128-Bit Processor Local Bus Architecture Specifications Version 4.7

May 2, 2007

Printed in the United States of America May 2007

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**

The IBM Semiconductor Solutions home page can be found at **ibm.com/chips**

title.fm.1.0
May 2, 2007

Contents

List of Figures

List of Tables

Revision Log

Revision Date	Pages	Description
May 2, 2007	—	Initial release

About This Book

This book begins with an overview of the IBM® 128-bit processor local bus (PLB). Following the overview is detailed information about PLB signals, interfaces, timing, and operations.

The PLB has the following features:

- Overlapping of read and write transfers allows two data transfers per clock cycle for maximum bus usage.
- 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.
- Hidden (overlapped) bus request protocol and bus grant protocol reduce arbitration latency.
- 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.
- A bus arbitration locking mechanism allows master-driven atomic operations.
- Byte-enable capability allows unaligned transfers and byte transfers.
- Support for 16-byte, 32-byte, 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, word, and doubleword 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.
- Direct memory access (DMA) buffered, flyby, peripheral-to-memory, memory-to-peripheral, and DMA memory-to-memory operations are also supported.

Who Should Use This Book

This book is for hardware, software, and application developers who need to understand core and application-specific integrated circuit (ASIC) (called Core+ASIC in this document) development and system-on-a-chip designs. The audience must understand embedded system design, operating systems, and the principles of computer organization.

Related Publications

The following publications contain related information:

- *On-Chip Peripheral Bus Architecture Specifications*
- *Device Control Register Bus Architecture Specifications*
- *Processor Local Bus Toolkit User's Manual*
- *OPB Bus Functional Model Toolkit User's Manual*
- *Device Control Register Bus Toolkit User's Manual*
- *32-Bit OPB Arbiter Core User's Manual*
- *128-Bit OPB to PLB Bridge Core User's Manual*
- *64-Bit PLB Arbiter Core User's Manual*
- *128-bit PLB Arbiter Core User's Manual*
- *64-Bit PLB to OPB Bridge Core User's Manual*
- *128-bit PLB to OPB Bridge Core User's Manual*

Conventions and Notations Used in this Manual

Binary values in sentences and appear in single quotation marks. For example: '1010'.

How This Book is Organized

This book is organized as follows:

- *PLB Overview* on page 17
- *PLB Signals* on page 23
- *PLB Interfaces* on page 63
- *PLB Timing Guidelines* on page 67
- *PLB Operations* on page 83
- *Double Data Rate Protocol* on page 165

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

- *Contents* on page 3
- *List of Figures* on page 9
- *List of Tables* on page 11
- *Index* on page 171

1. PLB Overview

The processor local bus (PLB) is a high-performance 64-bit address bus and a 128-bit data bus. The PLB provides a standard interface between the processor cores and integrated bus controllers. With the PLB, a library of processor cores and bus controllers can be developed for use in core and application-specific integrated circuits (called Core+ASIC in this document) and system-on-a-chip (SOC) designs.

In addition, the PLB is a high-performance on-chip bus which is used in highly integrated Core+ASIC systems. The PLB supports read and write data transfers between master devices and slave devices that are equipped with a PLB bus interface and are connected through PLB signals.

