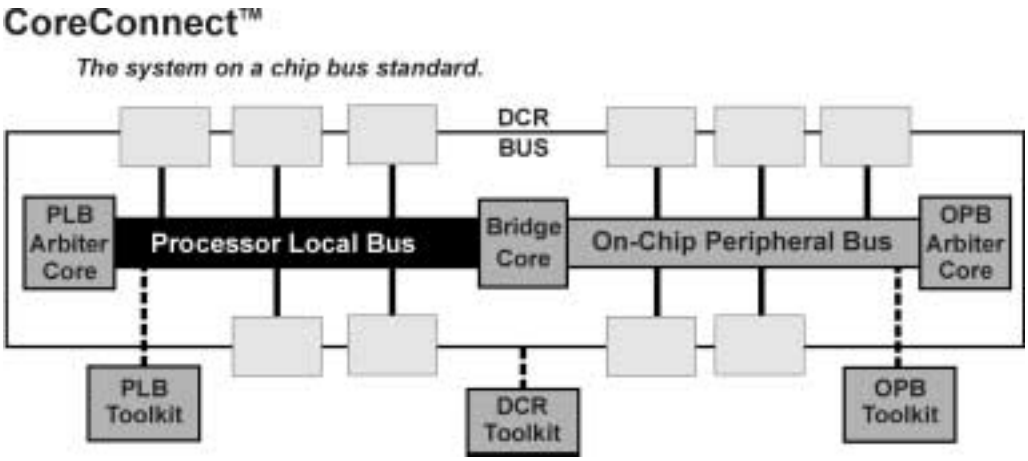
32-bit Processor Local Bus

Architecture Specifications

Version 2.9



SA-14-2531-01

First Edition (May 2001)

This edition of 32-bit Processor Local Bus Architecture Specifications applies to the 32-Bit Implementation of IBM PLB bus, until otherwise indicated in new versions or application notes.

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.

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 YM5A
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.

4 3 2 1

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

Contents

Figures

Tables

About This Book

This book begins with an overview followed by detailed information on Processor Local Bus (PLB) signals, interfaces, timing and operations.

The Processor Local Bus features:

- 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.
- 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.

Who Should Use This Book

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

Related Publications

The following publications contain related information:

Processor Local Bus Architecture Specifications

On-Chip Peripheral Bus Architecture Specifications

Device Control Register Bus Architecture Specifications

Processor Local Bus Toolkit User’s Manual

On-Chip Peripheral Bus Toolkit User’s Manual

Device Control Register Bus Toolkit User’s Manual

Processor Local Bus Arbiter Core User's Manual

On-Chip Peripheral Bus Arbiter Core User's Manual

On-Chip Peripheral Bus Bridge Core User's Manual

How This Book is Organized

This book is organized as follows:

Chapter 1, "PLB Overview"

Chapter 2, "PLB Signals"

Chapter 3, "PLB Interfaces"

Chapter 4, "PLB Timing Guidelines"

Chapter 5, "PLB Operations"

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

- "Contents" on page v
- "Figures" on page ix
- "Tables" on page xi
- "Index" on page 73

Chapter 1. PLB Overview

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.

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.

Figure 1 demonstrates how the processor local bus is inter connected for the purpose of Core+ASIC development or system-on-a-chip design.

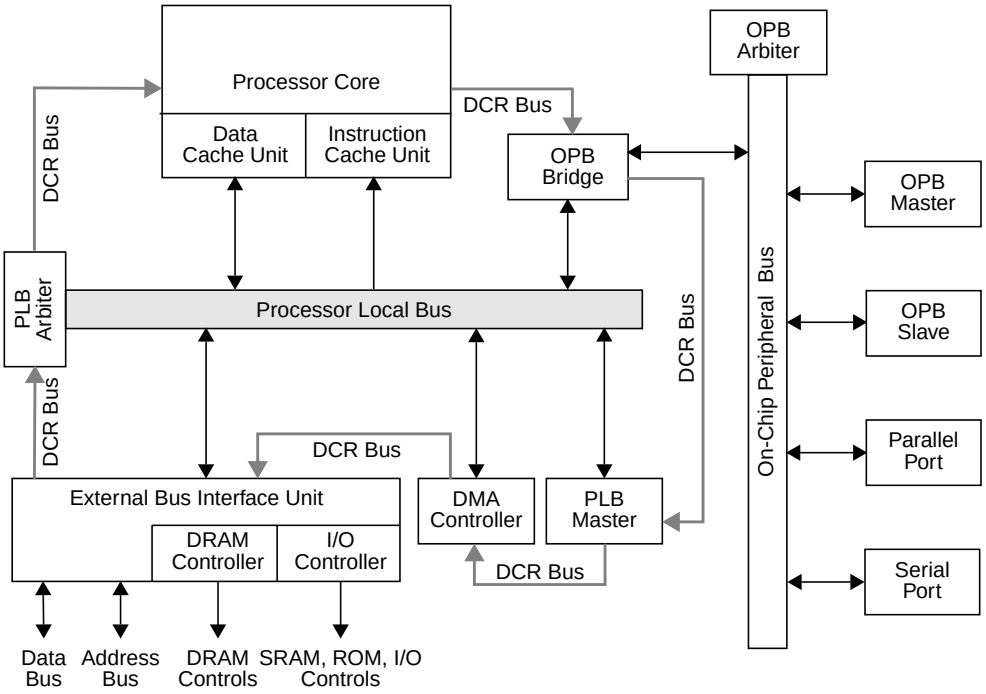


Figure 1. Processor Local Bus Interconnection

As shown in the above figure, 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.

1.1 PLB Features

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

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.

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.

1.2 PLB Implementation

The PLB implementation consists of a PLB core to which all masters and slaves are attached. The logic within the PLB core 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 core 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 core in a particular system will directly affect the performance of the PLB core in that system.

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

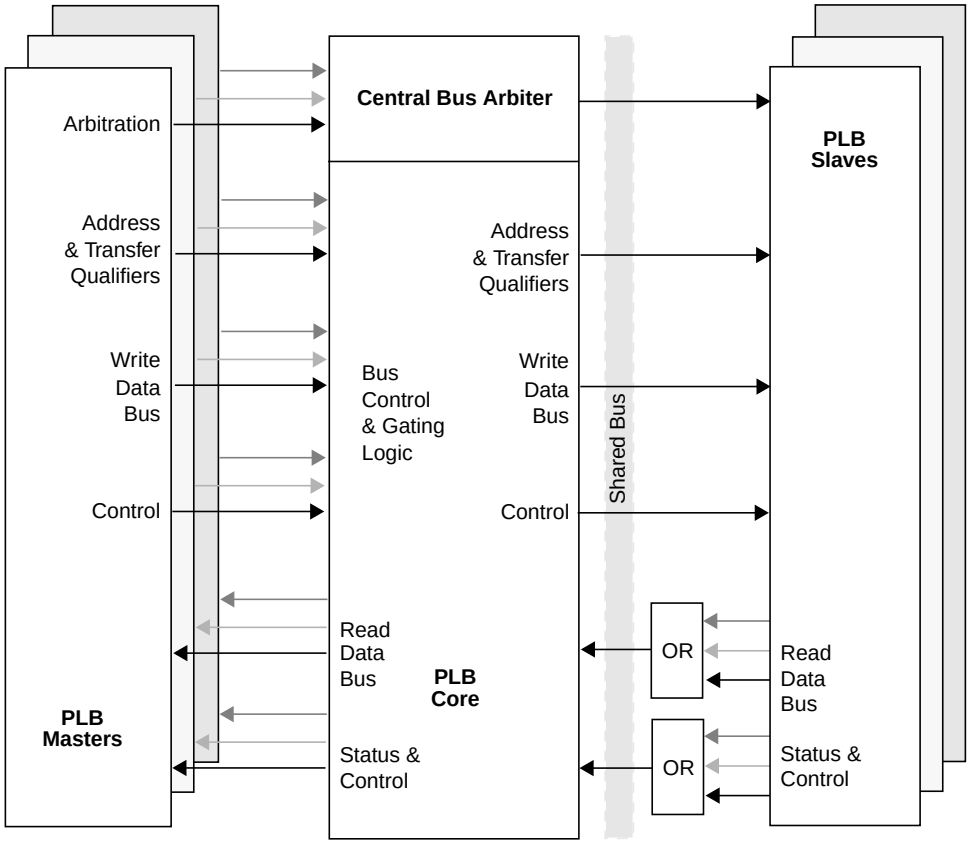


Figure 2. PLB Interconnect Diagram

1.3 PLB Transfer Protocol

A PLB bus transaction as shown in Figure 3 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 3 demonstrates PLB address and data cycles.

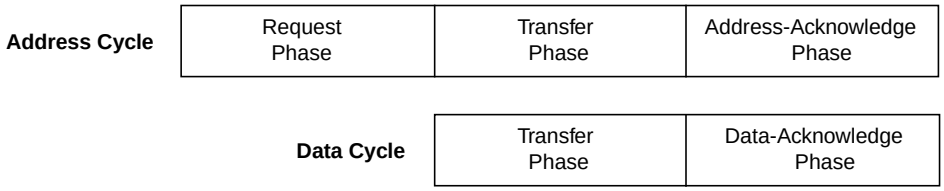


Figure 3. PLB Address and Data Cycles

1.4 Overlapped PLB Transfers

Figure 4 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.

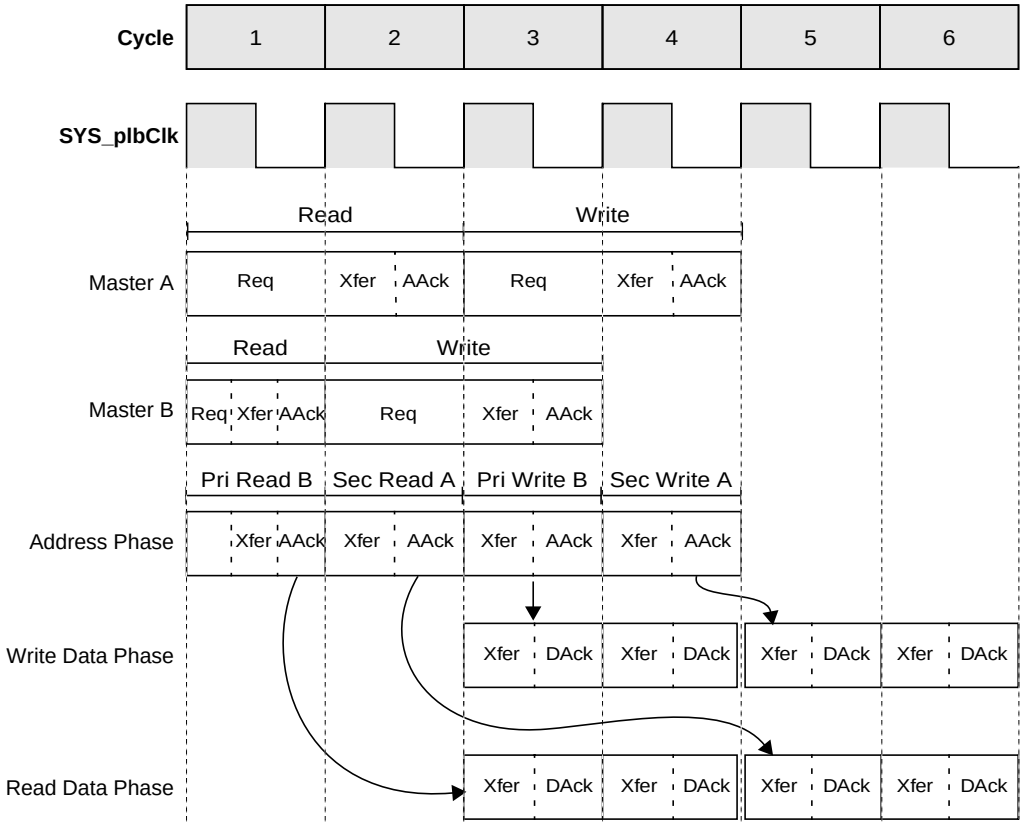


Figure 4. Overlapped PLB Transfers

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.

Chapter 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

2.1 Signal Naming Conventions

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

- Signals which are outputs of the PLB core and inputs to the slave devices are prefixed with “PLB_”. There will only be one output of the PLB core for each one of these signals and it will be received as an input by each slave attached to the PLB core. For example, PLB_PAValid is an output of the PLB core and is an input to each slave attached to the PLB core.
- Signals which are outputs of the slaves and inputs to the PLB core are prefixed with “Sl_”. 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 core. 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 core 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 core 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 core to a master have a prefix PLB_Mn to indicate that this signal is connected from the PLB core to a specific master (that is, PLB_MnAddrAck). The PLB core 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 “Sl” 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 cores capable of supporting less than sixteen masters.

Table 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 1. Summary of PLB Signals

Signal Name	Interface	I/O	Description	Page
Mn_request	Master n	I	Master n bus request	11
Mn_abort	Master n	I	Master n abort bus request indicator	16
Mn_priority(0:1)	Master n	I	Master n bus request priority	11
Mn_busLock[1]	Master n	I	Master n bus lock	12
Mn_RNW	Master n	I	Master n read/write	19
Mn_BE(0:3)	Master n	I	Master n byte enables	19
Mn_size(0:3)	Master n	I	Master n transfer size	21
Mn_type(0:2)	Master n	I	Master n transfer type	22
Mn_compress	Master n	I	Master n compressed data transfer indicator	22
Mn_guarded	Master n	I	Master n guarded transfer indicator	23
Mn_ordered	Master n	I	Master n synchronize transfer indicator	23
Mn_lockErr	Master n	I	Master n lock error indicator	24
Mn_ABus(0:31)	Master n	I	Master n address bus	24
Mn_wrDBus(0:31)	Master n	I	Master n write data bus	25
Mn_wrBurst	Master n	I	Master n burst write transfer indicator	26
Mn_rdBurst	Master n	I	Master n burst read transfer indicator	30
PLB_MnAddrAck	Master n	O	PLB master n address acknowledge	16
PLB_MnRearbitrate	Master n	O	PLB master n bus rearbitrate indicator	16
PLB_Mn_Busy	Master n	O	PLB master n slave busy indicator	32
PLB_Mn_Err	Master n	O	PLB master n slave error indicator	32
PLB_Mn_WrDAck	Master n	O	PLB master n write data acknowledge	26
PLB_Mn_WrBTerm	Master n	O	PLB master n terminate write burst indicator	27
PLB_MnRdDBus(0:31)	Master n	O	PLB master n read data bus	28
PLB_MnRdWdAddr(0:3)	Master n	O	PLB master n read word address	29
PLB_MnRdDAck	Master n	O	PLB master n read data acknowledge	29
PLB_MnRdBTerm	Master n	O	PLB master n terminate read burst indicator	30
PLB_masterID(0:1)	Arbiter	O	PLB current master identifier	18
PLB_PAValid	Arbiter	O	PLB primary address valid indicator	12
PLB_SAValid	Arbiter	O	PLB secondary address valid indicator	14
PLB_pendReq	Arbiter	O	PLB pending bus request indicator	18

Table 1. Summary of PLB Signals (Continued)

Signal Name	Interface	I/O	Description	Page
PLB_abort	Arbiter	O	PLB abort bus request indicator	16
PLB_reqPri(0:1)	Arbiter	O	PLB current request priority	18
PLB_pendPri(0:1)	Arbiter	O	PLB pending request priority	18
PLB_busLock	Arbiter	O	PLB bus lock	12
PLB_RNW	Arbiter	O	PLB read not write	19
PLB_BE(0:3)	Arbiter	O	PLB byte enables	19
PLB_size(0:3)	Arbiter	O	PLB transfer size	21
PLB_type(0:2)	Arbiter	O	PLB transfer type	22
PLB_compress	Arbiter	I	PLB compressed data transfer indicator	22
PLB_guarded	Arbiter	O	PLB guarded transfer indicator	23
PLB_ordered	Arbiter	O	PLB synchronize transfer indicator	23
PLB_lockErr	Arbiter	O	PLB lock error indicator	24
PLB_ABus(0:31)	Arbiter	O	PLB address bus	24
PLB_wrDBus(0:31)	Arbiter	O	PLB write data bus	25
PLB_wrBurst	Arbiter	O	PLB burst write transfer indicator	26
PLB_rdBurst	Arbiter	O	PLB burst read transfer indicator	30
PLB_wrPrim	Arbiter	O	PLB secondary to primary write request indicator	27
PLB_rdPrim	Arbiter	O	PLB secondary to primary read request indicator	31
Sl_addrAck	Slave	I	Slave address acknowledge	16
Sl_rearbitrate	Slave	I	Slave rearbitrate bus indicator	16
Sl_wait	Slave	I	Slave wait indicator	15
Sl_rdComp	Slave	I	Slave read transfer complete indicator	29
Sl_rdDAck	Slave	I	Slave read data acknowledge	29
Sl_rdBTerm	Slave	I	Slave terminate read burst transfer	30
Sl_rdDBus(0:31)	Slave	I	Slave read data bus	28
Sl_rdWdAddr(0:3)	Slave	I	Slave read word address	29
Sl_wrComp	Slave	I	Slave write transfer complete indicator	26
Sl_wrDAck	Slave	I	Slave write data acknowledge	26
Sl_wrBTerm	Slave	I	Slave terminate write burst transfer	27
Sl_MBusy(0:3)	Slave	I	Slave busy indicator	32
Sl_MErr(0:3)	Slave	I	Slave error indicator	32
SYS_plbClk	System	I	System C2 clock	10
SYS_plbReset	System		System PLB reset	10

2.2 PLB System Signals

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

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 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_PAValid is negated).
- The read and write data buses are not being used (that is, all Sl_rdDAck and Sl_wrDAck signals are negated and all SlSl_rdDbus(0:31) and Sl_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.

