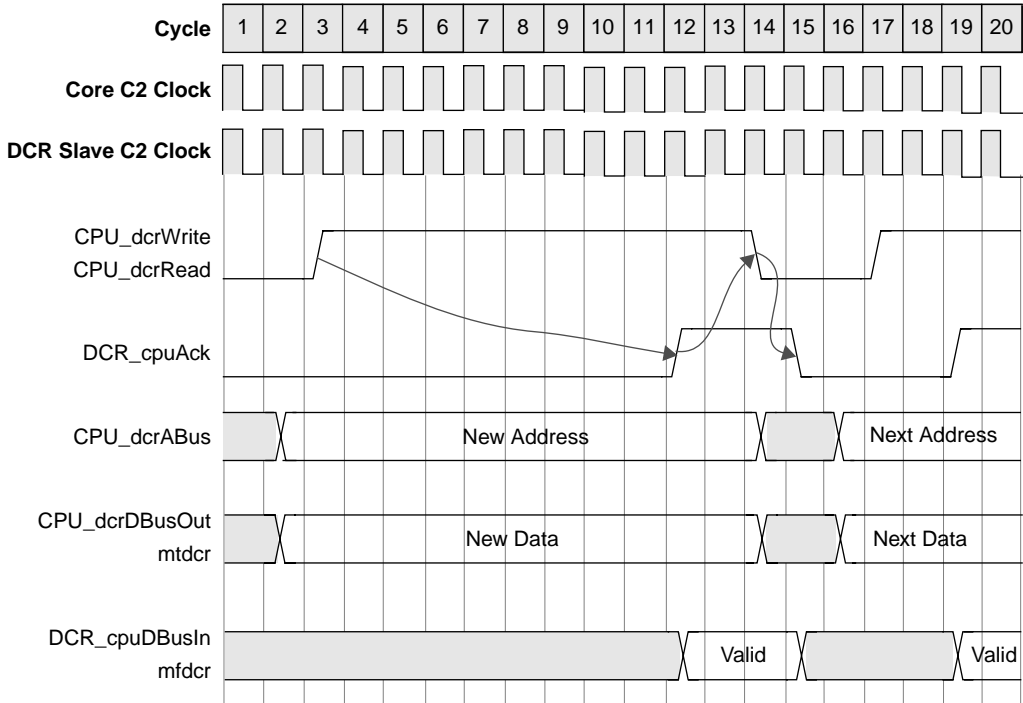


Figure 4-9. CPU Same Speed As DCR Slave With Longer DCR Access

4.4.3.3 CPU Two Times Faster Than DCR Slave

The signals from the CPU are relative to the CPU clock and the signals from the DCR slave are relative to the DCR slave clock.

Figure 4-10 describes CPU 2X faster than the DCR slave.

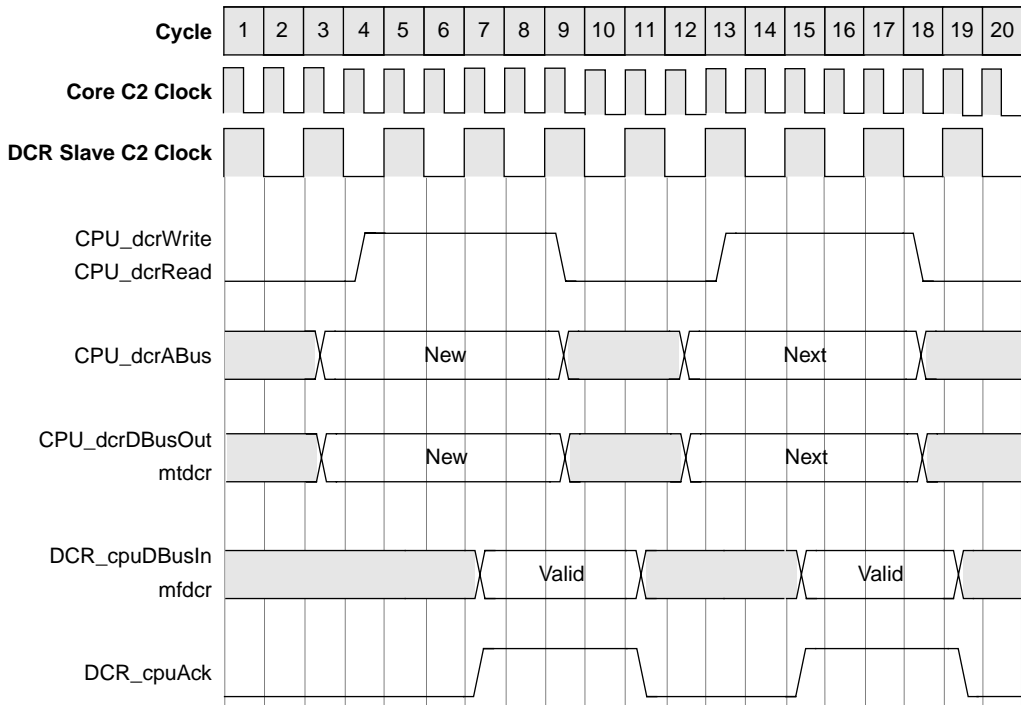


Figure 4-10. CPU 2X Faster Than DCR Slave

4.4.3.4 CPU Slower than DCR Slave

Figure 4-11 describes CPU slower than the DCR slave.