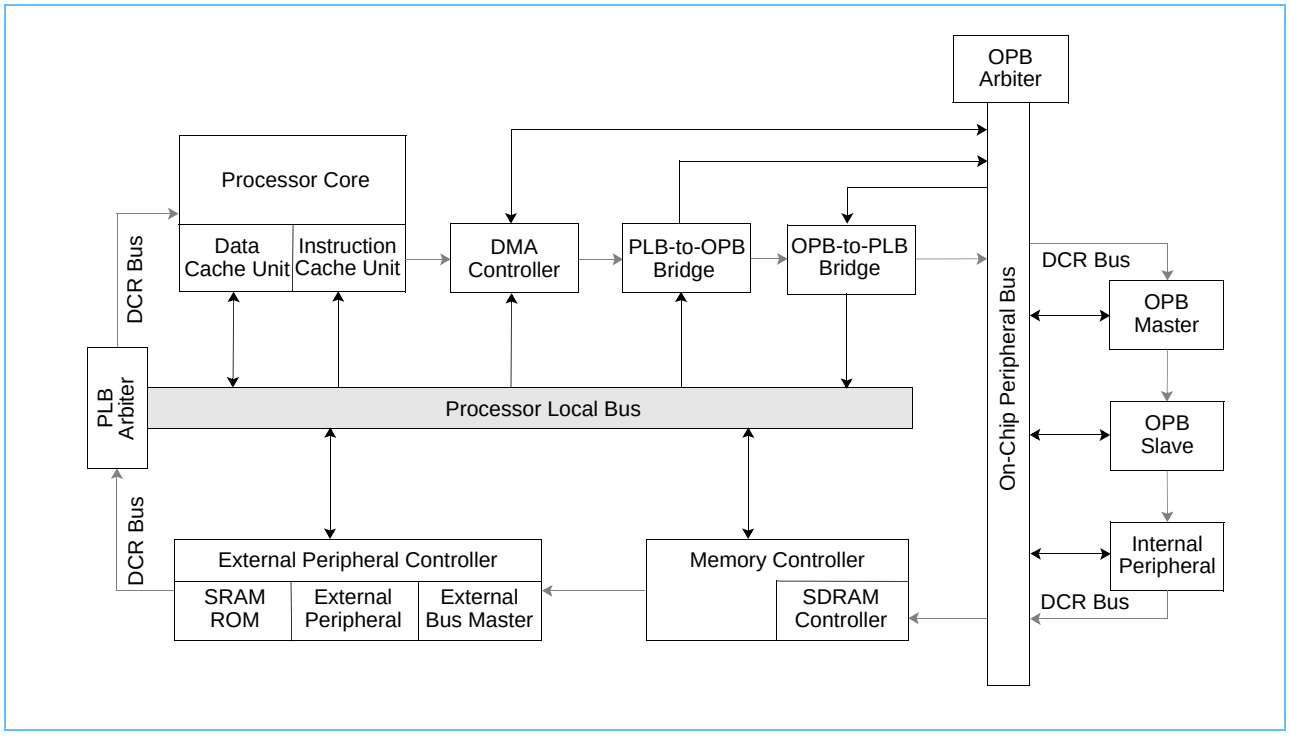
Each PLB master is attached to the PLB through separate address buses, read data buses, write data buses, and transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address buses, read data buses, write data buses, and transfer control and status signals for each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

The PLB grants access 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. Also, an arbitration locking mechanism is provided to support master-driven atomic operations.

The PLB is a fully-synchronous bus. A single clock source provides timing for all PLB signals. All masters and slaves that are attached to the PLB share this clock source.

Figure 1-1 demonstrates how the PLB is interconnected for the purpose of Core+ASIC development or SOC design.

Figure 1-1. Processor Local Bus Interconnections



As *Figure 1-1 Processor Local Bus Interconnections* on page 17 shows, the on-chip bus structure provides a link between the processor core and other peripherals. The on-chip bus structure consist of PLB master devices, on-chip peripheral bus (OPB) master devices, and slave devices.

The PLB is the high-performance bus that is used to access memory through the bus interface units. The external peripheral controller and memory controller are the PLB slaves. These slaves are illustrated in *Figure 1-1*. The processor core has two PLB master connections, one for the instruction cache and one for the data cache. The direct memory access (DMA) controller is a PLB master device that is used in data intensive applications to improve the transfer performance of data.

Lower performance peripherals, such as OPB masters, OPB slaves, and other internal peripherals, are attached to the OPB. A bridge between the PLB and the OPB enables PLB masters to transfer data to OPB slaves and from OPB slaves. *Figure 1-1* shows two bridges. The first bridge is a PLB-to-OPB bridge, which is a slave on the PLB. The other bridge is an OPB-to-PLB bridge, which is a slave on the OPB and a master on the PLB. OPB peripherals can also consist of DMA peripherals.

The device control register (DCR) bus is used primarily for accessing status and control registers within the various PLB and OPB masters and slaves. It is meant to off-load the PLB from the lower performance, status- and- control read-and-write transfers. The DCR bus architecture allows data transfers among OPB peripherals to occur independently from, and concurrent with, data transfers between the processor and memory, or among other PLB devices.

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

In this category, the PLB includes the following features:

- Read and write transfers overlap to allow two data transfers per clock cycle for maximum bus usage.
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth.
- Extendable address pipelining reduces overall bus latency by allowing the latency that is associated with a new request to be overlapped with an ongoing data transfer in the same direction.
- Hidden (overlapped) bus request protocol and bus grant protocol reduces arbitration latency.
- The PLB is a fully synchronous bus.
- The PLB supports double data rate (DDR) transfers for 128-bit devices that choose to implement DDR. See *Section 6 Double Data Rate Protocol* on page 165 for more information about DDR.