2.3 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_PAValid or PLB_SAValid signal. The maximum length of the transfer phase is controlled by the slave's Sl_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 Sl_addrAck or Sl_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.

2.3.1 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 Sl_addrAck (PLB_MnAddrAck) or Sl_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.

2.3.2 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 Sl_addrAck or Sl_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 shows Mn_priority(0:1) request priority level.

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

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

2.3.3 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 Sl_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 "PLB_PAValid (PLB Primary Address Valid)" on page 12 and "PLB_SAValid (Secondary Address Valid)" on page 14 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.

2.3.4 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_PAValid signal is summarized in Table 3.

Table 3. PLB_PAValid 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_PAValid 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 Sl_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 Sl_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 Sl_wrComp is asserted if PLB_wrPrim is not also asserted
Read/Write	Y	Busy	Idle	In the clock cycle Sl_rdComp is asserted if PLB_rdPrim is not also asserted
Read/Write	Y	Busy	Busy	In the clock cycle Sl_rdComp is asserted, or the clock cycle following the clock cycle Sl_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 Sl_addrAck and ending with the assertion of Sl_rdComp. For the write data bus, the busy state corresponds to the window of time starting with the clock cycle following the assertion of Sl_addrAck and ending with the clock cycle in which Sl_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 Sl_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 Sl_wait signal to require the PLB arbiter to wait for the request to be properly terminated, or assert the Sl_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 Sl_wait, Sl_AddrAck, or Sl_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 “Bus Transfer Time-Out Notes” on page 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 Sl_rearbitrate, or Sl_addrAck, or Mn_abort, is asserted, or the PLB arbiter times-out.

Note 3: Once a slave has asserted the Sl_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.

2.3.5 PLB_SAValid (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_SAValid 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_SAValid signal is summarized in Table 4 below..

Table 4. PLB_SAValid 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_SAValid is asserted relative to the assertion of Mn_request
Read	N	Idle	Don’t care	PLB_SAValid 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_SAValid 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_SAValid 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 Sl_addrAck and ending with the clock cycle in which Sl_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 Sl_addrAck and ending the clock cycle in which Sl_wrComp is asserted.

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

1. Sl_addrAck is asserted by a slave, or
2. The request is aborted by the requesting master, or
3. Sl_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_SAValid and assert PLB_PAValid, in the same clock cycle Sl_rdComp is asserted, to indicate that there is a new valid primary address and transfer qualifiers on the PLB outputs.

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

1. Sl_addrAck is asserted by a slave, or
2. The request is aborted by the requesting master, or
3. Sl_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_SAValid signal and assert the PLB_PAValid signal, in the clock cycle following the assertion of Sl_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 is Sl_wait or Sl_rearbitrate signals since these will be ignored by the PLB arbiter until the secondary request has become a primary request, PLB_SAValid has been negated, and PLB_PAValid has been asserted.

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

2.3.6 Sl_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 Sl_addrAck or Sl_rearbitrate.

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

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

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

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

2.3.7 Sl_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_PAValid or PLB_SAValid 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_PAValid or PLB_SAValid are first asserted.

2.3.8 Sl_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_PAValid or PLB_SAValid are asserted and should remain negated at all other times.

When asserted in response to the assertion of PLB_PAValid, 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_SAValid, 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_PAValid to be asserted, and then sample Sl_wait, Sl_addrAck, Sl_rearbitrate, and Mn_abort.

Note 1: If Sl_addrAck and Sl_rearbitrate are sampled asserted in the same clock cycle, the PLB arbiter will ignore the Sl_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 Sl_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 “Slave Requested Re-arbitration With Bus Locked” on page 61 for detailed information.

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 Sl_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 Sl_wrDAck and Sl_wrComp with Sl_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 Sl_rearbitrate is asserted in the same clock cycle as Mn_abort, the PLB arbiter will ignore the Sl_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_PAValid or PLB_SAValid are asserted.

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.

2.4.1 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.

2.4.2 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.

2.4.3 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_PAValid or PLB_SAValid 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.

2.4.4 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_PAValid or PLB_SAValid 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 Sl_MBusy signal and Sl_MErr 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 5) is determined by the maximum number of masters supported by the particular PLB arbiter implementation.

Table 5. PLB Master Identification

Maximum # of Masters Supported by PLB Arbiter	PLB_masterID(0:n) Width
2	n = 0

Table 5. PLB Master Identification

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

2.4.5 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_PAValid or PLB_SAValid are asserted. The PLB slave should latch the transfer qualifier signals at the end of the address acknowledge cycle.

2.4.6 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.

2.4.7 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 Sl_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 6 below.

Table 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

Table 6. Byte Enable Signals (Continued)

Mn_BE(0:3)	Transfer Request	Mn_ABus(30:31)
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 7:

Table 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 “Fixed Length Burst Read Transfer” on page 56 for detailed description).

2.4.8 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 8 defines all PLB transfer size signals.

Table 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 Sl_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.

2.4.9 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 9 defines all PLB transfer type signals.

Table 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.

2.4.10 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.

2.4.11 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 Sl_BTerm signals to force the master to terminate the burst transfer. However, masters must not depend on a slave using the Sl_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.

2.4.12 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.

2.4.13 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.

2.4.14 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 “Mn_BE(0:3), PLB_BE(0:3) (Byte Enables)” on page 19 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 10 indicates the bits of the Mn_ABus which must be zeroed for line write transfers.

Table 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 Sl_addrAck.

2.5 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 Sl_wrDAck signal to acknowledge the latching of the write data.

For single-beat transfers, the slave will assert the Sl_wrDAck signal for one clock cycle only. For four-beat, eight-beat, or 16-beat transfers, the slave will assert the Sl_wrDAck signal for 4, 8, and 16 clock cycles, respectively. For burst transfers, the slave will assert the Sl_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 Sl_wrComp signal for one clock cycle.

A slave may request the termination of a write burst transfer by asserting the Sl_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.

2.5.1 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.

2.5.2 Sl_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 an the end of the current clock cycle). For a primary write request, the slave may begin to assert the Sl_wrDAck signal in the clock cycle Sl_addrAck is asserted. For a secondary write request, the slave may begin to assert the Sl_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.

2.5.3 Sl_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.

2.5.4 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 Sl_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 Sl_wrDAck signal is asserted.

The slave will complete the burst transfer by asserting the Sl]_wrComp signal. Note that since it is permissible for the slave to assert the Sl_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 Sl_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 “Pipelined Back-to-Back Write Burst Transfers” on page 70.

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

2.5.5 Sl_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 Sl_addrAck, or during any clock cycle thereafter, up to and including the clock cycle in which Sl_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 Sl_wrComp signal.

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

2.5.6 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_SAValid 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 Sl_wrDAck signal for a secondary write request in the clock cycle following the assertion of PLB_wrPrim.

Note: If a secondary write request is either aborted or address acknowledged in the same clock cycle that Sl_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.

2.6 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 Sl_rdDBus(0:31) onto the PLB_MnRdDBus(0:31) bus. The master will then await for the slave to assert the Sl_rdDAck signal to acknowledge that the data currently on the read data bus is valid.

For single-beat transfers, the slave will assert the Sl_rdDAck signal for one clock cycle only. For four-beat, eight-beat, or 16-beat line transfers, the slave will assert the Sl_rdDAck signal for 4, 8, and 16 clock cycles, respectively. For burst transfers, the slave will assert the Sl_rdDAck 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 Sl_rdComp signal for one clock cycle.

A slave may request the termination of a read burst transfer by asserting the Sl_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.

2.6.1 Sl_rdDBus(0:31), PLB_MnRdDBus(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 Sl_rdDBus(0:31) two clock cycles following the assertion of Sl_addrAck. For a secondary read request, the Slave may begin to drive data on Sl_rdDBus(0:31) two cycles following the assertion of PLB_rdPrim. In both cases, the Slave may drive data on Sl_rdDBus(0:31) thru one clock cycle following the assertion of Sl_rdComp.

Also for a primary read request, the master must begin to sample the PLB_MnRdDAck signal two clock cycles following the assertion of PLB_MnAddrAck. For a secondary read request, the master must begin to sample the PLB_MnRdDAck 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_MnRdDBus(0:31) at the end of the clock cycle in which PLB_MnRdDAck is sampled asserted.

For non-line read transfers, data must always 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 which 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, Sl_rdDBus(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.

2.6.2 Sl_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 Sl_rdDAck is asserted for a read line transfer.

Table 10-1. 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, Sl_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 Sl_rdWdAddr(0:3) signals with non-zero logic values two clock cycles after it has asserted the Sl_addrAck signal, thru one clock cycle following the assertion of Sl_rdComp.

2.6.3 Sl_rdDAck, PLB_MnRdDAck (Read Data Acknowledge)

This signal is driven by the slave to indicate that the data on the Sl_rdDBus(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 Sl_rdDAck signal two clock cycles following the assertion of Sl_addrAck. For a secondary read request, the slave may begin to assert the Sl_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.

2.6.4 Sl_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 Sl_rdComp signal will normally be asserted either in the clock cycle in which the last Sl_rdDAck is asserted, or in a subsequent clock cycle. The Sl_rdComp signal may only be asserted in the cycle prior to the last Sl_rdDAck during read burst transfers if the PLB_rdBurst

signal is negated, or if the Sl_rdBTerm signal is also asserted by the slave (see “Fixed Length Burst Transfer - Notes” on page 54 for more details).

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

2.6.5 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 Sl_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 Sl_rdDAck signal. If asserted, the slave will terminate the transfer by asserting its Sl_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 Sl_rdComp signal with Sl_rdDAck 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 “Pipelined Back-to-Back Read Burst Transfers” on page 69.

Note 2: It is not permissible for the master to assert the Mn_rdBurst signal until one cycle following the assertion of Sl_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.

2.6.6 Sl_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 the clock cycle following the assertion of Sl_addrAck (for a primary read burst request) or PLB_rdPrim (for a secondary read burst request), up to and including the clock cycle in which Sl_rdComp is asserted. This signal must only be asserted for one clock cycle per termination request. In response to the assertion of this signal, the master is required to negate its Mn_rdBurst signal in the following clock cycle for the current burst transfer, unless it has a pipelined read burst transfer acknowledged or

currently being acknowledged. Once the slave asserts Sl_rdBTerm it must no longer sample PLB_rdBurst for the current transfer. The slave will supply one, and only one, additional piece of data. The transfer is then completed when the Sl_rdDAck signal is asserted.

Note 1: If the slave asserts the Sl_rdBTerm signal in the same clock cycle the master negates its Mn_rdBurst signal, no further response is required by the master.

Note 2: 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), if PLB_MnRdBTerm is or has been previously asserted or the Mn_rdBurst signal is negated, the master must sample PLB_MnRdBTerm during the last data transfer of the first request. This is done to determine if the slave is requesting a burst termination for the secondary burst transfer, since Sl_rdComp may be asserted in the clock before the last Sl_rdDAck of the first read request.

2.6.7 PLB_rdPrim (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 Sl_addrAck in response to the assertion of PLB_SAValid. When transferring data for a secondary read request, the slave may begin to drive the Sl_rdDBus(0:31) bus two clock cycles after it has sampled the PLB_rdPrim 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_rdPrim signal will not be asserted.

2.7 Additional Slave Output Signals

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

2.7.1 Sl_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. Sl_MBusy(0) corresponds to Master ID0, Sl_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 Sl_MBusy signal will remain asserted until the final Sl_rdDAck is asserted by the slave. During write transfers, the signal may remain asserted following the assertion of the last Sl_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 Sl_MBusy(0:n) and PLB_MBusy(0:n) signals is determined by the number of masters supported by the particular PLB based system.

2.7.2 Sl_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, Sl_MErr(0) corresponds to master ID 0, Sl_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 Sl_MErr(0:n) and PLB_MErr(0:n) signals is determined by the number of masters supported by the particular PLB based system.

Chapter 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 “PLB Signals” on page 7.

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

3.1 PLB Master Interface

Figure 5 demonstrates all PLB master interface input/output signals. See “PLB Signals” on page 7 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.

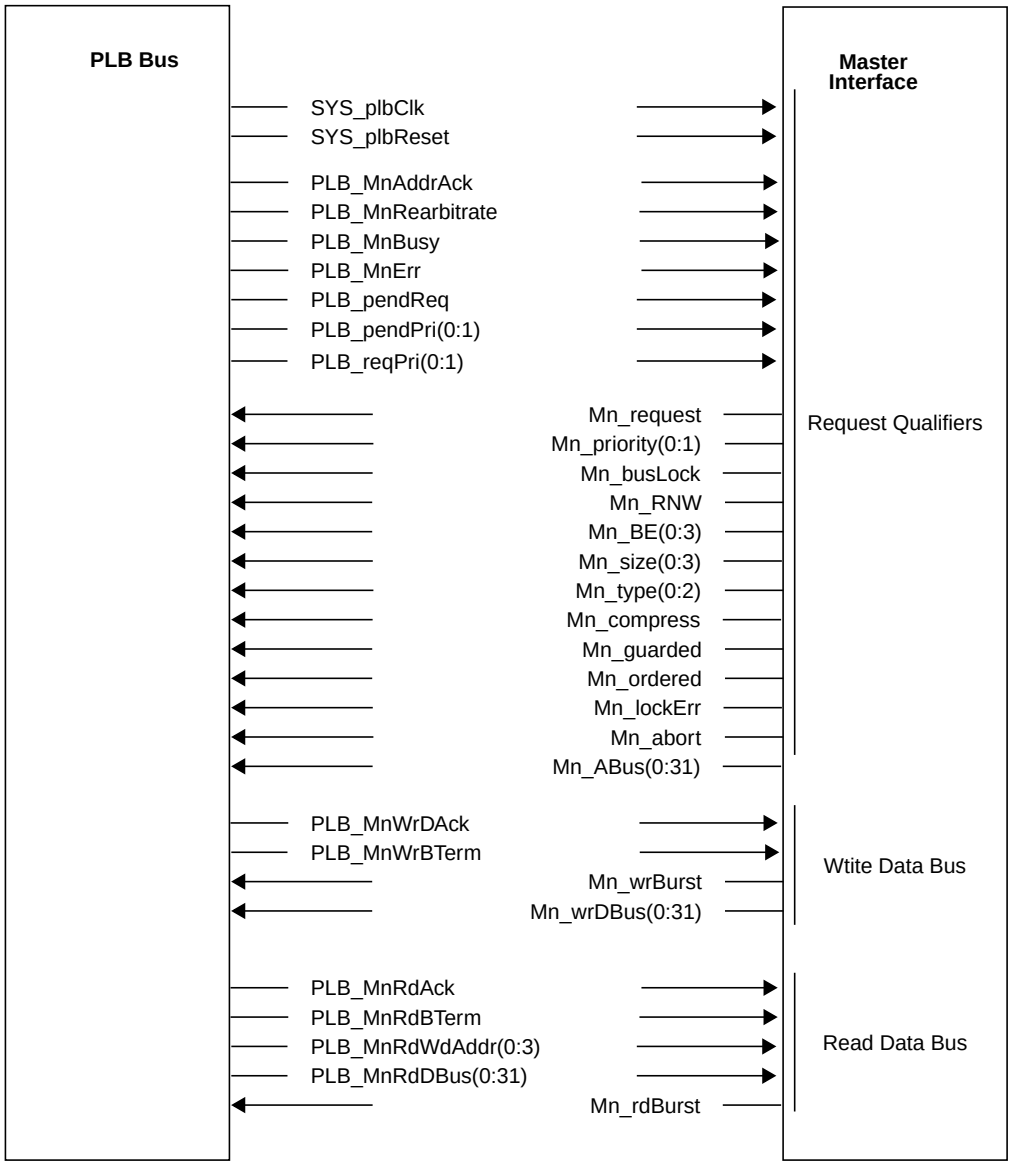


Figure 5. Master Interface

3.2 PLB Slave Interface

Figure 6 demonstrates all PLB slave interface input/output signals. See “PLB Signals” on page 7 for detailed functional description of the signals.