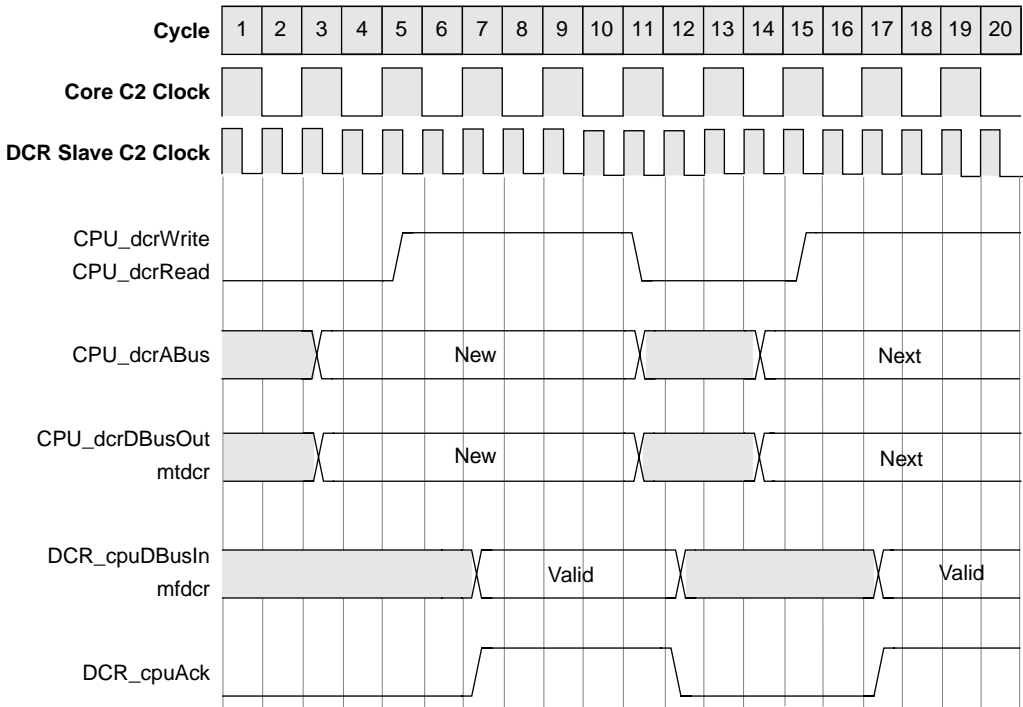


Figure 4-11. CPU Slower Than DCR Slave

Chapter 5. Timing Guidelines

5.1 PLB Timing Guidelines

The PLB signal timing guidelines described in this section are based on single clock cycle address and data transfers across the PLB bus. These guidelines are provided in an attempt to maximize bus performance and promote the reusability of PLB masters and slaves at various frequencies and technologies. Actual input set-up times and output delays for PLB bus, PLB master, and PLB slave macros may vary from these guidelines. Early timing analysis should be performed on all Core+ASIC chips using the PLB bus macro and the associated PLB masters and slaves at the chip level to ensure that the timing objectives of the application can be met. See individual macro user manuals for details.

Begin	Signal is valid within 8% of the clock cycle from the rise of the Sys_plbClk signal.
Early	Signal is valid within 18% of the clock cycle from the rise of the Sys_plbClk signal.
Early +	Signal is valid within 28% of the clock cycle from the rise of the Sys_plbClk signal.
Middle -	Signal is valid within 33% of the clock cycle from the rise of the Sys_plbClk signal.
Middle	Signal is valid within 43% of the clock cycle from the rise of the Sys_plbClk signal.
Middle +	Signal is valid within 53% of the clock cycle from the rise of the Sys_plbClk signal.
Late -	Signal is valid within 58% of the clock cycle from the rise of the Sys_plbClk signal.
Late	Signal is valid within 68% of the clock cycle from the rise of the Sys_plbClk signal.
End	Signal is valid within 78% of the clock cycle from the rise of the Sys_plbClk signal.

Note: These definitions assume that there is 0ns of clock delay. For outputs, these delays represent the total logic delay from the C2 clock at the input to a register to the output of the macro. For inputs, these delays represent the arrival time of the input relative to a 0ns delayed clock.

5.1.1 PLB Master Timing Guidelines

Table 5-1 describes PLB master signal timing guidelines.

Table 5-1. PLB Master Signal Timing Guidelines

PLB Signal Name	Driven By	Output Valid	Received by
Mn_request	PLB master n	Begin	PLB arbiter
Mn_priority	PLB master n	Begin	PLB arbiter
Mn_RNW	PLB master n	Begin	PLB arbiter
Mn_busLock	PLB master n	Early	PLB arbiter
Mn_BE	PLB master n	Early	PLB arbiter
Mn_size	PLB master n	Early	PLB arbiter
Mn_type	PLB master n	Early	PLB arbiter
Mn_compress	PLB master n	Early	PLB arbiter
Mn_guarded	PLB master n	Early	PLB arbiter
Mn_ordered	PLB master n	Early	PLB arbiter
Mn_lockErr	PLB master n	Early	PLB arbiter
Mn_ABus	PLB master n	Early	PLB arbiter
Mn_DBus	PLB master n	Early	PLB arbiter
Mn_wrBurst	PLB master n	Early	PLB arbiter
Mn_rdBurst	PLB master n	Early	PLB arbiter
Mn_abort	PLB master n	Late -	PLB arbiter

5.1.2 PLB Arbiter Timing Guidelines

Table 5-2 describes PLB arbiter signal timing guidelines.