1.1.2 System Design Flexibility

In this category, the PLB includes the following features:

- Bus architecture supports up to sixteen masters and any number of slave devices.
- Four levels of request priority for each master allow PLB implementations with various arbitration schemes.
- 32-bit, 64-bit, 128-bit data bus implementations.

- Bus arbitration-locking mechanism allows for master-driven atomic operations.
- Byte-enable capability allows for unaligned transfers and odd-byte transfers.
- Support for 16-byte, 32-byte, 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.
- Optional parity support provides enhanced data protection where necessary.

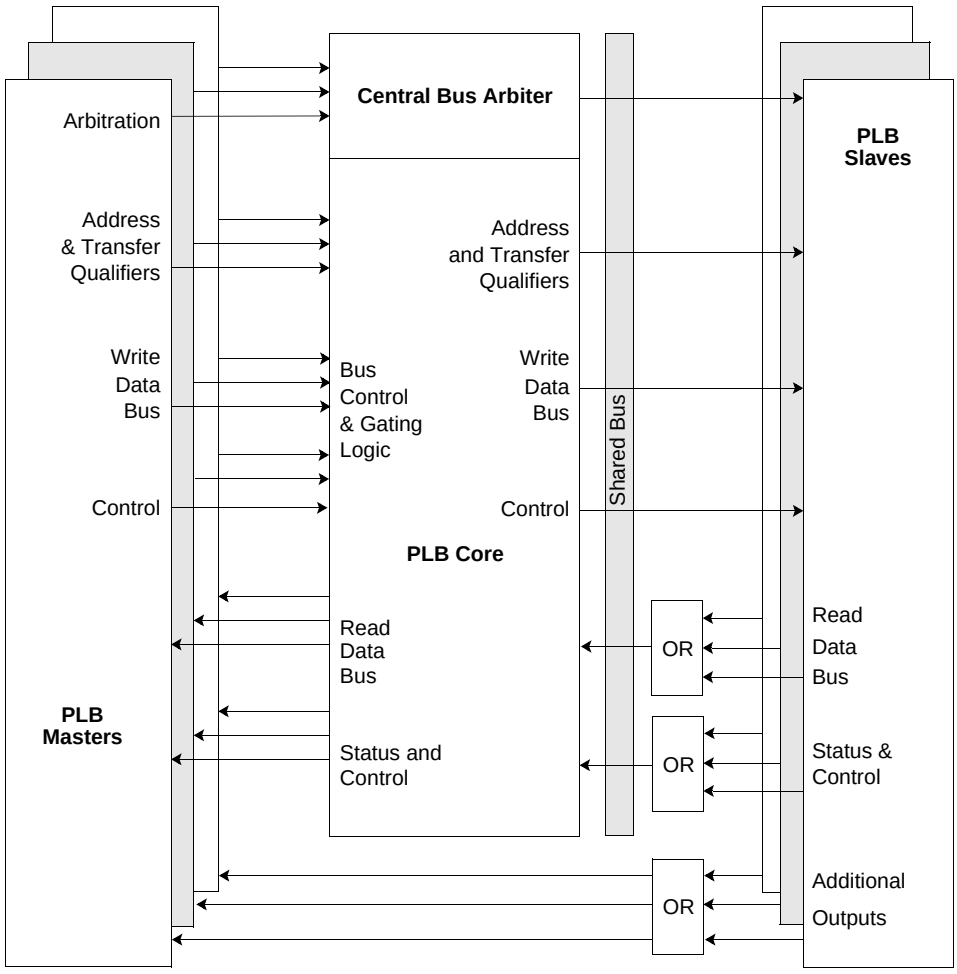
1.2 PLB Implementation

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 fewer than sixteen masters are allowed. The PLB architecture also supports any number of slave devices. However, the number of masters and slaves that are attached to a PLB core in a particular system directly affects the performance of the PLB core in that system.

Figure 1-2 PLB Interconnects on page 20 shows an example of the PLB connections for a system with three masters and three slaves.