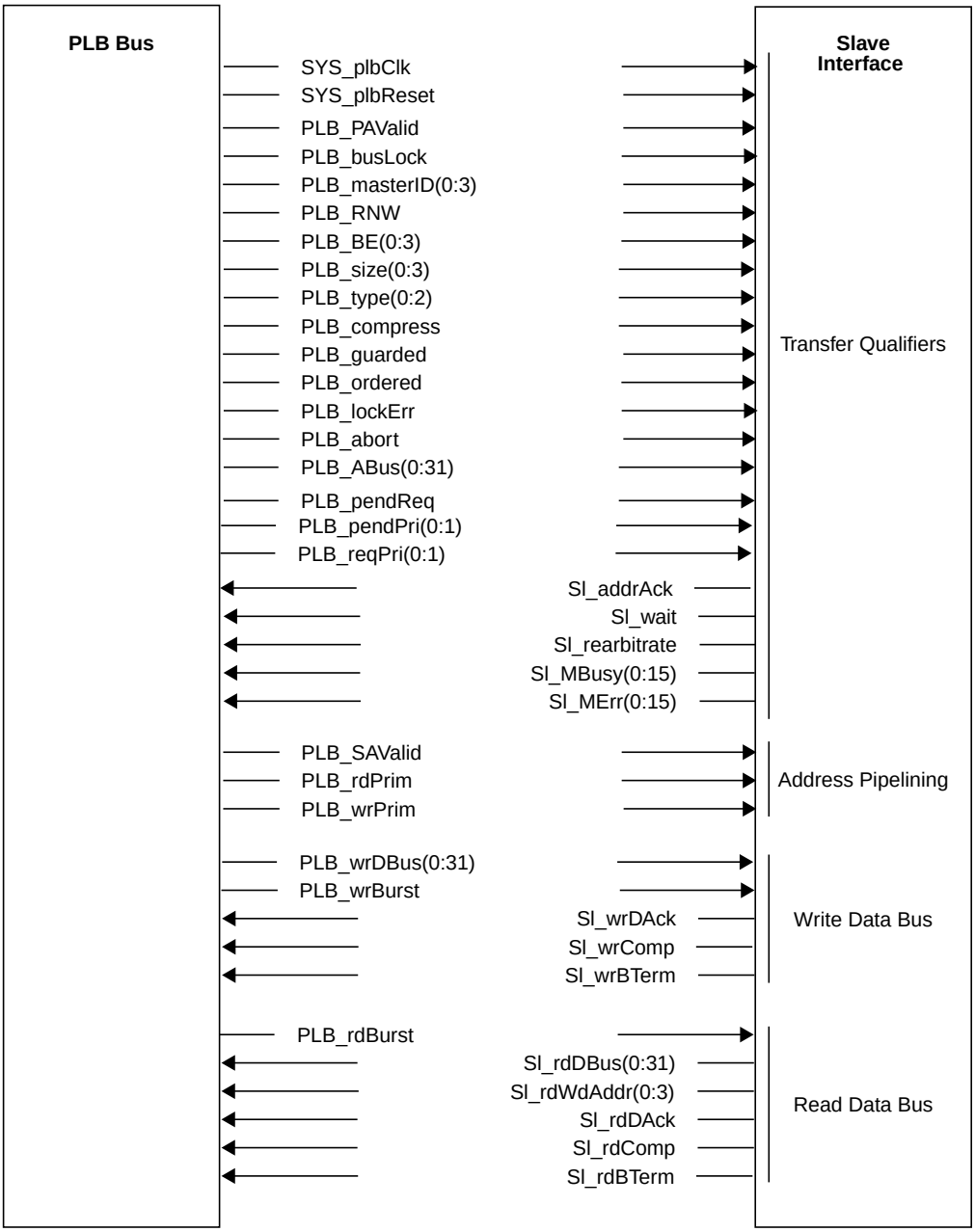


Figure 6. PLB Slave Interface

3.3 PLB Arbiter Interface

Figure 7 demonstrates all PLB arbiter interface input/output signals. See “PLB Signals” on page 7 for detailed functional description of the signals.

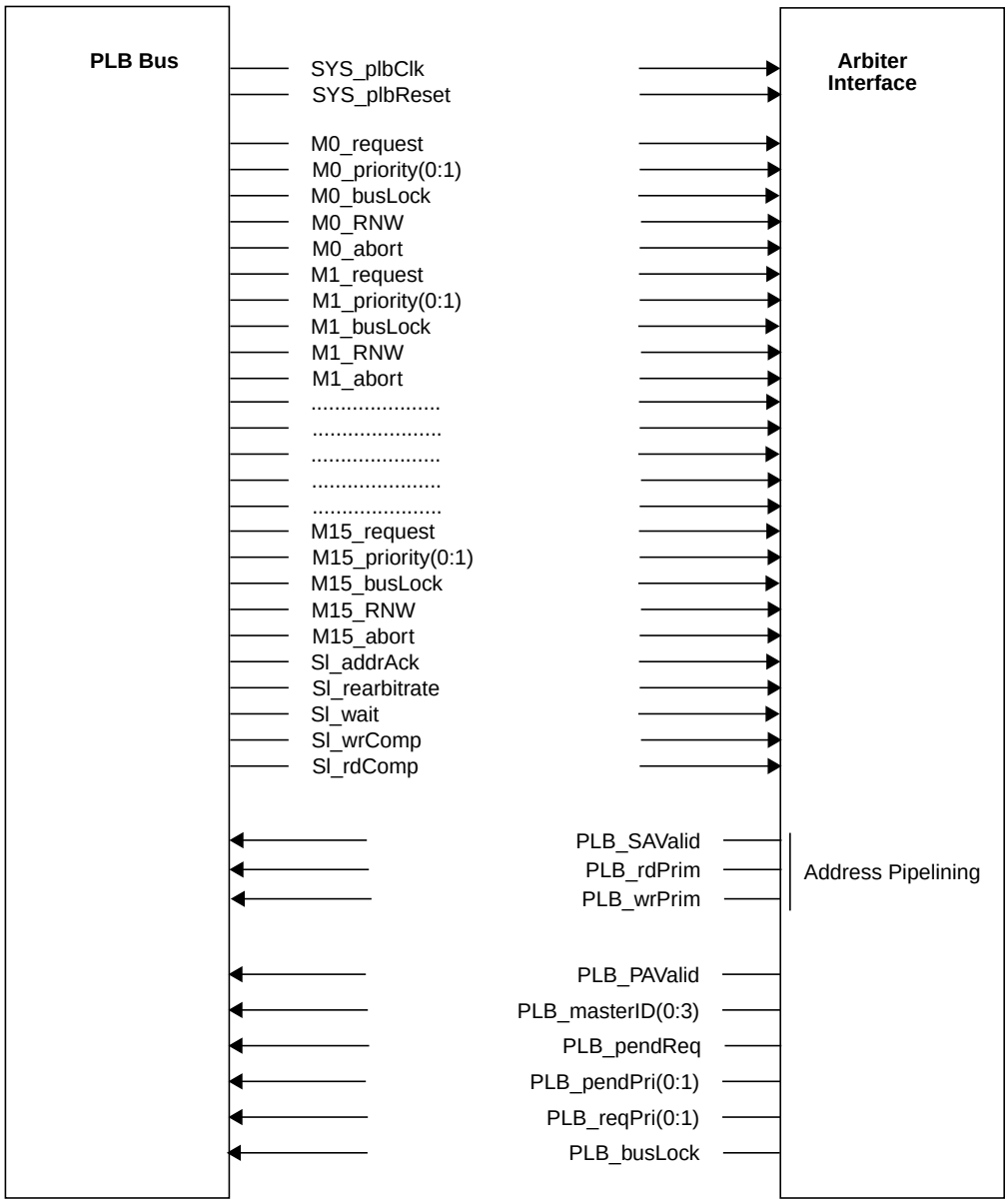


Figure 7. PLB Arbiter Interface

Chapter 4. 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 design guides 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.

4.1 PLB Master Timing Guidelines

Table 11 describes PLB master signal timing guidelines.

Table 11. 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

Table 11. PLB Master Signal TIming Guidelines

PLB Signal Name	Driven By	Output Valid	Received by
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

4.2 PLB Arbiter Timing Guidelines

Table 12 describes PLB arbiter signal timing guidelines.

Table 12. 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
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_PAValid	PLB arbiter	Middle	Slaves
PLB_SAValid	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

Table 12. PLB Arbiter Signal Timing Guidelines (Continued)

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

4.3 PLB Slave Timing Guidelines

Table 13 describes PLB slave signal timing guidelines.

Table 13. PLB Slave SIgnal Timing Guidelines

PLB Signal Name	Driven By	Output Valid	Received by
Sl_MBusy	Slaves	Early	PLB arbiter
Sl_MErr	Slaves	Early	PLB arbiter
Sl_rdwdAddr	Slaves	Early	PLB arbiter
Sl_rdDAck	Slaves	Early	PLB arbiter
Sl_rdComp	Slaves	Early	PLB arbiter
Sl_rdDBus	Slaves	Early	PLB arbiter
Sl_rdBTerm	Slaves	Middle -	PLB arbiter
Sl_wrBTerm	Slaves	Late -	PLB arbiter
Sl_wrDAck	Slaves	Late -	PLB arbiter
Sl_wrComp	Slaves	Late	PLB arbiter
Sl_addrAck	Slaves	Late -	PLB arbiter
Sl_wait	Slaves	Late -	PLB arbiter
Sl_rearbitrate	Slaves	Late -	PLB arbiter

Chapter 5. 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.

5.1 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 “PLB Timing Guidelines” on page 37.

5.1.1 Read Transfers

Figure 8 shows the operation of a single read data transfer on the PLB. The slave asserts Sl_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_PAValid signal as well as the address and transfer qualifier signals until the slave device asserts the Sl_addrAck signal. The slave then asserts the Sl_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.

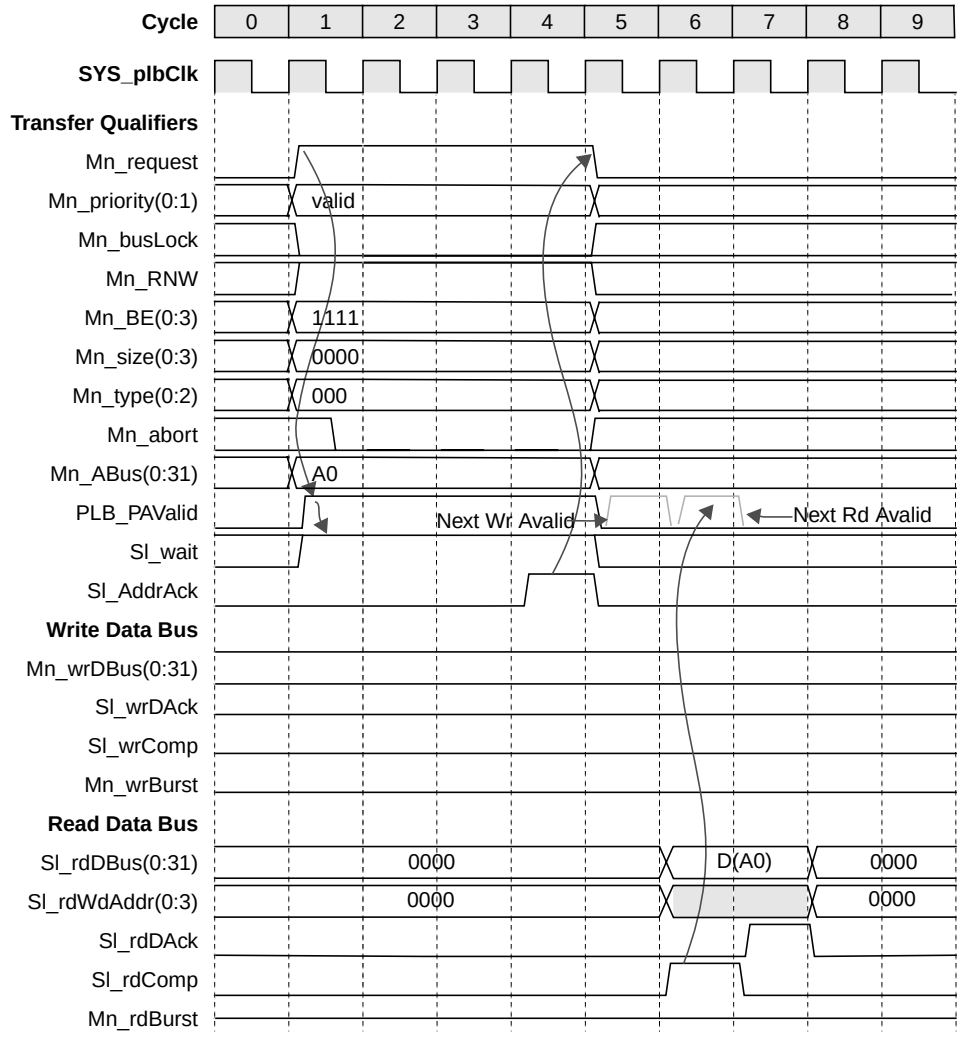


Figure 8. Read Transfers

5.1.2 Write Transfers

Figure 9 shows the operation of a single write data transfer on the PLB. The slave asserts the Sl_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 Sl_addrAck signal. The slave then asserts the Sl_wrComp and Sl_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 Sl_wrDAck signal is asserted.

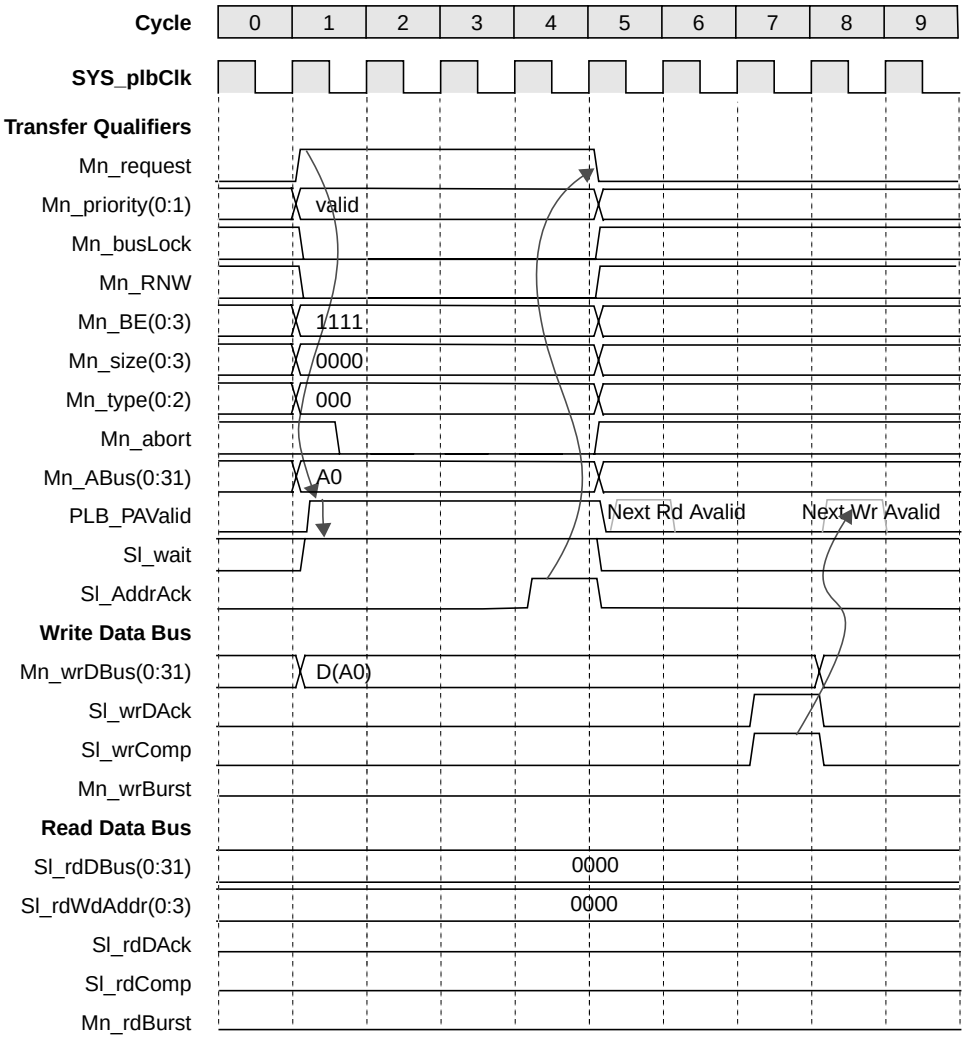


Figure 9. Write Transfers

5.1.3 Transfer Abort

Figure 10 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.

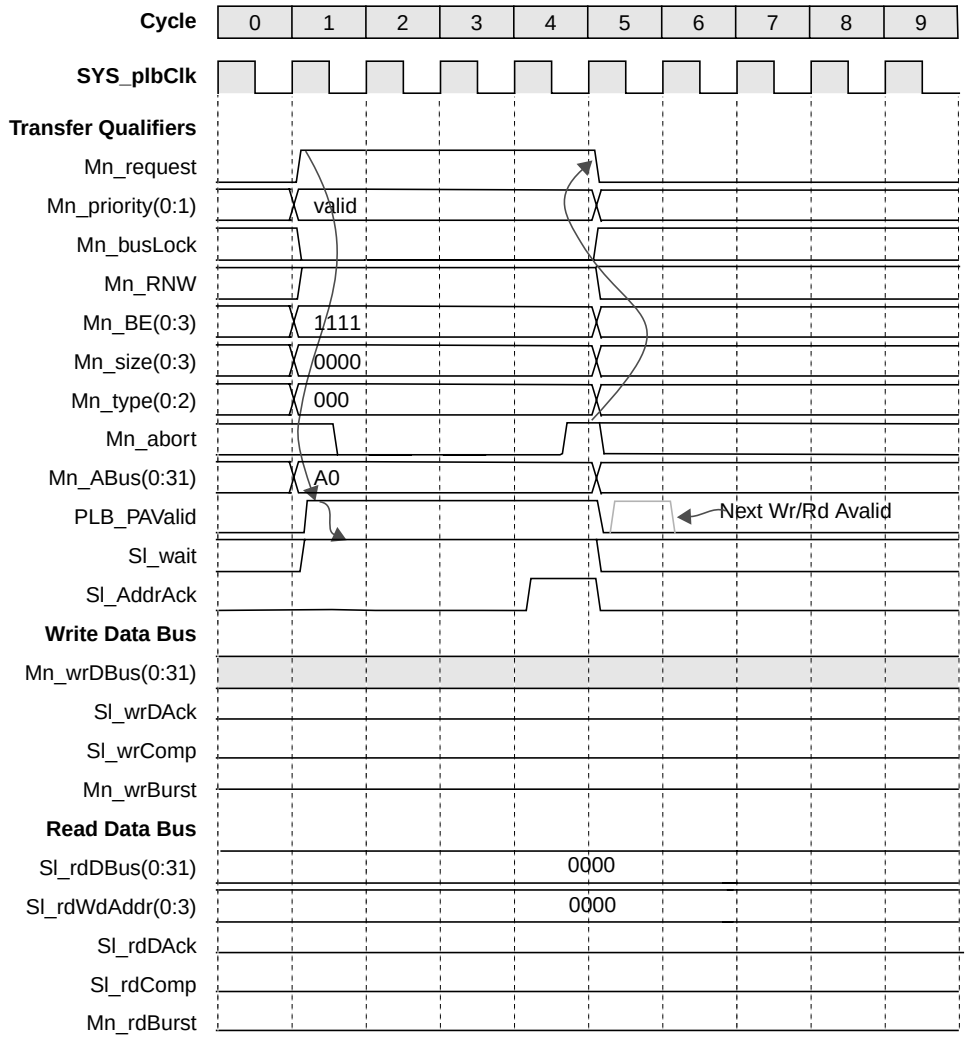


Figure 10. Transfer Abort

5.1.4 Back-to-Back Read Transfers

Figure 11 shows the operation of several back-to-back single read transfers on the PLB. The slave asserts the Sl_rdComp signal in the clock cycle preceding the Sl_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 Sl_rdDAck for the data read until two clock cycles following the assertion of the corresponding Sl_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.