Table 5-2. PLB Arbiter Signal Timing Guidelines

PLB Signal Name	Driven By	Output Valid	Received by
PLB_MnBusy	PLB arbiter	Early +	PLB master n
PLB_MnErr	PLB arbiter	Early +	PLB master n
PLB_MnRdDBus	PLB arbiter	Early +	PLB master n

Table 5-2. PLB Arbiter Signal Timing Guidelines (cont.)

PLB Signal Name	Driven By	Output Valid	Received by
PLB_MnRdWdAddr	PLB arbiter	Early +	PLB master n
PLB_MnRdDAck	PLB arbiter	Early +	PLB master n
PLB_rdBurst	PLB arbiter	Early +	Slaves
PLB_pendReq	PLB arbiter	Early	Slaves
PLB_pendPri	PLB arbiter	Early +	Slaves
PLB_busLock	PLB arbiter	Middle +	Slaves
PLB_reqPri	PLB arbiter	Middle	Slaves
PLB_MasterID	PLB arbiter	Middle	Slaves
PLB_PAVValid	PLB arbiter	Middle	Slaves
PLB_SAVValid	PLB arbiter	Middle	Slaves
PLB_rdPrim	PLB arbiter	Middle	Slaves
PLB_wrPrim	PLB arbiter	End	Slaves
PLB_RNW	PLB arbiter	Middle	Slaves
PLB_BE	PLB arbiter	Middle	Slaves
PLB_size	PLB arbiter	Middle	Slaves
PLB_type	PLB arbiter	Middle	Slaves
PLB_compress	PLB arbiter	Middle	Slaves
PLB_guarded	PLB arbiter	Middle	Slaves
PLB_ordered	PLB arbiter	Middle	Slaves
PLB_lockErr	PLB arbiter	Middle	Slaves
PLB_ABus	PLB arbiter	Middle	Slaves
PLB_wrDBus	PLB arbiter	Middle +	Slaves
PLB_wrBurst	PLB arbiter	Middle	Slaves
PLB_MnWrBTerm	PLB arbiter	Late	Slaves
PLB_MnRdBTerm	PLB arbiter	Middle	Slaves
PLB_MnAddrAck	PLB arbiter	Late	PLB master n

Table 5-2. PLB Arbiter Signal Timing Guidelines (cont.)

PLB Signal Name	Driven By	Output Valid	Received by
PLB_MnRearbitrate	PLB arbiter	Late	PLB master n
PLB_MnWrDAck	PLB arbiter	Late	PLB master n
PLB_abort	PLB arbiter	Late	Slaves

5.1.3 PLB Slave Timing Guidelines

Table 5-3 describes PLB slave signal timing guidelines.

Table 5-3. PLB Slave Signal Timing Guidelines

PLB Signal Name	Driven By	Output Valid	Received by
SI_MBusy	Slaves	Early	PLB arbiter
SI_MErr	Slaves	Early	PLB arbiter
SI_rdwdAddr	Slaves	Early	PLB arbiter
SI_rdDAck	Slaves	Early	PLB arbiter
SI_rdComp	Slaves	Early	PLB arbiter
SI_rdBUS	Slaves	Early	PLB arbiter
SI_rdBTerm	Slaves	Middle -	PLB arbiter
SI_wrBTerm	Slaves	Late -	PLB arbiter
SI_wrDAck	Slaves	Late -	PLB arbiter
SI_wrComp	Slaves	Late	PLB arbiter
SI_addrAck	Slaves	Late -	PLB arbiter
SI_wait	Slaves	Late -	PLB arbiter
SI_rearbitrate	Slaves	Late -	PLB arbiter

5.2 OPB Timing Guidelines

The timing diagrams included in this specification are examples of operations on the OPB. All signals on the OPB are positive active and are either direct outputs of edge triggered latches which are clocked by the OPB clock, or are derived from the output of a register using several levels of combinatorial logic. All input signals should be captured in the OPB masters or OPB slaves on the rising edge of the OPB clock.

- Begin** Signal is valid within 8% of the clock cycle from the rise of the OPB clock signal.
- Early** Signal is valid within 18% of the clock cycle from the rise of the OPB clock signal.
- Early +** Signal is valid within 28% of the clock cycle from the rise of the OPB clock signal.
- Middle -** Signal is valid within 33% of the clock cycle from the rise of the OPB clock signal.
- Middle** Signal is valid within 43% of the clock cycle from the rise of the OPB clock signal.
- Middle +** Signal is valid within 53% of the clock cycle from the rise of the OPB clock signal.
- Late -** Signal is valid within 58% of the clock cycle from the rise of the OPB clock signal.
- Late** Signal is valid within 68% of the clock cycle from the rise of the OPB clock signal.
- End** Signal is valid within 78% of the clock cycle from the rise of the OPB clock signal.

Note: These definitions assume that there is 0ns of clock delay. For outputs, these delays represent the total logic delay from the C2 clock at the input to a register to the output of the macro. For inputs, these delays represent the arrival time of the input relative to a 0ns delayed clock.

Set-up and Hold times for the OPB inputs and output delays for the OPB outputs are dependent on the technology and the physical implementation of the bus. These parameters are specified as a percentage of the bus clock cycle relative to the rise of the OPB clock.