Figure 1-2. PLB Interconnects



1.3 PLB Transfer Protocol

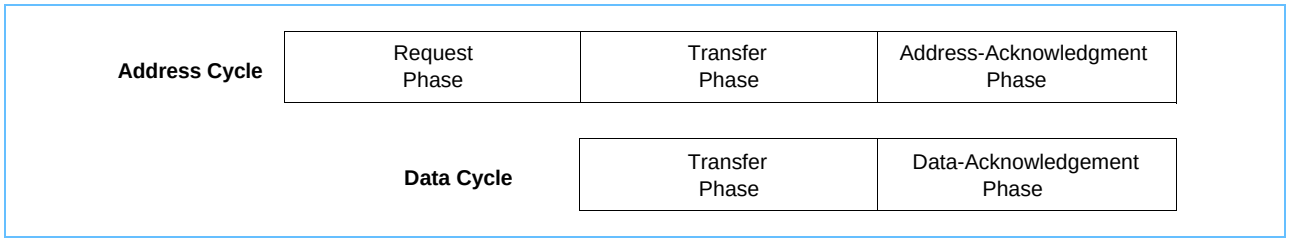
Figure 1-3 PLB Address and Data Cycles on page 21 shows that a PLB transaction is grouped under an address cycle and a data cycle.

The address cycle has three phases: request, transfer, and address acknowledgment. A PLB 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. After the PLB arbiter has granted bus ownership, the address and transfer qualifiers for the master are presented to the slave devices during the transfer phase. During normal operation, the address cycle is terminated by a slave latching the address and transfer qualifiers for the master during the address acknowledgment phase.

Each data beat in the data cycle has two phases: transfer and data acknowledgment. During the transfer phase, the master drives the write data bus for a write transfer or samples the read data bus for a read transfer. Data acknowledgment signals are required during the data acknowledgment phase for each data beat in a data cycle.