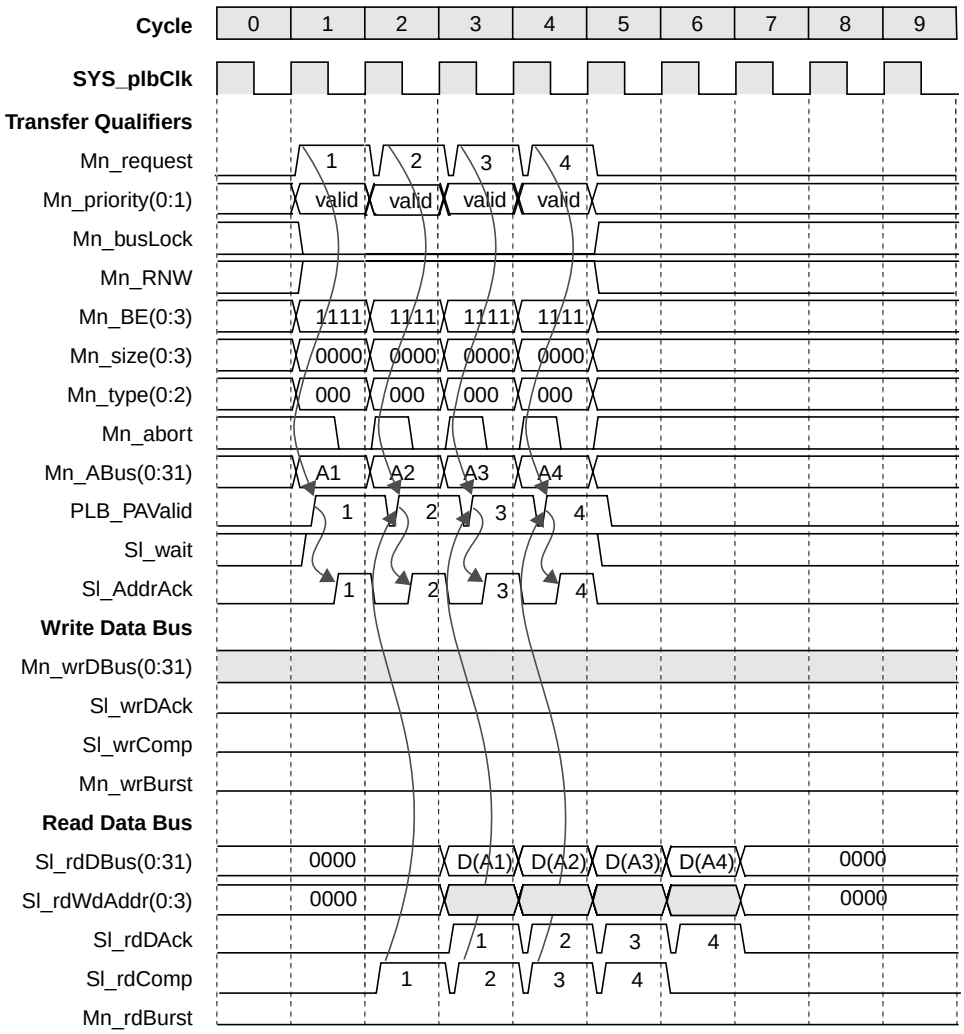


Figure 11. Back-to-Back Read Transfers

5.1.5 Back-to-Back Write Transfers

Figure 12 shows the operation of several back-to-back single write transfers on the PLB. The slave must assert the Sl_addrAck, Sl_wrDAck, and Sl_wrComp signals in the same clock cycle that the PLB_PAValid 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 Sl_wrComp signal.

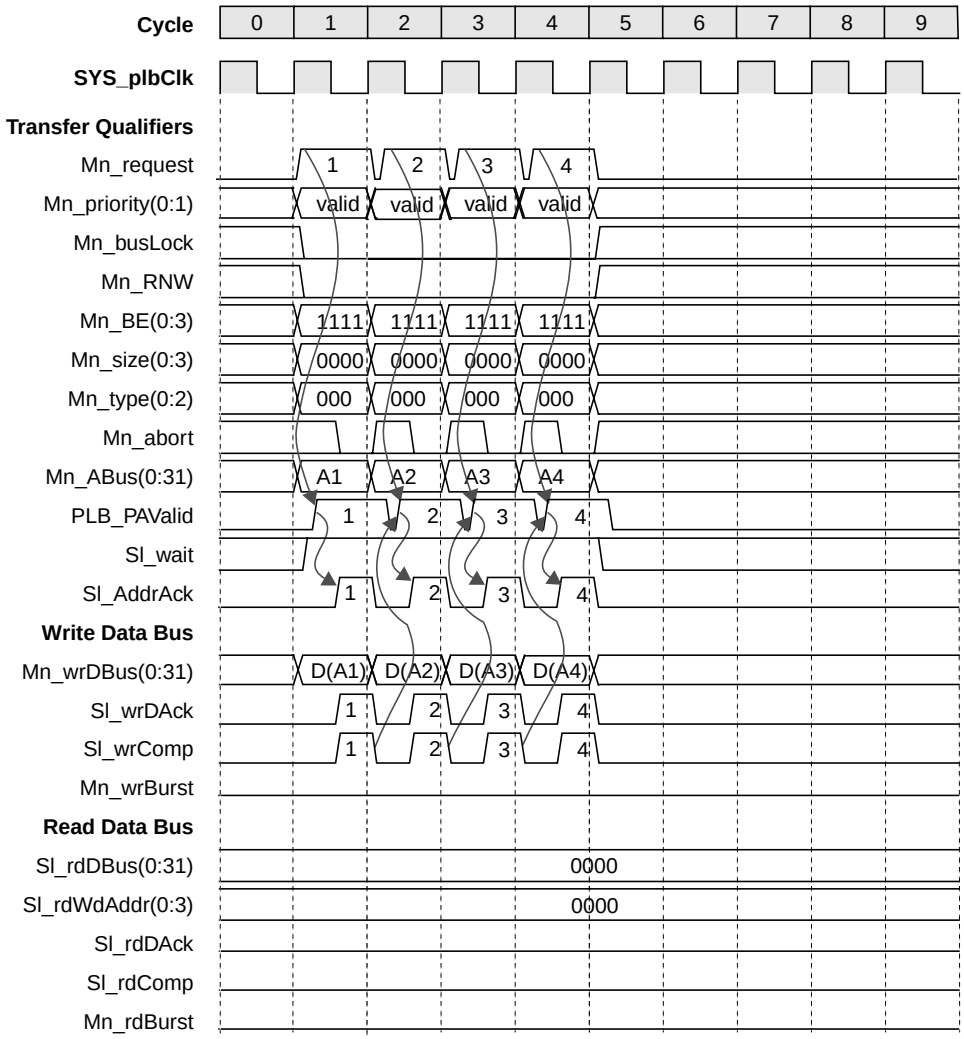


Figure 12. Back-to-Back Write Transfers

5.1.6 Back-to-Back Read - Write - Read - Write Transfers

Figure 13 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.

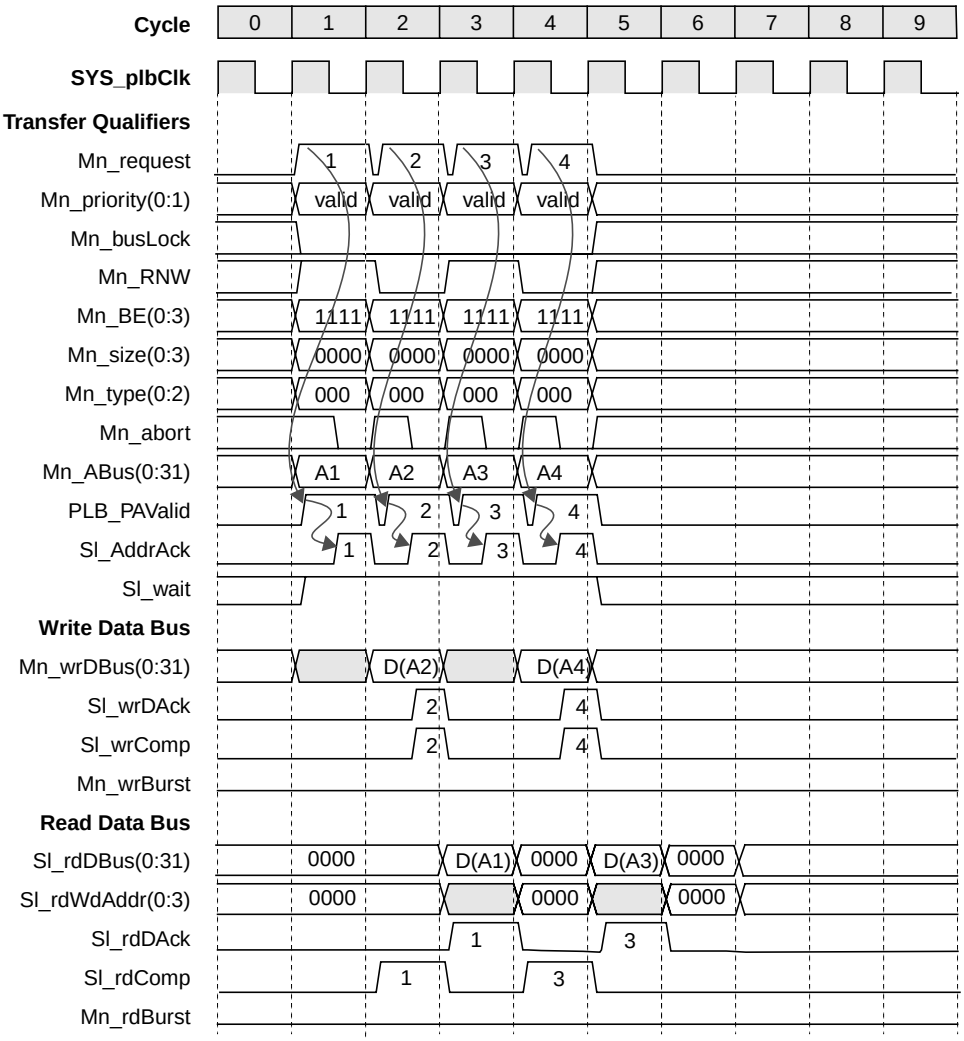


Figure 13. Back-to-Back Read - Write - Read - Write

5.1.7 Four-word Line Read Transfers

Figure 14 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 Sl_rdWdAddr(0:3) outputs of the slave will indicate to the master which word is being transferred in each data transfer cycle. The Sl_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.

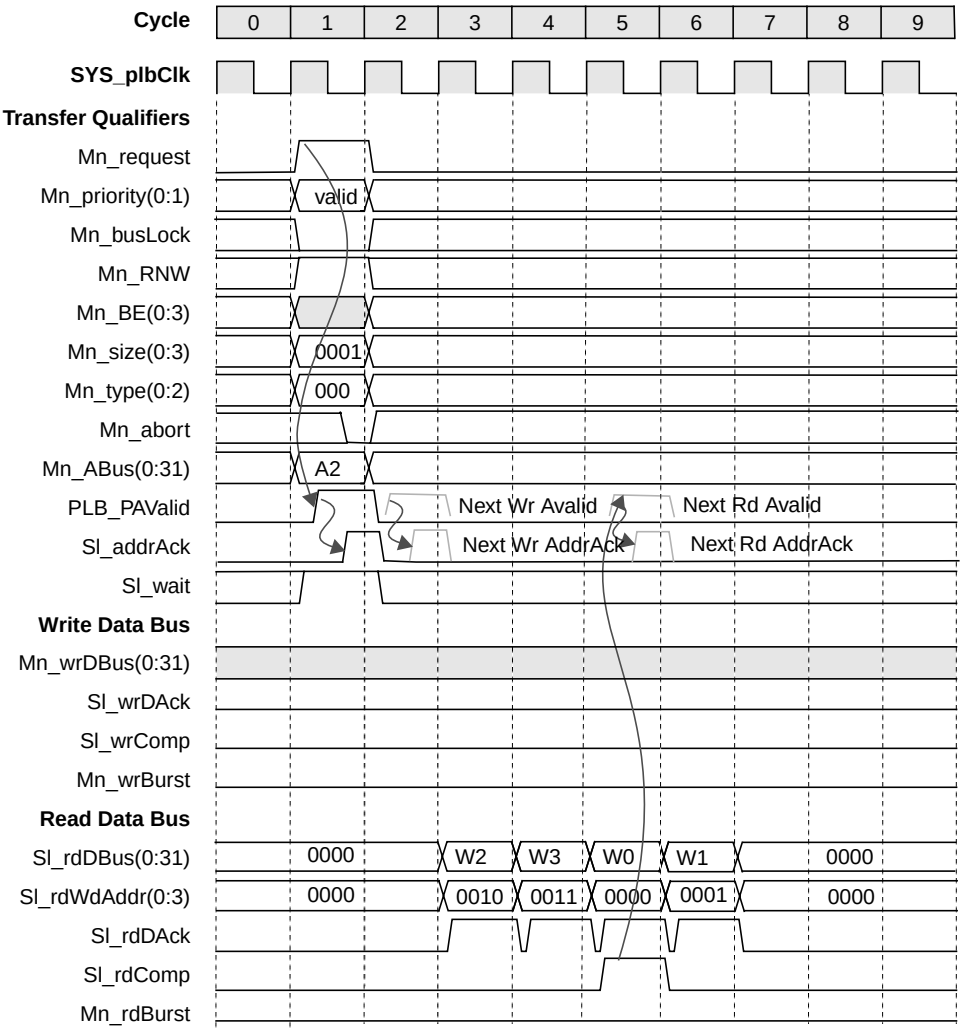


Figure 14. Four Word Line Read

5.1.8 Four-word Line Write Transfers

Figure 15 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 Sl_addrAck and the Sl_wrDAck signals but not the Sl_wrComp signal. The Sl_wrComp signal is asserted during the clock cycle in which the last Sl_wrDAck signal is asserted and is used by the PLB arbiter to allow the next write request to be gated onto the PLB.

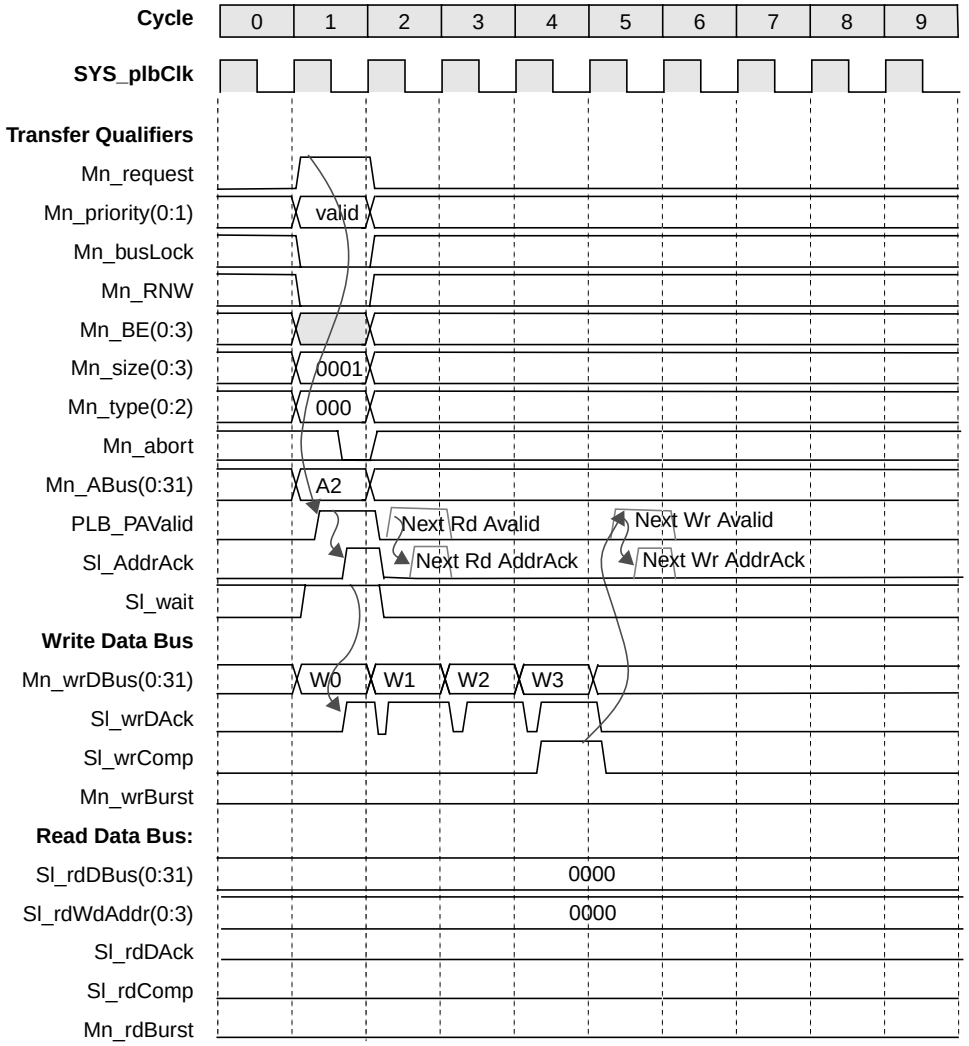


Figure 15. Four Word Line Write

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

Figure 16 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.