Table 5-4 describes the OPB timing guidelines

Table 5-4. OPB Timing Guidelines

Signal Name	Driven By	Output Valid	Received By
Mn_request	Master n	Begin	OPB arbiter
Mn_busLock	Master n	Begin	OPB bus
OPB_MnGrant	OPB arbiter	Late	Master n
OPB_timeout	OPB arbiter	Middle	All Masters
Mn_Select	Master n	Begin	OPB bus
Mn_RNW	Master n	Begin	OPB bus
Mn_hwXfer	Master n	Begin	OPB bus

Table 5-4. OPB Timing Guidelines (cont.)

Signal Name	Driven By	Output Valid	Received By
Mn_fwXfer	Master n	Begin	OPB bus
Mn_seqAddr	Master n	Begin	OPB bus
Mn_ABus	Master n	Begin	OPB bus
Mn_DBus	Master n	Early	OPB bus
Mn_DBusEn	Master n	Early	N/A
OPB_Select	OPB bus	Early	All slaves and OPB arbiter
OPB_RNW	OPB bus	Early	All slaves
OPB_hwXfer	OPB bus	Early	All slaves
OPB_fwXfer	OPB bus	Early	All slaves
OPB_seqAddr	OPB bus	Early	All slaves
OPB_ABus	OPB bus	Early	All slaves
OPB_DBus	OPB bus	Late	All masters and slaves
OPB_xferAck	OPB bus	Middle +	All masters and OPB arbiter
OPB_hwAck	OPB bus	Middle +	All masters
OPB_fwAck	OPB bus	Middle +	All masters
OPB_errAck	OPB bus	Middle +	All masters
OPB_retry	OPB bus	Middle +	All masters and OPB arbiter
OPB_ToutSup	OPB bus	Early +	OPB arbiter
Sln_xferAck	Slave	Middle	OPB bus
Sln_hwAck	Slave	Middle	OPB bus
Sln_fwAck	Slave	Middle	OPB bus
Sln_errAck	Slave	Middle	OPB bus
Sln_retry	Slave	Middle	OPB bus
Sln_ToutSup	Slave	Early	OPB bus
Sln_DBus	Slave	Middle	OPB bus
Sln_DBusEn	Slave	Middle	N/A

5.3 DCR Timing Guidelines

The DCR signal timing guidelines described in this section are based on single clock cycle address and data transfers across the DCR bus.

Begin Signal is valid within 8% of the clock cycle from the rise of the SysClk signal.

Early Signal is valid within 18% of the clock cycle from the rise of the SysClk signal.

Early + Signal is valid within 28% of the clock cycle from the rise of the SysClk signal.

Middle - Signal is valid within 33% of the clock cycle from the rise of the SysClk signal.

Middle Signal is valid within 43% of the clock cycle from the rise of the SysClk signal.

Middle + Signal is valid within 53% of the clock cycle from the rise of the SysClk signal.

Late - Signal is valid within 58% of the clock cycle from the rise of the SysClk signal.

Late Signal is valid within 68% of the clock cycle from the rise of the SysClk signal.

End Signal is valid within 78% of the clock cycle from the rise of the SysClk signal.

Note: These definitions assume that there is 0ns of clock delay. For outputs, these delays represent the total logic delay from the C2 clock at the input to a register to the output of the macro. For inputs, these delays represent the arrival time of the input relative to a 0ns delayed clock.

Table 5-5 describes the DCR timing guidelines

Table 5-5. DCR Timing Guidelines

Signal Name	Driven By	Output Valid	Received By
CPU_dcrABus(0:9)	CPU	Early	DCR
CPU_dcrDBusOut(0:31) (Write)	CPU	Early	DCR
CPU_dcrRead	CPU	Early	DCR
CPU_dcrWrite	CPU	Early	DCR
DCR_cpuAck	DCR	Late	CPU
DCR_cpuDBusIn(0:31) (Read)	DCR	Late	CPU
DCR_cpuDBusIn(0:31) (Bypass)	DCR		CPU

Chapter 6. Signal Summary

This chapter provides a summary of all signals discussed in this document.

Table 6-1. Signal Summary Table

Signal Name	Interface	I/O	Description	Page
Processor Local Bus				
Mn_request	Master n	I	Master n bus request.	2-11
Mn_abort	Master n	I	Master n abort bus request indicator.	2-18
Mn_priority(0:1)	Master n	I	Master n bus request priority.	2-11
Mn_busLock ¹	Master n	I	Master n bus lock.	2-12
Mn_RNW	Master n	I	Master n read/write.	2-19
Mn_BE(0:3)	Master n	I	Master n byte enables	2-20
Mn_size(0:3)	Master n	I	Master n transfer size	2-22
Mn_type(0:2)	Master n	I	Master n transfer type	2-23
Mn_compress	Master n	I	Master n compressed data transfer indicator	2-24
Mn_guarded	Master n	I	Master n guarded transfer indicator	2-24
Mn_ordered	Master n	I	Master n synchronize transfer indicator	2-25
Mn_lockErr	Master n	I	Master n lock error indicator	2-25
Mn_ABus(0:31)	Master n	I	Master n address bus	2-26
Mn_wrDBus(0:31)	Master n	I	Master n write data bus	2-27
Mn_wrBurst	Master n	I	Master n burst write transfer indicator	2-28
Mn_rdBurst	Master n	I	Master n burst read transfer indicator	2-32
PLB_MnAddrAck	Master n	O	PLB master n address acknowledge	2-17
PLB_MnRearbitrate	Master n	O	PLB master n bus rearbitrate indicator	2-17
PLB_Mn_Busy	Master n	O	PLB master n slave busy indicator	2-34
PLB_Mn_Err	Master n	O	PLB master n slave error indicator	2-34

Table 6-1. Signal Summary Table (cont.)