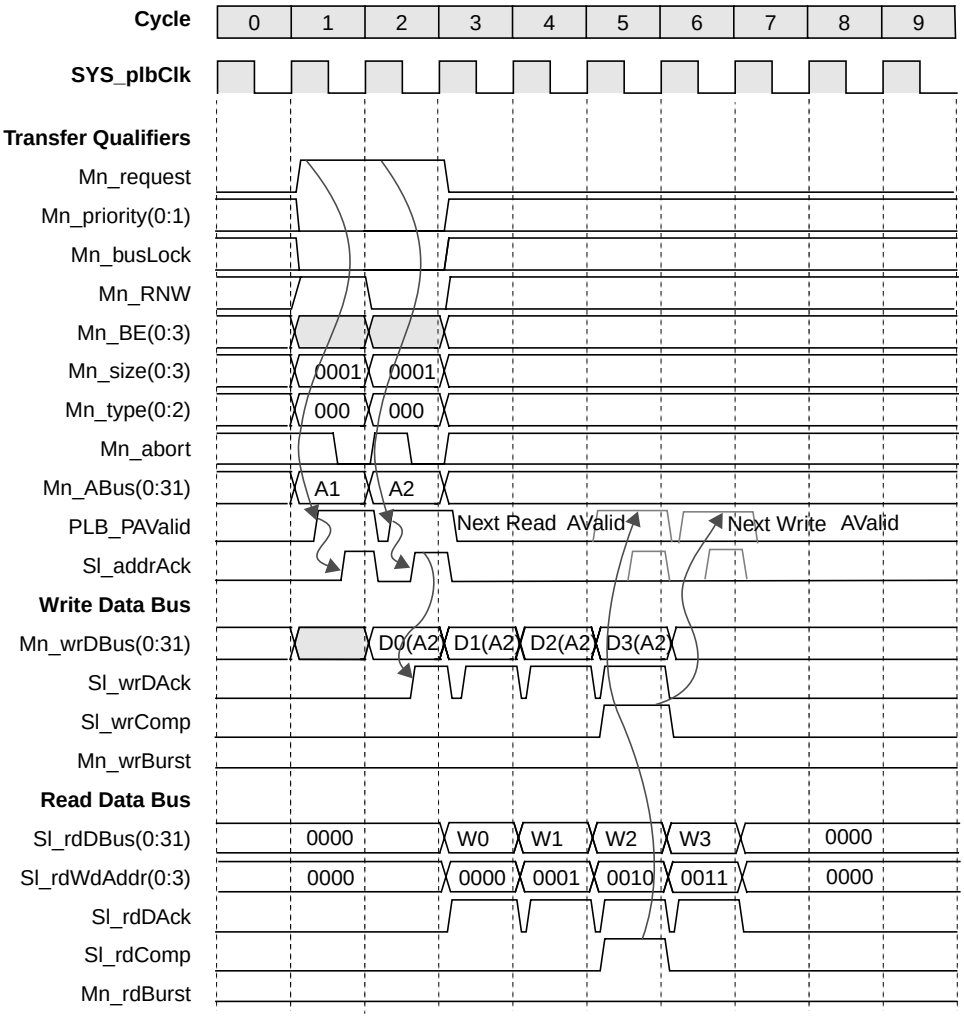


Figure 16. Four Word Line Read followed by Four Word Line Write

5.1.10 Sequential Burst Read Transfer Terminated by Master

Figure 17 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 Sl_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 Sl_rdComp in the data acknowledge phase of the last data transfer cycle following the negation of the Mn_rdBurst signal.

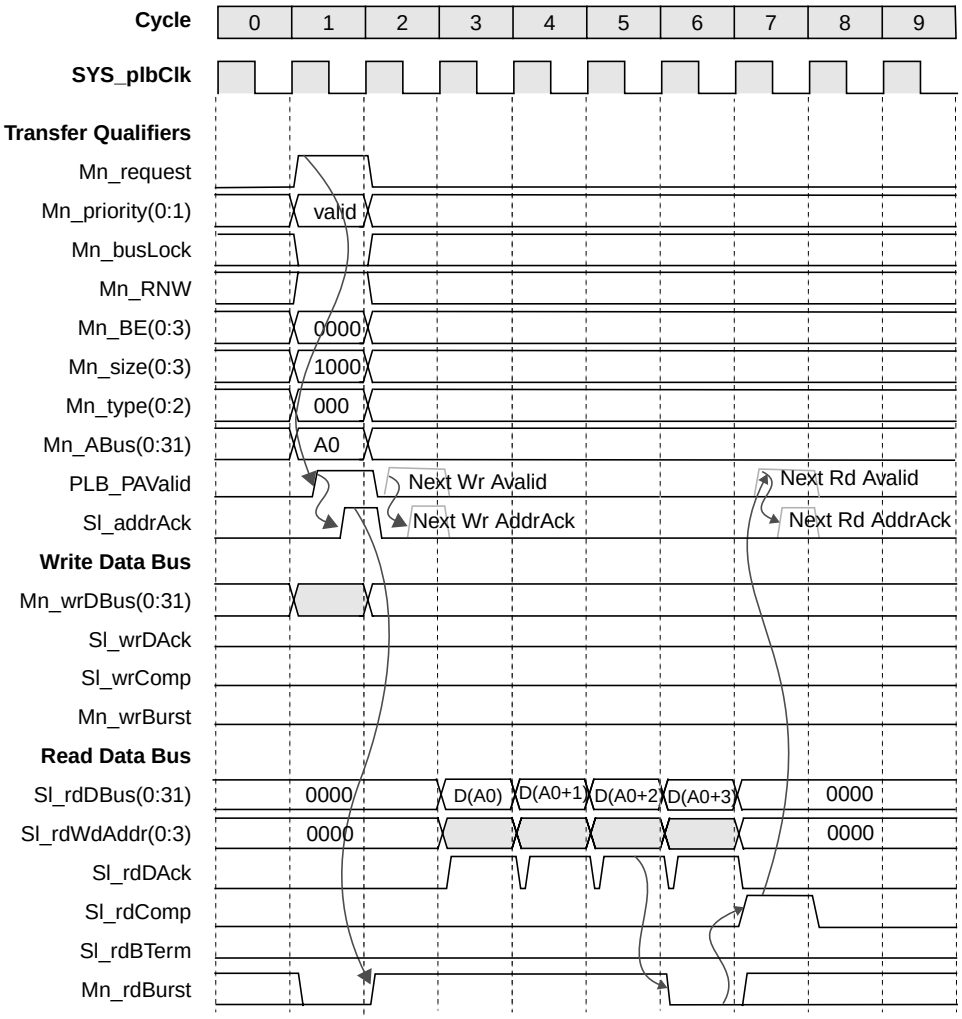


Figure 17. Burst Read Transfer Terminated By Master)

5.1.11 Sequential Burst Read Transfer Terminated By Slave

Figure 18 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 17 in that the transfer is terminated by the master negating the Mn_rdBurst signal in response to the assertion of the Sl_rdBTerm by the slave device. The burst transfer is then completed by the slave device asserting the Sl_rdComp in the data acknowledge phase of the last data transfer cycle.

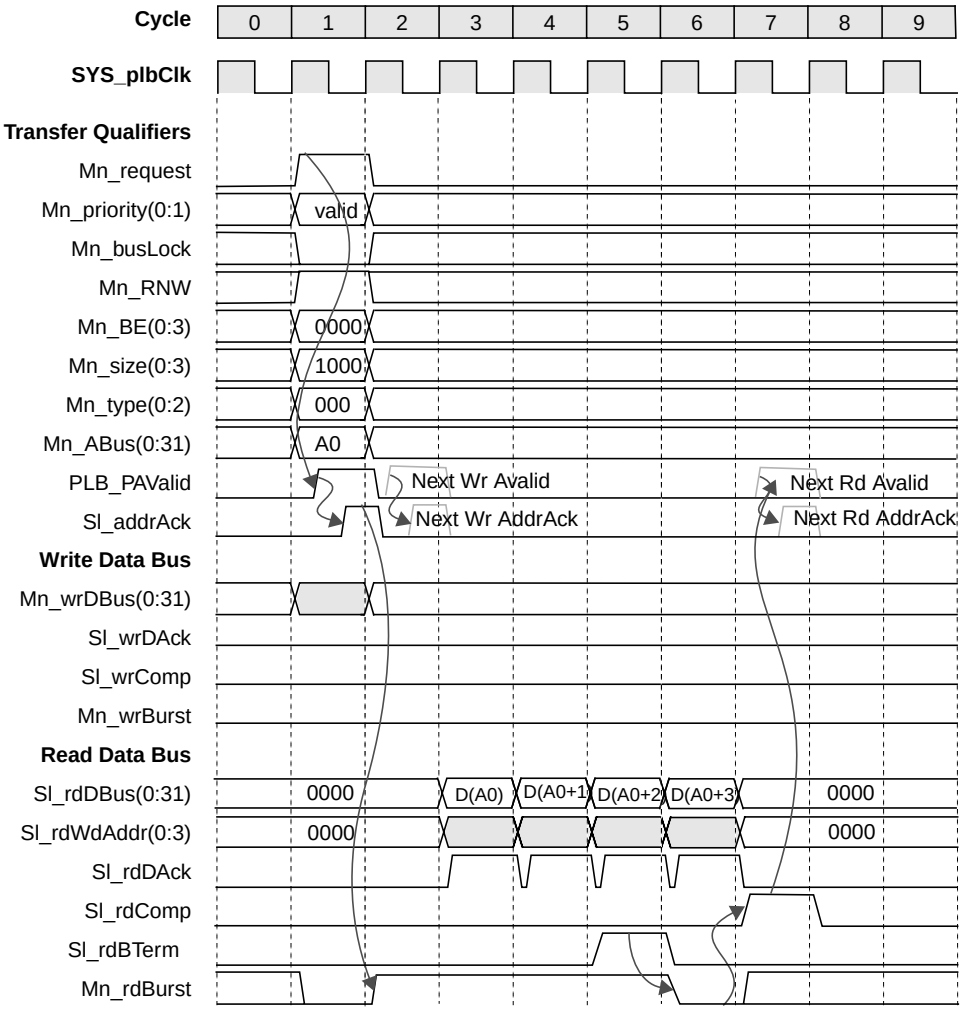


Figure 18. Burst Read Transfer Terminated By Slave

5.1.12 Sequential Burst Write Transfer Terminated by Master

Figure 19 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 Sl_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 Sl_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.

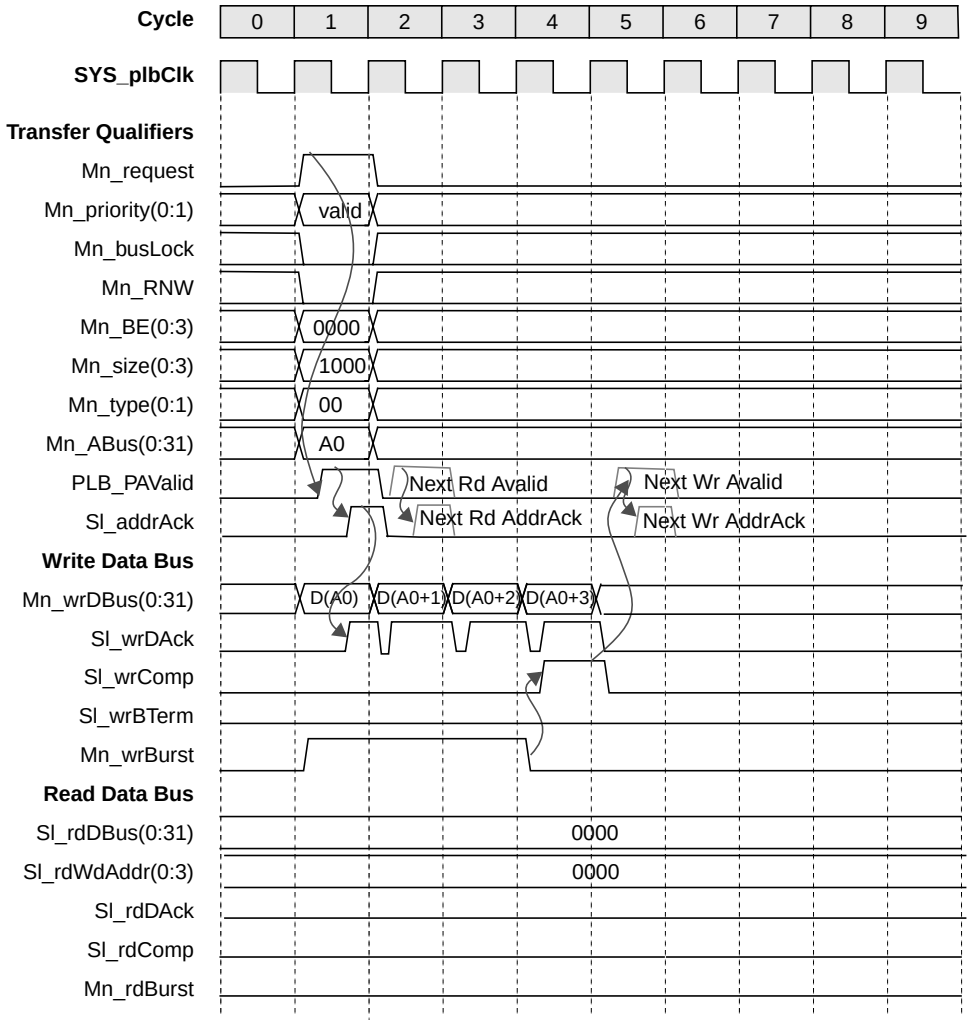


Figure 19. Burst Write Transfer Terminated by Master

5.1.13 Sequential Burst Write Transfer Terminated By Slave

Figure 20 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 19 in that the transfer is terminated by the master negating the Mn_wrBurst signal in response to the assertion of the Sl_wrBTerm signal by the slave device. The burst transfer is then completed by the slave device asserting the Sl_wrComp during the data acknowledge phase for the next (and last) data transfer cycle.

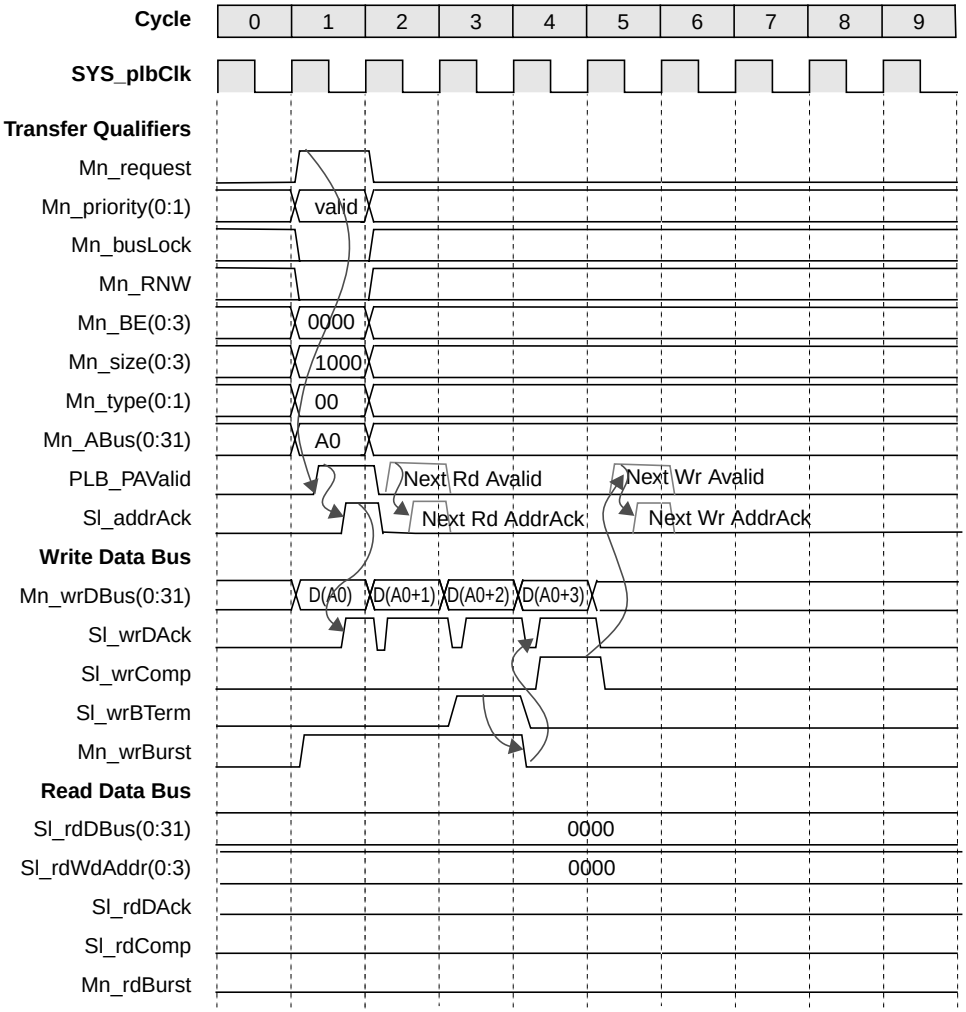


Figure 20. Burst Write Transfer Terminated By Slave

5.1.14 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 17 and Figure 18). 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 14.

Table 14. 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
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 Sl_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 Sl_rdBTerm and Sl_rdComp in the cycle prior to the last Sl_rdDAck. This will allow for a subsequent read or write transfer to be acknowledged in the Sl_rdComp cycle and thus fully utilize the read data bus. However, it should be noted that if the Sl_rdComp and Sl_rdBterm signals are asserted in the cycle prior to the last assertion of Sl_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 Sl_rdComp.

For write burst transfers, if the PLB_wrBurst signal is not negated early, then the slave should assert Sl_wrBTerm in the clock cycle prior to the last Sl_wrDAck and assert the Sl_wrComp in the same clock cycle as the assertion of the last Sl_wrDAck. This will allow for a subsequent read or write transfer to be acknowledged in the cycle following the assertion of Sl_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 Sl_rd/wrBTerm or in the clock cycle following the assertion of the second to last Sl_rd/wrDAck. This allows the transfer to complete when bursting to a slave which has not implemented the fixed length burst protocol.

5.1.15 Fixed Length Burst Read Transfer

Figure 21 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 Sl_rdComp and Sl_rdBTerm in the cycle prior to the last assertion of Sl_rdDAck. This allows a subsequent read transfer to be acknowledged.

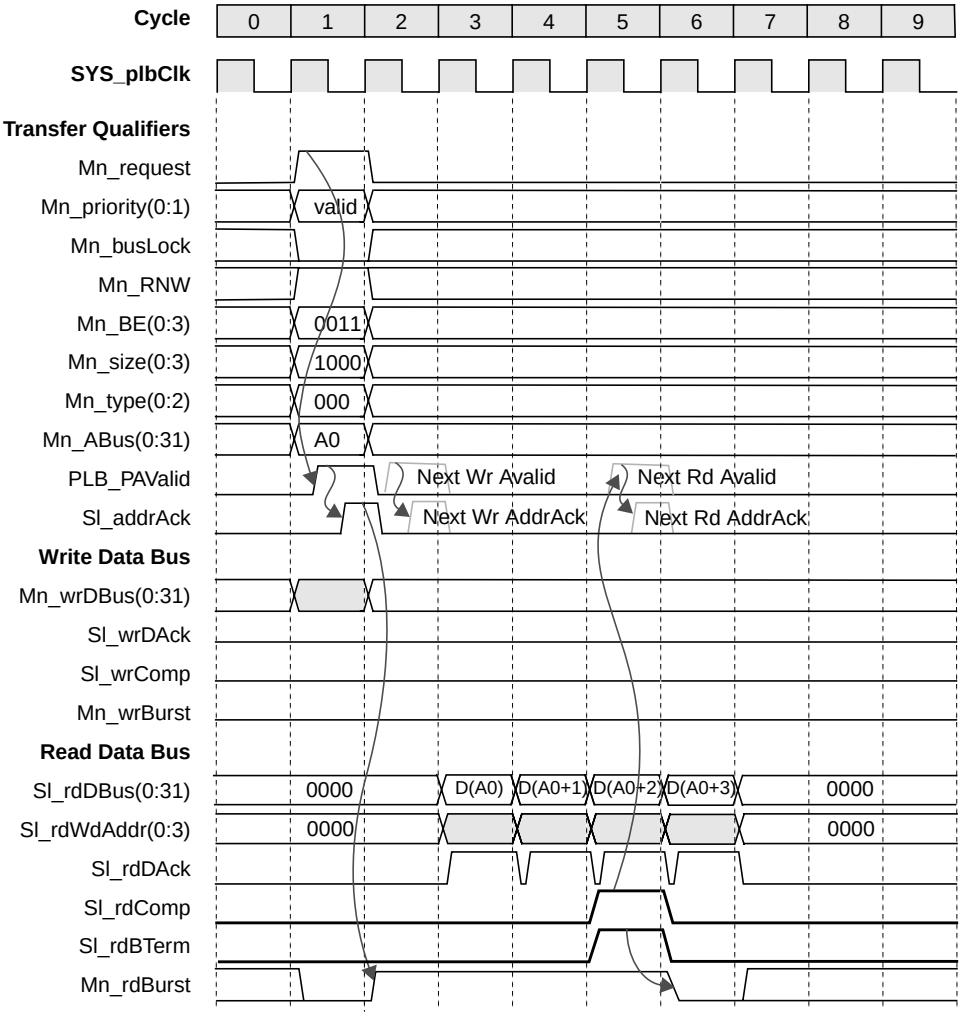


Figure 21. Fixed Length Burst Read Transfer

5.1.16 Fixed Length Burst Write Transfer

Figure 22 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 Sl_wrBTerm in the cycle prior to the last assertion of Sl_wrDAck.

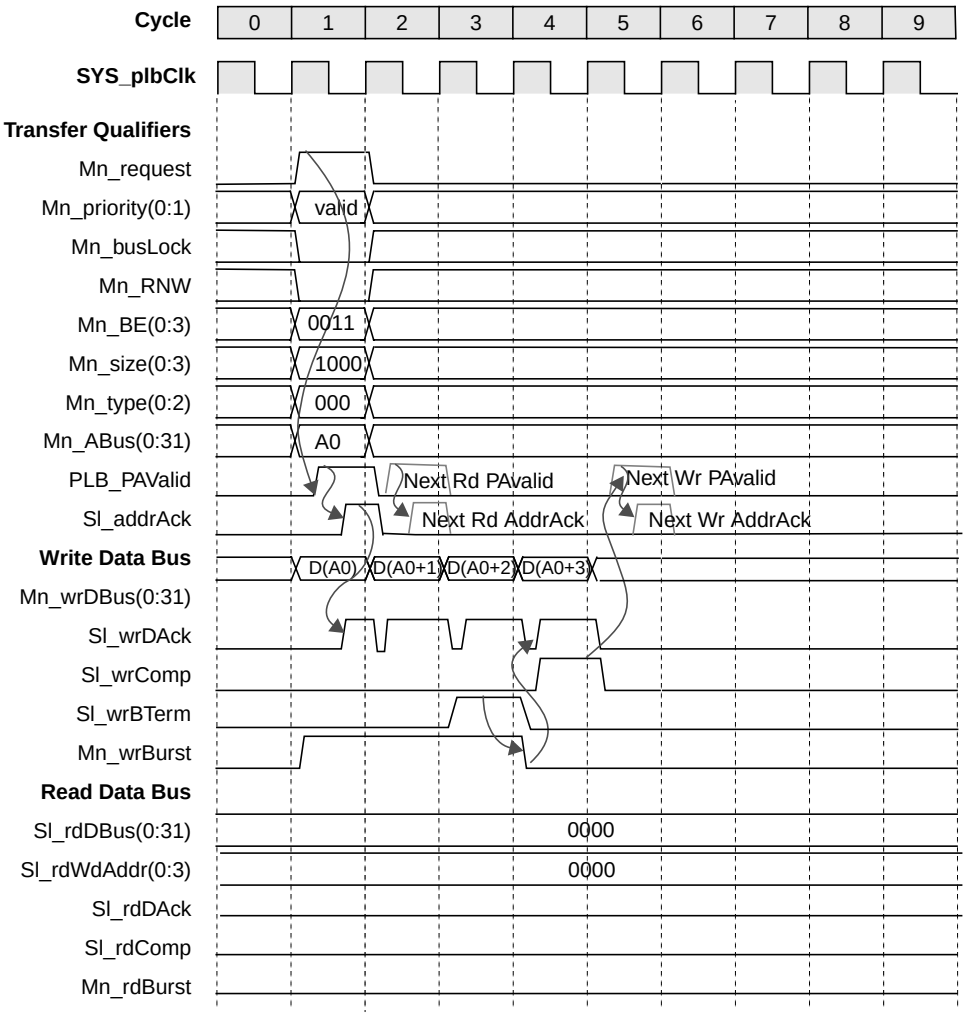


Figure 22. Fixed Length Burst Write Transfer

5.1.17 Back-to-Back Burst Read - Burst Write Transfers

Figure 23 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.

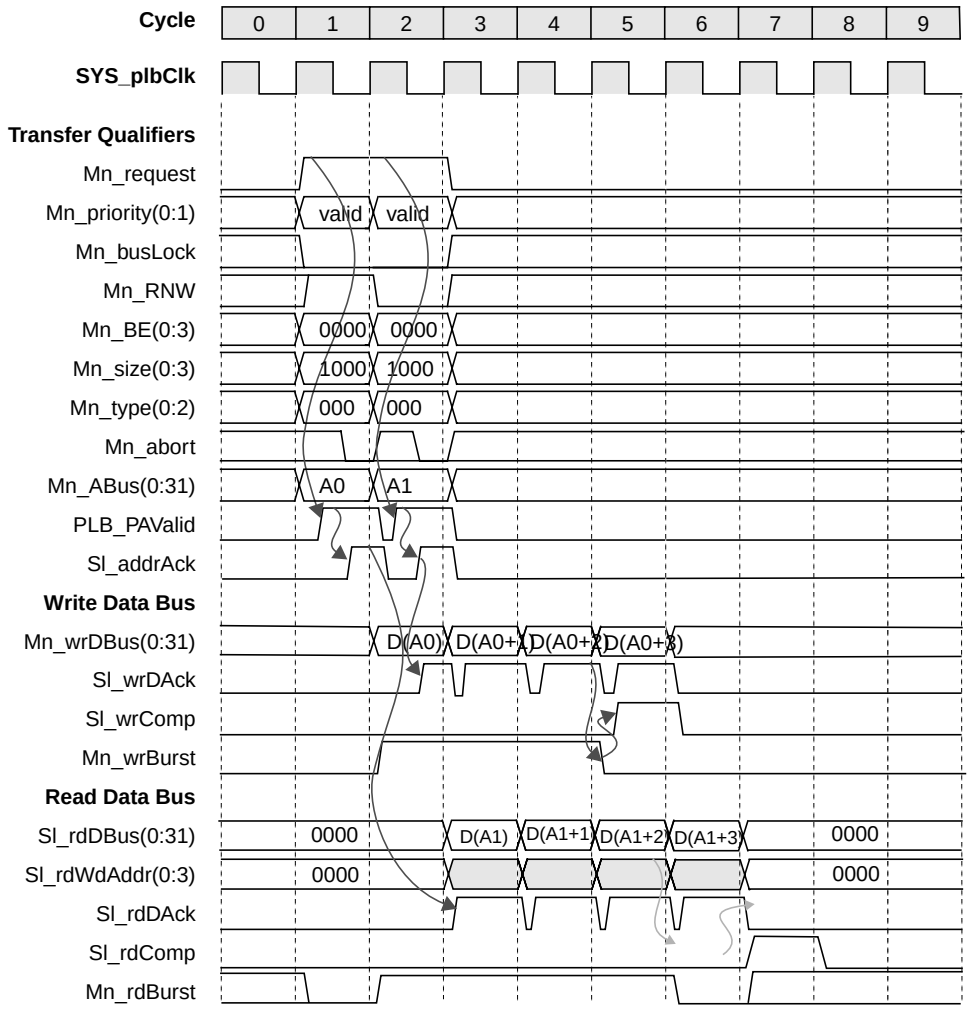


Figure 23. Back-to-Back Burst Read - Burst Write Transfers (Wait = 0, Hold =

5.1.18 Locked Transfer

Figure 24 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_PAValid only after detecting that both data busses are idle. The slave then asserts the Sl_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.

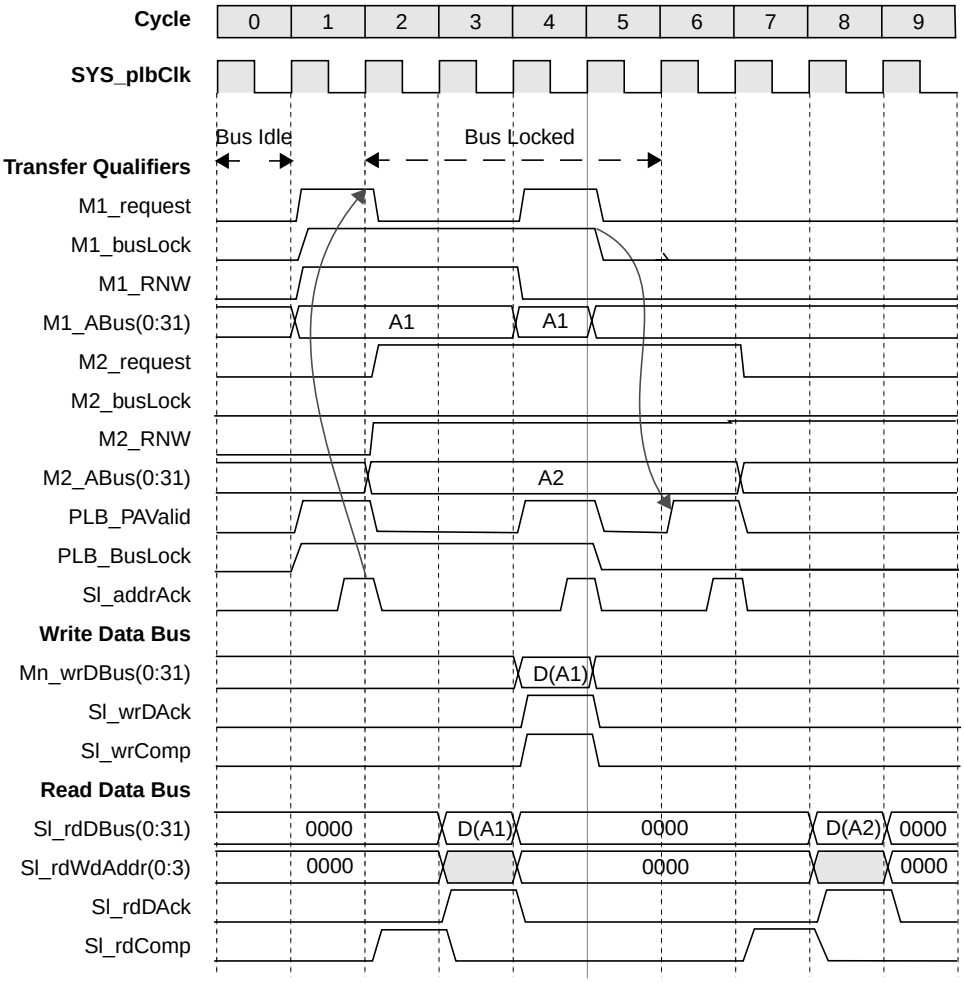


Figure 24. Locked Transfer

5.1.19 Slave Requested Re-arbitration With Bus Unlocked

Figure 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 Sl_rearbitrate signal to request re-arbitration of the PLB bus. In response to the assertion of the Sl_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_PAValid 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 Sl_rearbitrate signal was asserted.