Signal Name	Interface	I/O	Description	Page
PLB_Mn_WrDAck	Master n	O	PLB master n write data acknowledge	2-28
PLB_Mn_WrBTerm	Master n	O	PLB master n terminate write burst indicator	2-29
PLB_MnRdDBus(0:31)	Master n	O	PLB master n read data bus	2-30
PLB_MnRdWdAddr(0:3)	Master n	O	PLB master n read word address	2-31
PLB_MnRdDAck	Master n	O	PLB master n read data acknowledge	2-31
PLB_MnRdBTerm	Master n	O	PLB master n terminate read burst indicator	2-33
PLB_masterID(0:1)	Arbiter	O	PLB current master identifier	2-19
PLB_PAValiid	Arbiter	O	PLB primary address valid indicator	2-12
PLB_SAValiid	Arbiter	O	PLB secondary address valid indicator	2-15
PLB_pendReq	Arbiter	O	PLB pending bus request indicator	2-18
PLB_abort	Arbiter	O	PLB abort bus request indicator	2-18
PLB_reqPri(0:1)	Arbiter	O	PLB current request priority	2-19
PLB_pendPri(0:1)	Arbiter	O	PLB pending request priority	2-18
PLB_busLock	Arbiter	O	PLB bus lock	2-12
PLB_RNW	Arbiter	O	PLB read not write	2-19
PLB_BE(0:3)	Arbiter	O	PLB byte enables	2-20
PLB_size(0:3)	Arbiter	O	PLB transfer size	2-22
PLB_type(0:2)	Arbiter	O	PLB transfer type	2-23
PLB_compress	Arbiter	I	PLB compressed data transfer indicator	2-24
PLB_guarded	Arbiter	O	PLB guarded transfer indicator	2-24
PLB_ordered	Arbiter	O	PLB synchronize transfer indicator	2-25
PLB_lockErr	Arbiter	O	PLB lock error indicator	2-25
PLB_ABus(0:31)	Arbiter	O	PLB address bus	2-26
PLB_wrDBus(0:31)	Arbiter	O	PLB write data bus	2-27
PLB_wrBurst	Arbiter	O	PLB burst write transfer indicator	2-28

Table 6-1. Signal Summary Table (cont.)

Signal Name	Interface	I/O	Description	Page
PLB_rdBurst	Arbiter	O	PLB burst read transfer indicator	2-32
PLB_wrPrim	Arbiter	O	PLB secondary to primary write request indicator	2-29
PLB_rdPrim	Arbiter	O	PLB secondary to primary read request indicator	2-33
SI_addrAck	Slave	I	Slave address acknowledge	2-17
SI_rearbitrate	Slave	I	Slave rearbitrate bus indicator	2-17
SI_wait	Slave	I	Slave wait indicator	2-16
SI_rdComp	Slave	I	Slave read transfer complete indicator	2-32
SI_rdDAck	Slave	I	Slave read data acknowledge	2-31
SI_rdBTerm	Slave	I	Slave terminate read burst transfer	2-33
SI_rddb(0:31)	Slave	I	Slave read data bus	2-30
SI_rdWdAddr(0:3)	Slave	I	Slave read word address	2-31
SI_wrComp	Slave	I	Slave write transfer complete indicator	2-28
SI_wrDAck	Slave	I	Slave write data acknowledge	2-28
SI_wrBTerm	Slave	I	Slave terminate write burst transfer	2-29
SI_MBusy(0:3)	Slave	I	Slave busy indicator	2-34
SI_MErr(0:3)	Slave	I	Slave error indicator	2-34
Sys_plbClk	System	I	System C2 clock	2-10
Sys_plbReset	System		System PLB reset	2-10
On-Chip Peripheral Bus				
DMA_SlnAck	DMA	I	DMA slave acknowledge	3-13
Mn_request	Master	O	Master bus request	3-8
Mn_busLock	M/A	O	Master bus arbitration lock	3-8
Mn_select	Master	O	Master select	3-10
Mn_RNW	Master	O	Master read not write	3-11
Mn_hwXfer	Master	O	Master halfword transfer	3-11

Table 6-1. Signal Summary Table (cont.)

Signal Name	Interface	I/O	Description	Page
Mn_fwXfer	Master	O	Master fullword transfer	3-11
Mn_seqAddr	Master	O	Master sequential address	3-11
Mn_ABus(0:31)	Master	I	Master address bus	3-10
Mn_DBus(0:31)	Master	O	Master data bus	3-10
Mn_DBusEn	Master	O	Master data bus enable	3-12
OPB_busLock	Arbiter	I	OPB bus arbitration lock	3-8
OPB_MnGrant	M/A	I	OPB master bus grant	3-8
OPB_timeout	M/A	I	OPB timeout error	3-9
OPB_select	S/A	I	OPB select	3-10
OPB_RNW	Slave	I	OPB read not write	3-11
OPB_hwXfer	Slave	I	OPB halfword transfer	3-11
OPB_fwXfer	Slave	I	OPB fullword transfer	3-11
OPB_seqAddr	Slave	I	OPB sequential address	3-11
OPB_ABus(0:31)	M/S	I	OPB address bus	3-10
OPB_DBus(0:31)	M/S	I	OPB data bus	3-10
OPB_xferAck	M/A	I	OPB transfer acknowledge	3-12
OPB_hwAck	Master	I	OPB halfword acknowledge	3-12
OPB_fwAck	Master	I	OPB fullword acknowledge	3-12
OPB_errAck	Master	I	OPB error acknowledge	3-13
OPB_retry	Master	I	OPB bus cycle retry	3-9
OPB_ToutSup	Arbiter	I	OPB timeout suppress	3-13
SIn_DMAREq	DMA	O	Slave DMA request	3-13
SIn_xferAck	Slave	O	Slave transfer acknowledge	3-12
SIn_hwAck	Slave	O	Slave halfword acknowledge	3-12
SIn_fwAck	Slave	O	Slave fullword acknowledge	3-12
SIn_errAck	Slave	O	Slave error acknowledge	3-13

Table 6-1. Signal Summary Table (cont.)

Signal Name	Interface	I/O	Description	Page
SIn_retry	Slave	O	Slave bus cycle retry	3-9
SIn_ToutSup	Slave	O	Slave timeout suppress	3-13
SIn_DBus(0:31)	Slave	O	Slave data bus	3-10
SIn_DBusEn	Slave	O	Slave data bus enable	3-12
Device Control Register Bus				
CPU_dcrWrite	CPU	I	CPU DCR write	4-5
CPU_dcrRead	CPU	I	CPU DCR read	4-5
CPU_dcrABus(0:9)	CPU	I	CPU DCR address bus	4-5
CPU_dcrDBusOut(0:31)	CPU	I	CPU DCR data bus out	4-6
DCR_cpuDBusIn(0:31)	CPU	O	DCR CPU data bus in	4-6
DCR_cpuAck	CPU	O	DCR CPU acknowledge	4-6

Index

A

- address cycle 1-4
- address pipelining
 - back to back read 2-65
 - processor local bus 2-64
- arbiter output connection
 - on-chip peripheral bus 3-4

B

- back to back read and write 2-69
- bandwidth and latency 2-72
- beat 1-4
- bus transaction 1-4
- bus transfer 1-4

C

- clock cycle 1-4
- CPU_dcrABus(0:9) 4-5
- CPU_dcrDBusOut(0:31) 4-6
- CPU_dcrRead 4-5
- CPU_dcrWrite 4-5

D

- data cycle 1-4
- DCR 4-1
- DCR bus transfers
 - bus loading 4-9
 - CPU same speed as DCR slave 4-11
 - CPU same speed as DCR slave with longer DCR address 4-12
 - CPU slower than DCR slave 4-14
 - CPU two times faster than DCR slave 4-13
- DCR_cpuAck 4-6
- DCR_cpuDBusIn(0:31) 4-6
- device control register bus 4-1
 - implementation 4-3
 - interfaces 4-7
 - operations 4-8
 - cpu same speed as dcr slave 4-15
 - cpu same speed as dcr slave with longer dcr address 4-17

- cpu slower than dcr slave 4-19
- cpu two times faster than dcr slave 4-18

- signals 4-5

- timing guidelines 5-7

- device control registers 4-1

- DMA_dataXfer 2-75

- DMA_endOp 2-75

- DMA_extAck(0:3) 2-76

- DMA_flyByBurst 2-75

- DMA_intAck(0:3) 2-76

- DMA_SInAck 3-13

E

- EXT_dmaReq(0:3) 2-76

I

- instruction

- formats xxii

- forms xxii

- INT_dmaReq(0:3) 2-76

L

- latency count 2-72

- latency counter 2-72

M

- master output connection
 - on-chip peripheral bus 3-3

- Mn_abort 2-18

- Mn_ABus(0:31) 2-26, 3-10

- Mn_BE(0:3) 2-20

- Mn_busLock 3-8

- Mn_compress

- signals

- Mn_compress 2-24

- Mn_DBus(0:31) 3-10

- Mn_DBusEn 3-12

- Mn_fwXfer 3-11

- Mn_guarded 2-24

- Mn_hwXfer 3-11

- Mn_lockErr 2-25

- Mn_ordered 2-25

- Mn_priority(0:1) 2-11

- Mn_rdBurst 2-32

- Mn_request 2-11, 3-8

- Mn_RNW 2-19, 3-11

- Mn_select 3-10
- Mn_seqAddr 3-11
- Mn_size(0:3) 2-22
- Mn_type(0:2) 2-23
- Mn_wrBurst 2-28
- Mn_wrDBus(0:31) 2-27

O

- on-chip peripheral bus 3-1
 - arbiter output connection 3-4
 - arbitration signals 3-7
 - bus signals 3-10
 - data transfer control signals 3-10
 - DMA support signals 3-13
 - features 3-1
 - implementation 3-2
 - interfaces 3-14
 - DMA 3-18
 - master 3-15
 - slave 3-16, 3-17
 - master output connection 3-3
 - operations 3-19
 - signal naming conventions 3-5
 - signals 3-6
 - slave output connection 3-4
 - timing guidelines 5-5
- OPB 3-1
- OPB transfers
 - bus arbitration 3-20
 - basic 3-20
 - bus lock signal 3-22
 - bus master priority 3-24
 - bus parking 3-25
 - continuous bus request 3-21
 - multiple bus request 3-23
 - data 3-26
 - basic data 3-26
 - bus lock 3-32
 - bus lock penalty case 3-33
 - continuous bus request 3-31
 - one cycle latency 3-27
 - overlapped bus arbitration 3-29
 - sequential address signal 3-34
 - two cycle latency 3-28
 - DMA 3-52
 - dynamic bus sizing 3-40