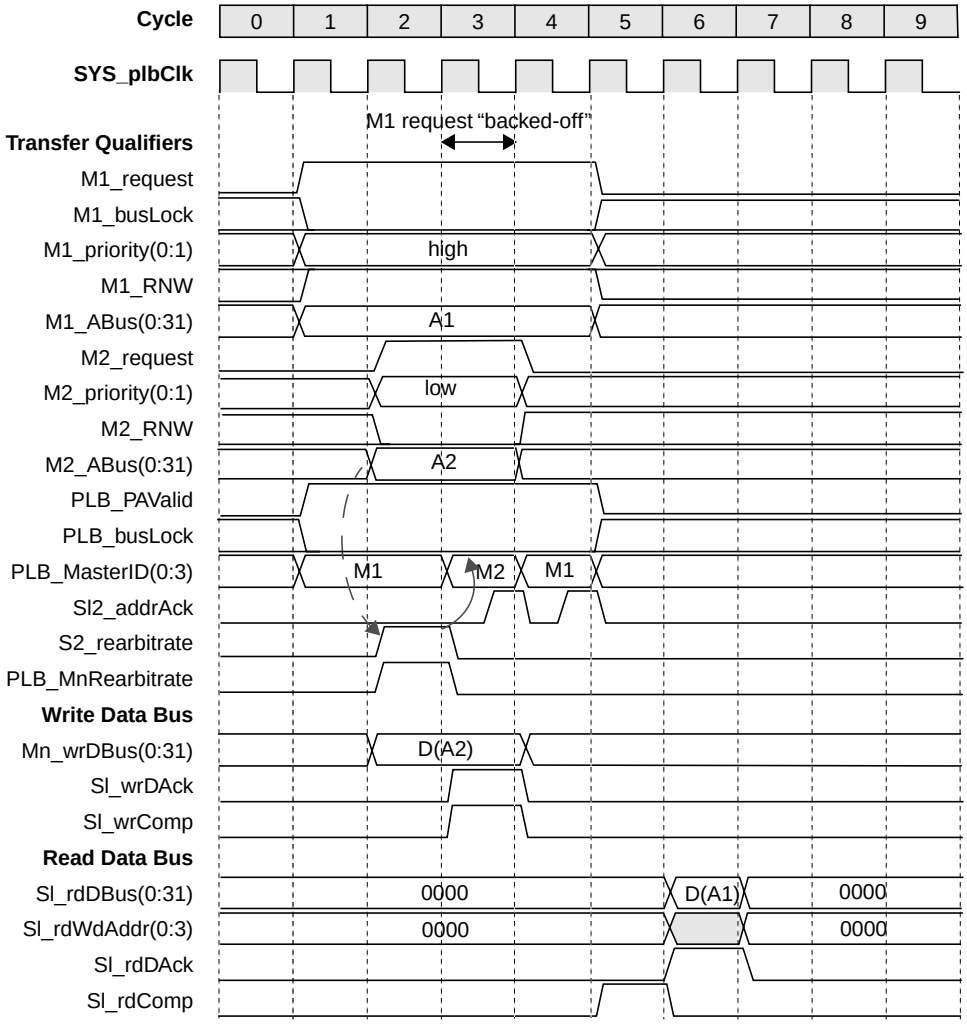


Figure 25. Slave Requested Re-arbitration With Bus Un-locked

5.1.20 Slave Requested Re-arbitration With Bus Locked

Figure 26 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 Sl_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.

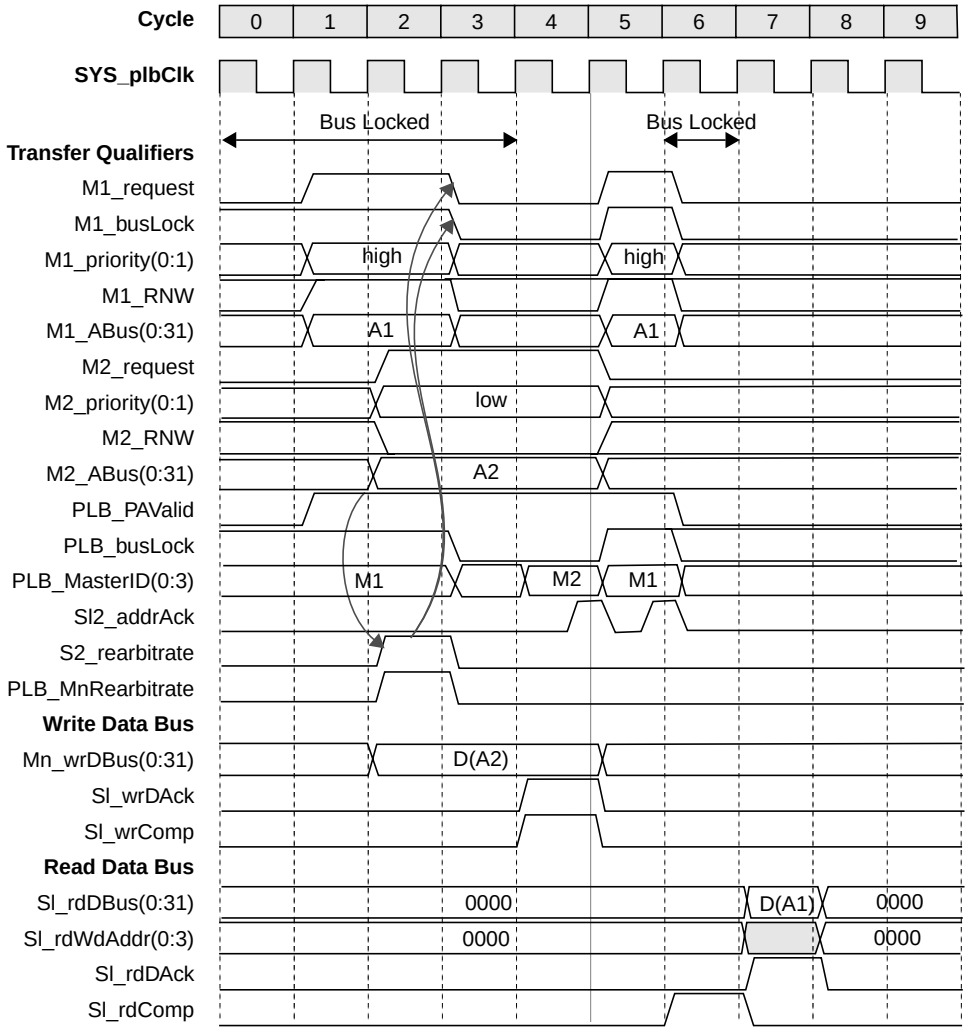


Figure 26. Slave Requested Re-arbitration With Bus Locked

5.1.21 Bus Time-Out Transfer

Figure 27 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_PAValid 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.

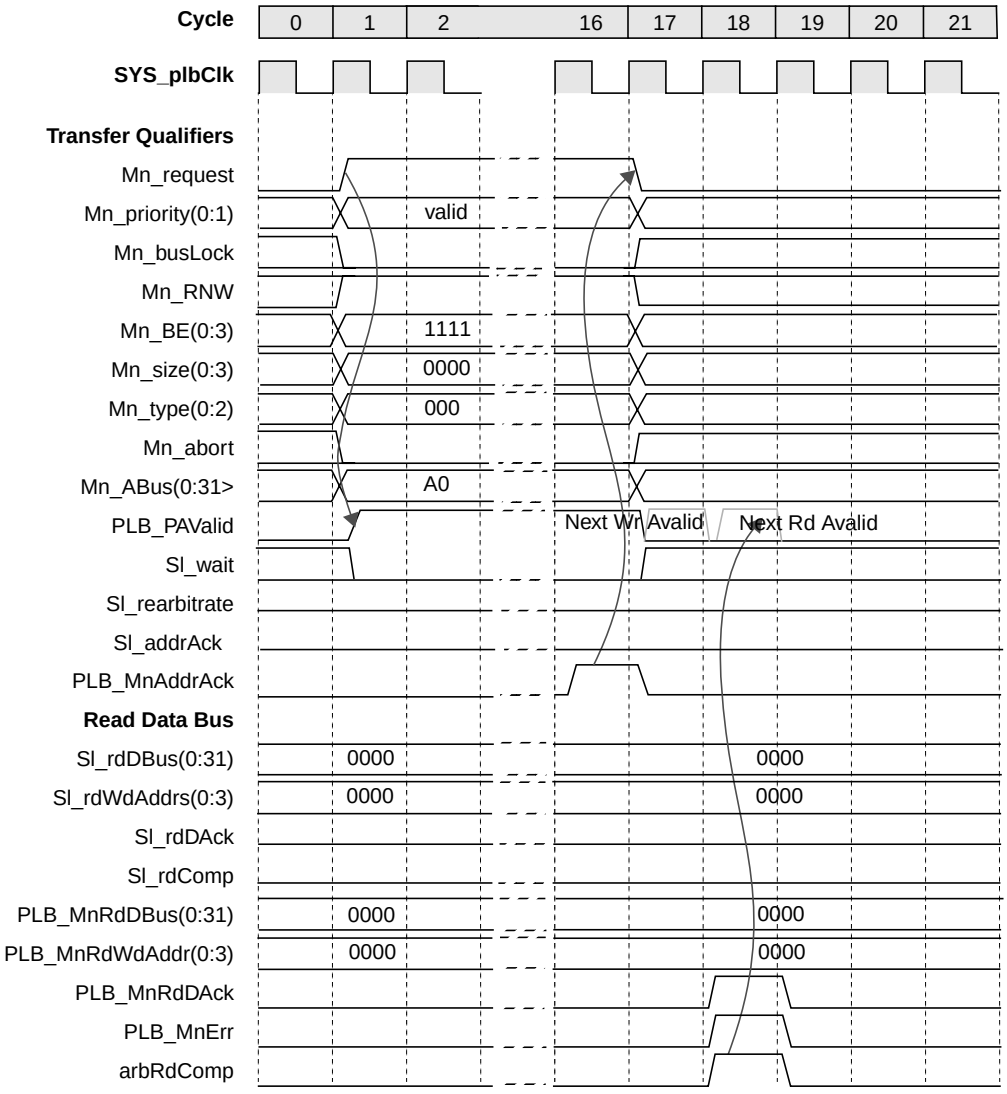


Figure 27. Bus Time-Out Transfer

5.1.22 Bus Transfer Time-Out Notes

As mentioned previously, once PLB_PAValid has been asserted for a master request, the PLB arbiter will wait for sixteen clock cycles for the assertion of either Sl_rearbitrate, Sl_wait, Sl_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.

5.2 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.

5.2.1 Pipelined Back-to-Back Read Transfers

Figure 28 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 Sl_rdComp signal in the clock cycle preceding the Sl_rdDAck. 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 Sl_rdDAck for the data read until two clock cycles following the assertion of the corresponding Sl_addrAck. For the secondary read requests, the slave may not assert its Sl_rdDAck or drive the Sl_rdDBus until two clock cycles following the assertion of PLB_rdPrim. 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.

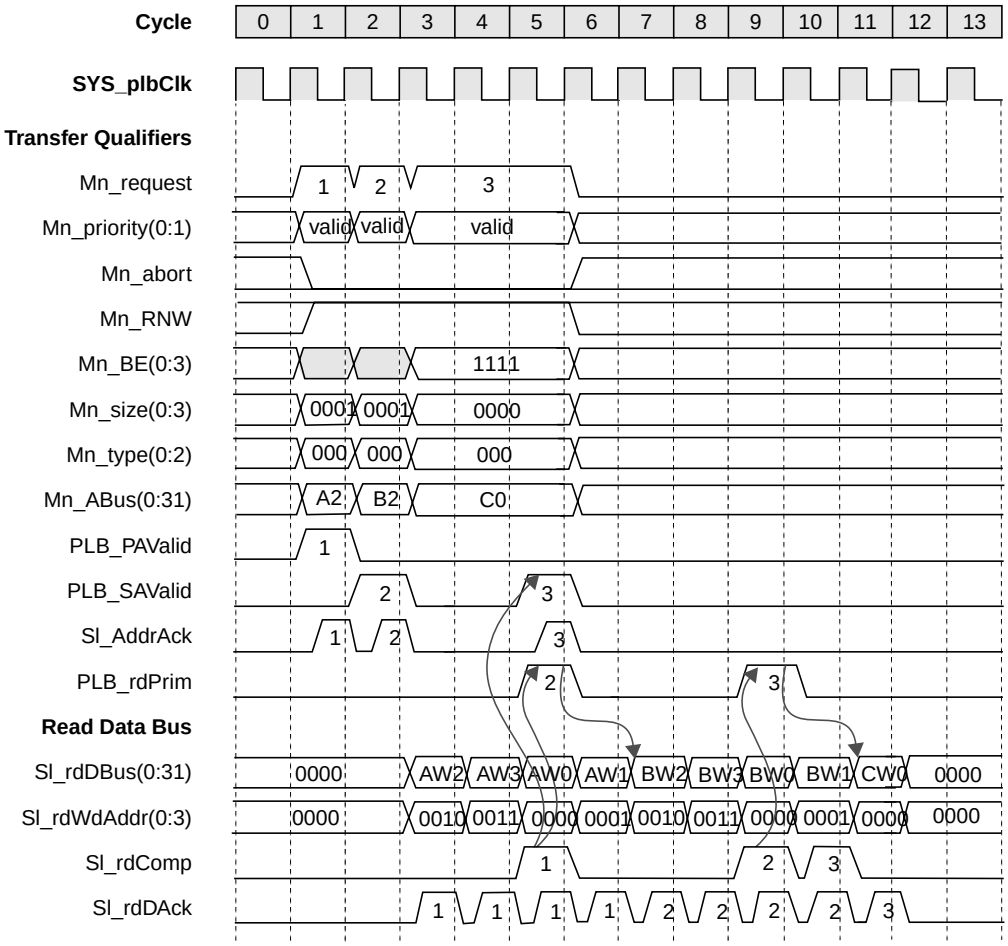


Figure 28. Pipelined Back-to-Back Read Transfers

5.2.2 Pipelined Back to Back Read Transfers - Delayed AAck

Figure 29 is similar to Figure 28, with one exception. For master 3's request, PLB_SAValid is negated and PLB_PAValid is asserted prior to the slave's assertion of Sl_addrAck. Note that the assertion of PLB_PAValid for the last read request is made possible by the assertion of Sl_rdComp for the previous secondary request. Note also that PLB_rdPrim is not asserted for this request.

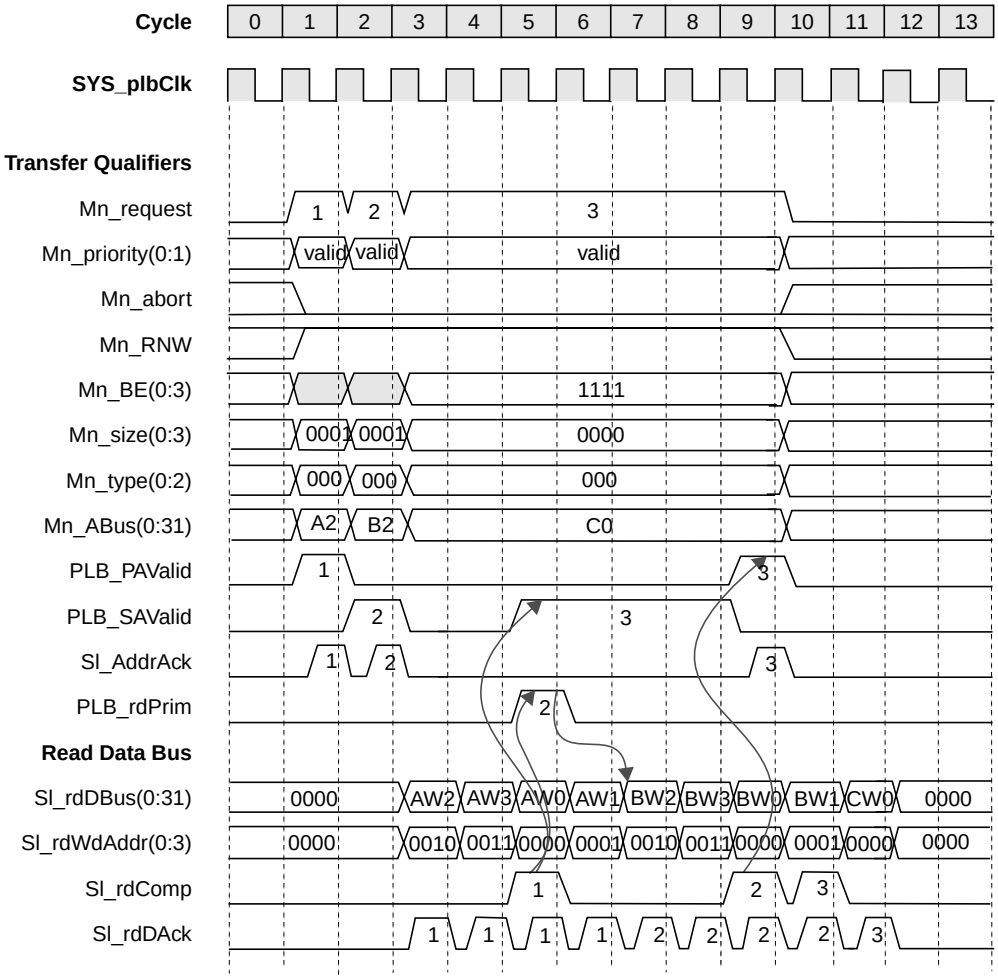


Figure 29. Pipelined Back-to-Back Read Transfers - Delayed AAck

5.2.3 Pipelined Back-to-Back Write Transfers

Figure 30 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 Sl_wrComp and Sl_wrDAck signals in the same clock cycle Sl_addrAck is asserted. For the secondary write requests, the slave may not assert its Sl_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.