- data alignment 3-42
- fullword byte read 3-44
- fullword byte read and write 3-50
- fullword halfword read and write 3-43
- halfword byte read and write 3-51
- locked with interruption 3-46
- locked with no interruption 3-48
- overlapped arbitration 3-45
- slave retry 3-35
 - bus timeout error 3-37
 - bus timeout error condition 3-38
 - bus timeout error suppression 3-39

- OPB_ABus(0:31) 3-10
- OPB_busLock 3-8
- OPB_DBus(0:31) 3-10
- OPB_errAck 3-13
- OPB_fwAck 3-12
- OPB_fwXfer 3-11
- OPB_hwAck 3-12
- OPB_hwXfer 3-11
- OPB_MnGrant 3-8
- OPB_retry 3-9
- OPB_RNW 3-11
- OPB_select 3-10
- OPB_seqAddr 3-11
- OPB_timeout 3-9
- OPB_toutSup 3-13
- OPB_xferAck 3-12
- overlapped PLB transfers 2-5

P

- PLB 2-1
- PLB DMA operations
 - DMA signals 2-74
 - DMA transfers 2-77
 - DMA buffered memory to
 - peripheral 2-85, 2-89
 - DMA buffered OPB peripheral 2-78
 - DMA buffered peripheral to
 - memory 2-86
 - DMA flyby 2-77
 - DMA flyby burst memory read 2-84
 - DMA flyby memory read 2-82
 - DMA flyby memory write 2-83
 - DMA PLB slave buffered memory to
 - memory 2-87

DMA PLB slave buffered peripheral to memory 2-88	slave requested re arbitration with bus unlocked 2-60
PLB slave buffered memory to memory 2-78, 2-80	transfer abort 2-43
	write 2-42
PLB DMA interface 2-73	PLB_abort 2-18
PLB DMA operations	PLB_ABus(0:31) 2-26
DMA transfers	PLB_BE(0:3) 2-20
DMA buffered external peripheral 2-77	PLB_busLock 2-12
DMA buffered memory 2-80	PLB_compress 2-24
Dma transfers	PLB_guarded 2-24
PLB slave buffered peripheral to/from memory 2-79	PLB_lockErr 2-25
PLB transfer	PLB_masterID(0:3) 2-19
address pipelining	PLB_MBusy(0:n) 2-34
back to back write burst 2-71	PLB_MErr(0:n) 2-34
PLB transfers 2-65	PLB_MnAddrAck 2-17
address pipelining 2-69	PLB_MnRdBTerm 2-33
back to back read burst 2-70	PLB_MnRdDAck 2-31
back to back write 2-67	PLB_MnRdDBus(0:31) 2-30
non address pipelining 2-40	PLB_MnRdWdAddr(0:3) 2-31
back to back burst read burst write 2-58	PLB_MnRearbitrate 2-17
back to back read 2-44	PLB_MnWrBTerm 2-29
back to back read write read 2-46	PLB_MnWrDack 2-28
back to back write 2-45	PLB_ordered 2-25
bus timeout 2-62	PLB_PAVValid 2-12
fixed length burst 2-54	PLB_pendPri(0:1) 2-18
fixed length burst read 2-56	PLB_pendReq 2-18
four word line read 2-47	PLB_rdBurst 2-32
four word line read followed by four word line write 2-49	PLB_rdPrim 2-33
four word line write 2-48	PLB_reqPri(0:1) 2-19
locked 2-59	PLB_RNW 2-19
read 2-41	PLB_SAVValid 2-15
sequential burst read terminated by master 2-50	PLB_size(0:3) 2-22
sequential burst read terminated by slave 2-51	PLB_type(0:2) 2-23
sequential burst write terminated by master 2-52	PLB_wrBurst 2-28
sequential burst write terminated by slave 2-53	PLB_wrDBus(0:31) 2-27
slave requested re arbitration with bus locked 2-61	PLB_wrPrim 2-29
	primary address 1-4
	primary request 1-4
	processor local bus 2-1, 2-72
	address pipelining 2-64
	arbitration signals 2-11
	bandwidth and latency 2-72
	master latency timer 2-72
	DMA PLB sideband signals 2-74
	DMA transfer signals 2-76

- features 2-2
- implementation 2-3
- interfaces 2-36
 - arbiter 2-39
 - master 2-37
 - slave 2-38
- operations 2-40
- overlapped transfers 2-5
- read data bus signals 2-30
- signal naming conventions 2-6
- signals 2-7
- slave output signals 2-34
- status signals 2-18
- system signals 2-10
- terms and definitions 1-4
- timing guidelines 5-1
- transfer protocol 2-4
- transfer qualifier signals 2-19
- write data bus signals 2-26

R

registers

- device control registers 4-1
- latency count 2-72
- latency counter 2-72

S

- secondary address 1-4
- secondary request 1-4
- signals
 - arbitration 2-11, 3-7
 - bus 3-10
 - CPU_dcrABus(0:9) 4-5
 - CPU_dcrDBusOut(0:31) 4-6
 - CPU_dcrRead 4-5
 - CPU_dcrWrite 4-5
 - data transfer control 3-10
 - DCR_cpuAck 4-6
 - DCR_cpuDBusIn(0:31) 4-6
 - device control register bus 4-5
 - DMA PLB sideband 2-74
 - DMA support 3-13
 - DMA transfer 2-76
 - DMA_dataXfer 2-75
 - DMA_endOp 2-75
 - DMA_extAck(0:3) 2-76

- DMA_flyByBurst 2-75
- DMA_intAck(0:3) 2-76
- DMA_SlnAck 3-13
- EXT_dmaReq(0:3) 2-76
- INT_dmaReq(0:3) 2-76
- Mn_abort 2-18
- Mn_ABus(0:31) 2-26, 3-10
- Mn_BE(0:3) 2-20
- Mn_busLock 3-8
- Mn_DBus(0:31) 3-10
- Mn_DBusEn 3-12
- Mn_fwXfer 3-11
- Mn_guarded 2-24
- Mn_hwXfer 3-11
- Mn_lockErr 2-25
- Mn_ordered 2-25
- Mn_priority(0:1) 2-11
- Mn_rdBurst 2-32
- Mn_request 2-11, 3-8
- Mn_RNW 2-19, 3-11
- Mn_select 3-10
- Mn_seqAddr 3-11
- Mn_size(0:3) 2-22
- Mn_type(0:2) 2-23
- Mn_wrBurst 2-28
- Mn_wrDBus(0:31) 2-27
- naming conventions 2-6, 3-5
- on-chip peripheral bus 3-6
- OPB_ABus(0:31) 3-10
- OPB_buslock 3-8
- OPB_DBus(0:31) 3-10
- OPB_errAck 3-13
- OPB_fwAck 3-12
- OPB_fwXfer 3-11
- OPB_hwAck 3-12
- OPB_hwXfer 3-11
- OPB_MnGrant 3-8
- OPB_retry 3-9
- OPB_RNW 3-11
- OPB_select 3-10
- OPB_seqAddr 3-11
- OPB_timeout 3-9
- OPB_toutSup 3-13
- OPB_xferAxk 3-12
- PLB_abort 2-18

- PLB_ABus(0:31) 2-26
- PLB_BE(0:3) 2-20
- PLB_busLock 2-12
- PLB_compress 2-24
- PLB_guarded 2-24
- PLB_lockErr 2-25
- PLB_masterID(0:3) 2-19
- PLB_MBusy(0:n) 2-34
- PLB_MErr(0:n) 2-34
- PLB_MnAddrAck 2-17
- PLB_MnRdDAck 2-31
- PLB_MnRdDBus(0:31) 2-30
- PLB_MnRdWdAddr(0:3) 2-31
- PLB_MnRearbitrate 2-17
- PLB_MnWrBTerm 2-29
- PLB_MnWrDAck 2-28
- PLB_ordered 2-25
- PLB_PAVValid 2-12
- PLB_pendPri(0:1) 2-18
- PLB_pendReq 2-18
- PLB_RdBTerm 2-33
- PLB_rdBurst 2-32
- PLB_rdPrim 2-33
- PLB_reqPri(0:1) 2-19
- PLB_RNW 2-19
- PLB_SAVValid 2-15
- PLB_size(0:3) 2-22
- PLB_type(0:2) 2-23
- PLB_wrBurst 2-28
- PLB_wrDBus(0:31) 2-27
- PLB_wrPrim 2-29
- processor local bus 2-7
- read data bus 2-30
- SI_DataXfer 2-74
- SI_dmaBurstTerm 2-76
- SI_hwAck 3-12
- SI_wrDataXfer 2-75
- slave output 2-34
- Sln_addrAck 2-17
- Sln_DBus(0:31) 3-10
- Sln_DBusEn 3-12
- Sln_dmaReq 3-13
- Sln_errAck 3-13
- Sln_fwAck 3-12
- Sln_MBusy(0:n) 2-34
- Sln_MErr(0:n) 2-34

- Sln_MErr(0:n) 2-34
- Sln_rdBTerm 2-33
- Sln_rdComp 2-32
- Sln_rdDAck 2-31
- Sln_rdDBus(0:31) 2-30
- Sln_rdWdAddr(0:3) 2-31
- Sln_rearbitrate 2-17
- Sln_retry 3-9
- Sln_toutSup 3-13
- Sln_wait 2-16
- Sln_wrBTerm 2-29
- Sln_wrComp 2-28
- Sln_wrDAck 2-28
- Sln_xferAck 3-12
- status 2-18
- Sys_plbClk 2-10
- Sys_plbReset 2-10
- system 2-10
- transfer qualifier 2-19
- write data bus 2-26
- SI_dmaBurstTerm 2-76
- SI_rdDataXfer 2-74
- SI_wrDataXfer 2-75
- slave output connection
 - on-chip peripheral bus 3-4
- Sln_addrAck 2-17
- Sln_DBus(0:31) 3-10
- Sln_DBusEn 3-12
- Sln_dmaReq 3-13
- Sln_errAck 3-13
- Sln_fwAck 3-12
- Sln_hwAck 3-12
- Sln_MBusy(0:n) 2-34
- Sln_MErr(0:n) 2-34
- Sln_rdBTerm 2-33
- Sln_rdComp 2-32
- Sln_rdDAck 2-31
- Sln_rdDBus(0:31) 2-30
- Sln_rdWdAddr(0:3) 2-31
- Sln_rearbitrate 2-17
- Sln_retry 3-9
- Sln_toutSup 3-13
- Sln_wait 2-16
- Sln_wrBTerm 2-29
- Sln_wrComp 2-28

SIn_wrDAck 2-28
SIn_xferAck 3-12
Sys_plbClk 2-10
Sys_plbReset 2-10

T

timing guidelines

device control register bus 5-7

on-chip peripheral bus 5-5

PLB arbiter 5-2

PLB master 5-2

PLB slave 5-4

processor local bus 5-1

transfer protocol

processor local bus 2-4