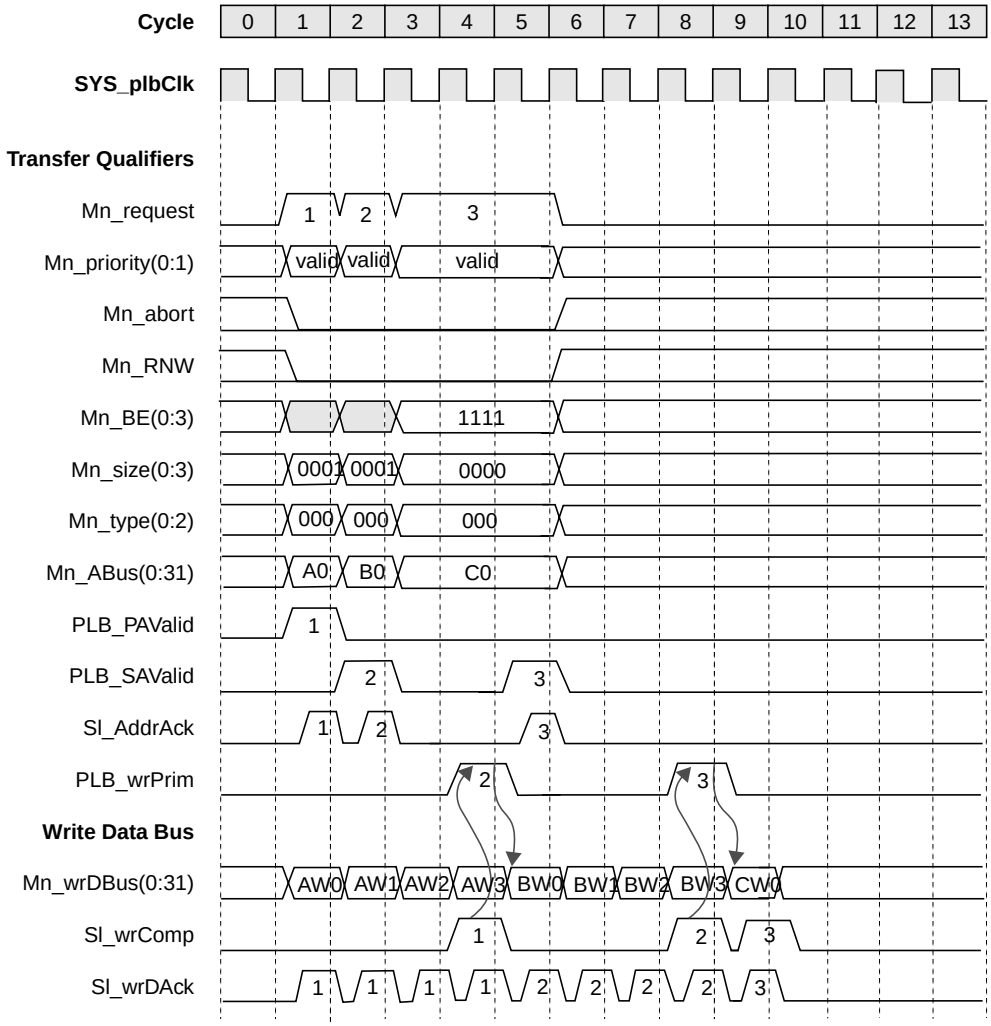


Figure 30. Pipelined Back-to-Back Write Transfers

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

Figure 31 is similar to Figure 30, with one exception. For master 3's write request in the series, PLB_SAValid is negated and PLB_PAValid is asserted prior to the slave's assertion of Sl_addrAck. Note that the assertion of PLB_PAValid for the last write request is made possible by the assertion of Sl_wrComp for the previous secondary request. Note also that PLB_wrPrim is not asserted for this request.

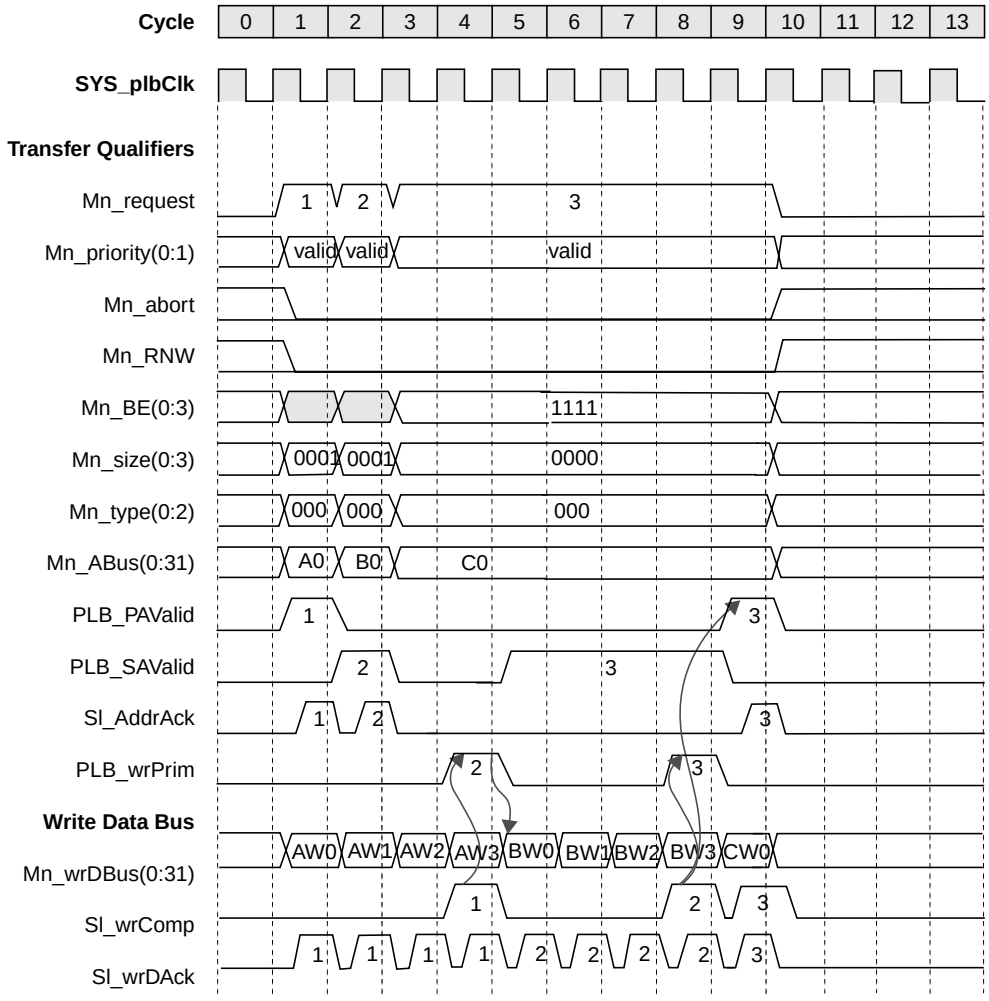


Figure 31. Pipelined Back-to-Back Write Transfers - Delayed AAck

5.2.5 Pipelined Back-to-Back Read and Write Transfers

Figure 32 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.

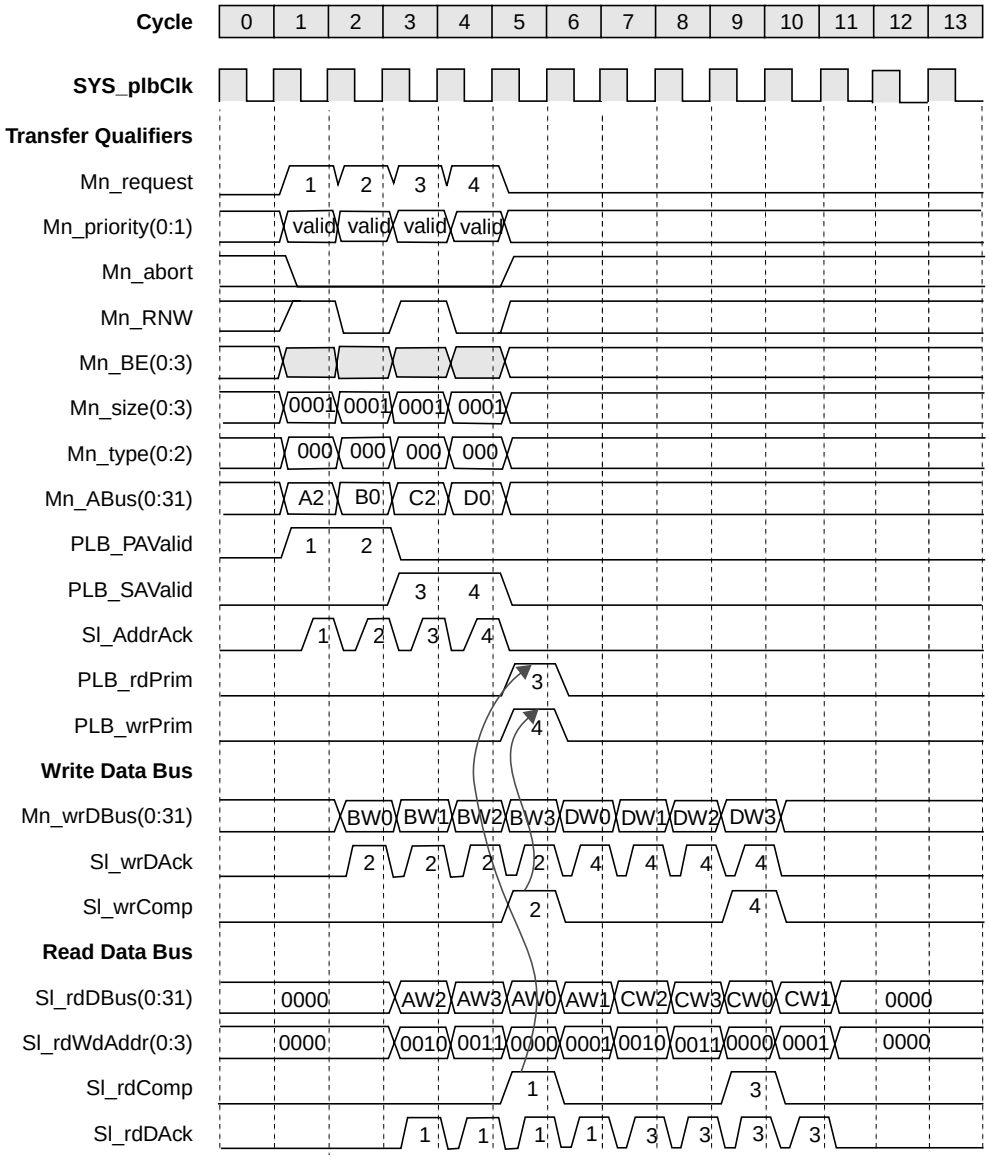


Figure 32. Pipelined Back-to-Back Read and Write Transfers

5.2.6 Pipelined Back-to-Back Read Burst Transfers

Figure 33 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.

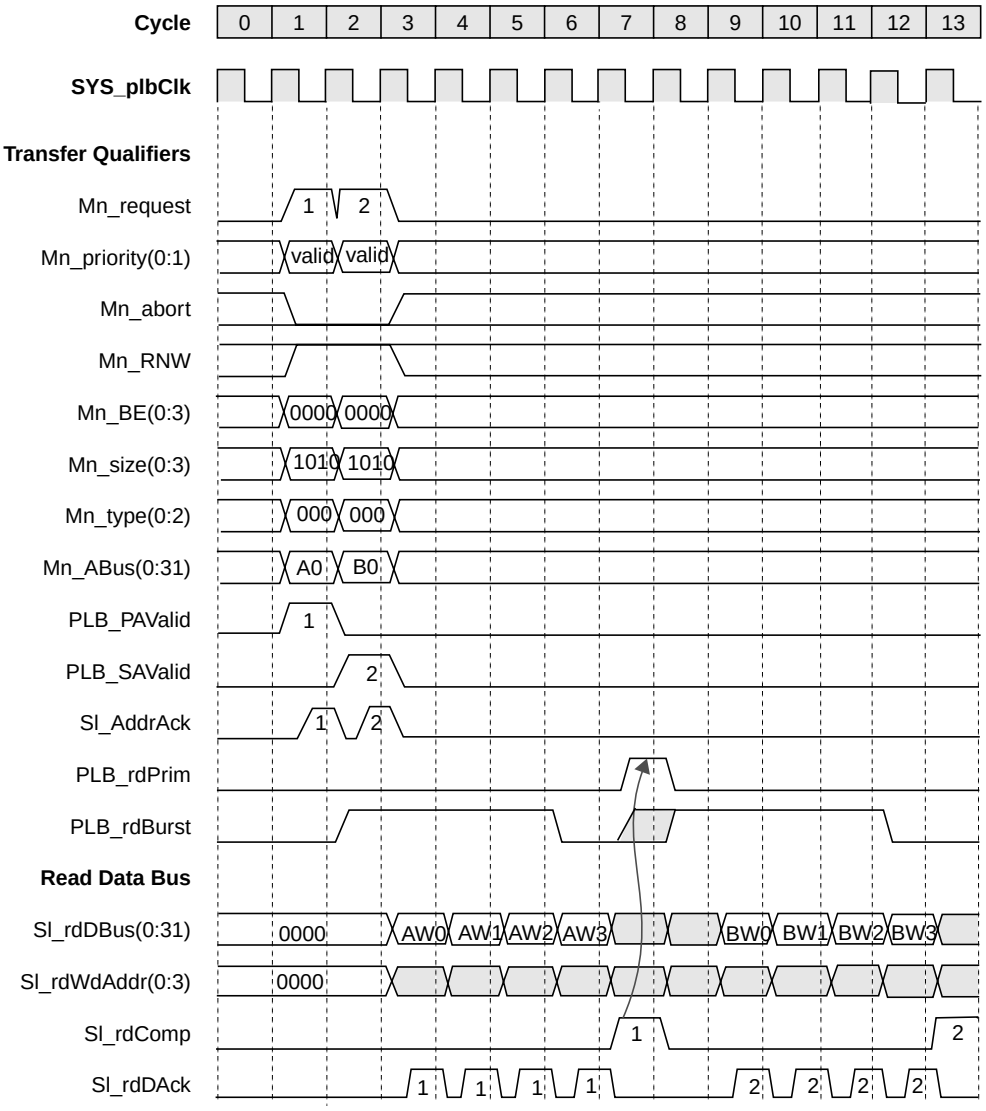


Figure 33. Pipelined Back-to-Back Read Burst Transfers

5.2.7 Pipelined Back-to-Back Write Burst Transfers

Figure 34 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.

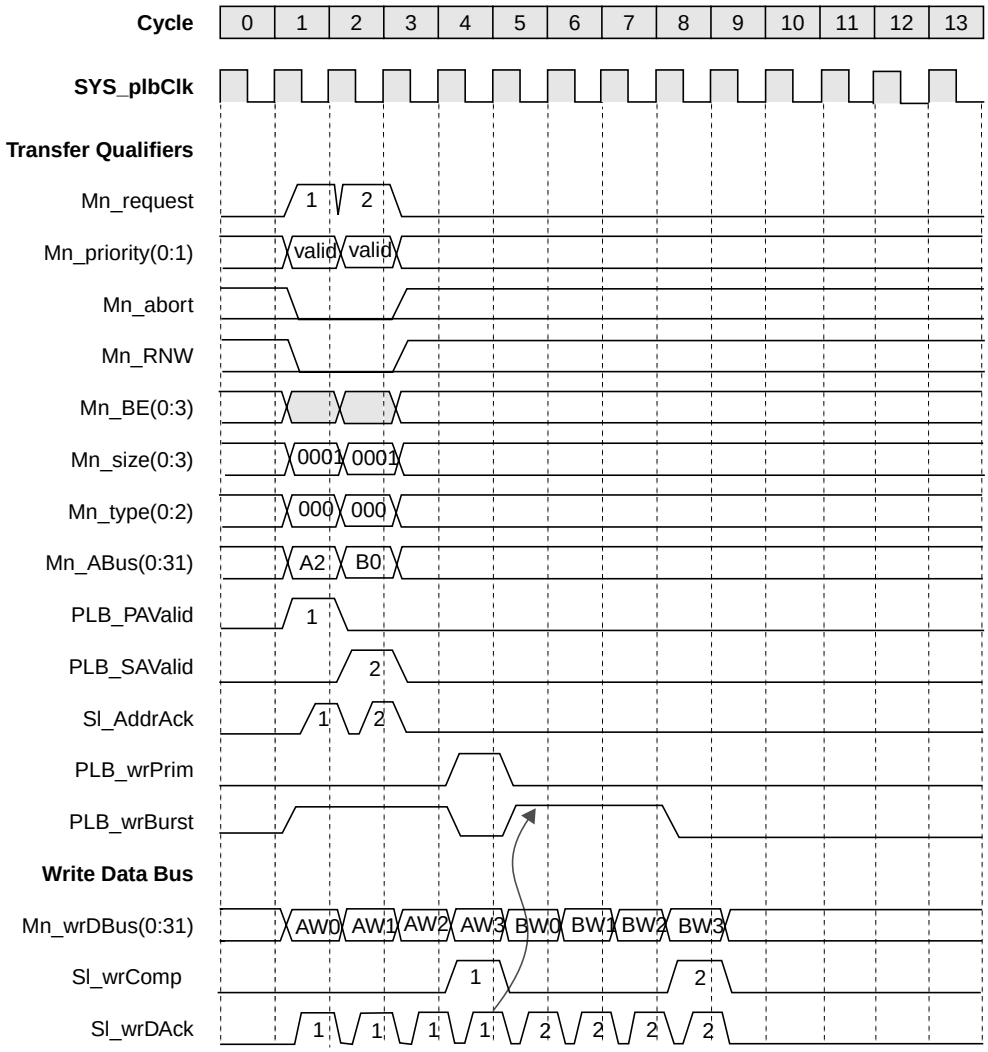


Figure 34. Pipelined Back-to-Back Write Burst Transfers

5.3 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.

5.3.1 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 Sl_rdComp or Sl_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.

Index

R

S

T

The information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change IBM's product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as illustration. The results obtained in other operating environments may vary.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for any damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY
12533-6531

The IBM home page can be found at http://www.ibm.com

The IBM Microelectronics Division home page can be found at
http://www.chips.ibm.com

Document No. SA-14-2531-01