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

Figure 1-3. PLB Address and Data Cycles

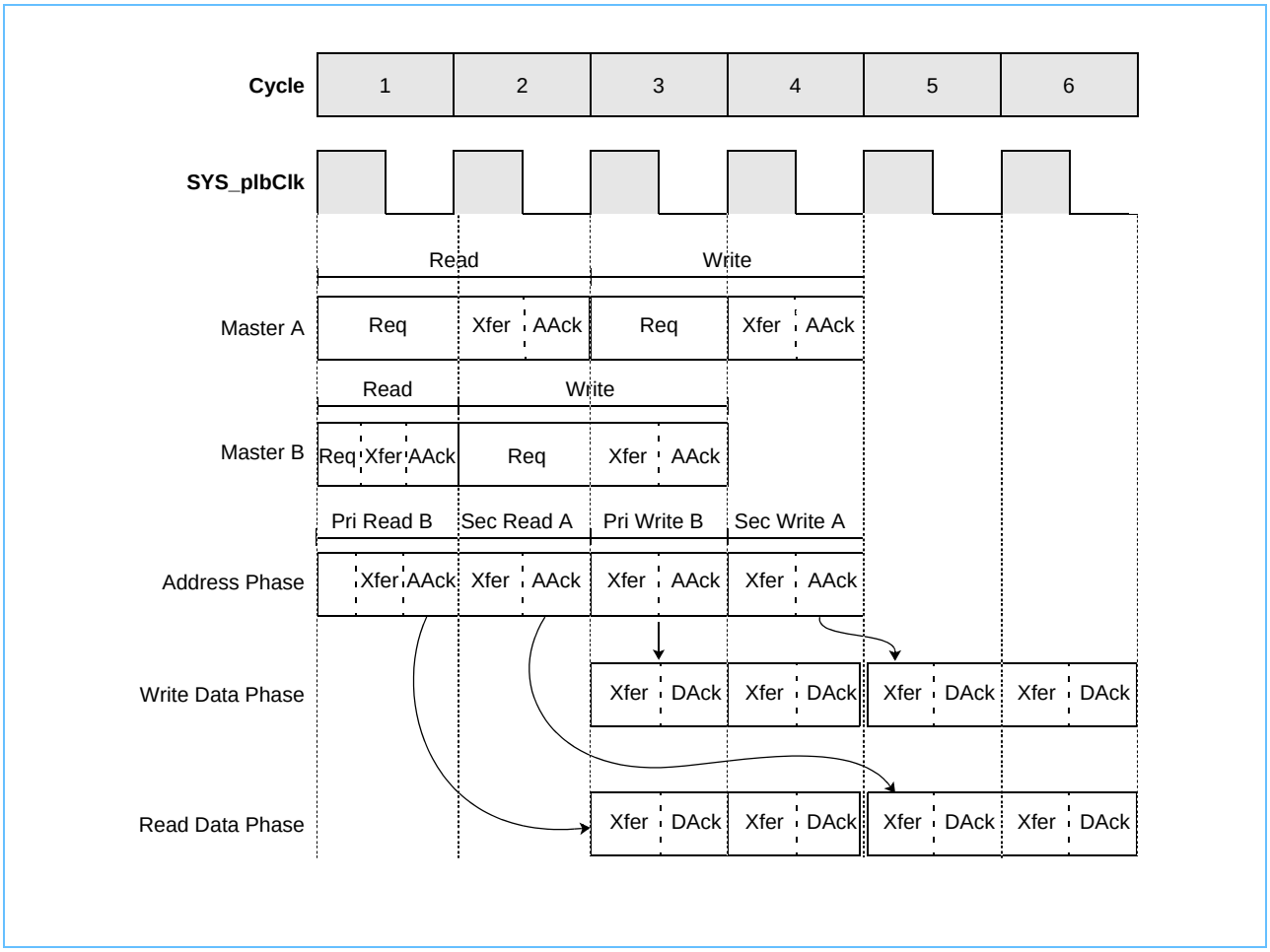


1.4 Overlapped PLB Transfers

Figure 1-4 Overlapped PLB Transfers on page 22 shows an example of overlapped PLB transfers. PLB address buses, read data buses, 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 that is associated with a new transfer request to be overlapped with an ongoing data transfer in the same direction.

Figure 1-4. Overlapped PLB Transfers



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

2. PLB Signals

Processor local bus (PLB) signals consist of the following categories:

- PLB System Signals
- PLB Arbitration Signals
- PLB Status Signals
- PLB Transfer Qualifier Signals
- PLB Read Data Bus Signals
- PLB Write 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 that are outputs of the PLB core and inputs to the slave devices are prefixed with PLB_. There is only one output of the PLB core for each one of these signals. Each slave that is attached to the PLB receives this output as an input. For example, the PLB_PAValid signal is an output of the PLB core and is an input to each slave that is attached to the PLB core.
- Signals that are outputs of the slaves and inputs to the PLB core are prefixed with Sl_. Each slave has its own output which is then logically ORed 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 signals, read data signals, and write data signals. These signals use the following naming convention:

- Signals that are driven by a master as an input to the PLB core are prefixed with Mn_. There can be as many as sixteen masters with their own set of PLB input signals. For example, when it is implemented, the Mn_request signal results in M0_request, M1_request, through M15_request.
- Signals that are driven by the PLB core to a master have the prefix, PLB_Mn. The PLB_Mn prefix indicates that this signal is connected from the PLB core to a specific master. 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 results in PLB_M0AddrAck, PLB_M1AddrAck, through PLB_M15AddrAck.
- Signals that are driven by bus logic that is external to the PLB core to a master have the prefix, PLB_Msignalname(0:n). This prefix indicates that this signal is connected from the PLB bus logic to a specific master. This logic is typically OR logic gathering PLB_MBusy, PLBMRdErr, PLB_MWrErr, and PLB_MIRQ signals into a vectored bus signal.

Note: The PLB architecture 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 outputs and master outputs must be prefixed by a 3-letter qualifier identifying the unit. In its current version, the PLB architecture allows a maximum of sixteen masters. However, this does not preclude the implementation of PLB cores capable of supporting fewer than sixteen masters.

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

Table 2-1. Summary of PLB Signals (Sheet 1 of 3)

Signal Name	Interface	I/O	Description	Page
Mn_abort	Master n	I	Master n abort bus request indicator	32
Mn_ABus(0:31)	Master n	I	Master n address bus	46
Mn_ABusPar	Master n	I	Master n address bus parity	47
Mn_ABusParEn	Master n	I	Master n address bus parity enable	47
Mn_BE	Master n	I	Master n byte enables	35
Mn_BEPar	Master n	I	Master n byte enables parity	41
Mn_BEParEn	Master n	I	Master n byte enables parity enable	41
Mn_busLock	Master n	I	Master n bus lock	28
Mn_lockErr	Master n	I	Master n lock error indicator	46
Mn_MSize(0:1)	Master n	I	Master data bus size	43
Mn_priority(0:1)	Master n	I	Master n bus request priority	28
Mn_rdBurst	Master n	I	Master n burst read transfer indicator	51
Mn_rdDBusParErr	Master n	I	Master n read data bus parity error	49
Mn_request	Master n	I	Master n bus request	28
Mn_RNW	Master n	I	Master n read/not write	35
Mn_size(0:3)	Master n	I	Master n transfer size	41
Mn_TAttribute(0:15)	Master n	I	Master n transfer attribute bus	44
Mn_type(0:2)	Master n	I	Master n transfer type	42
Mn_UABus(0:31)	Master n	I	Master n upper address bus	47
Mn_UABusPar	Master n	I	Master n upper address bus parity	47
Mn_UABusParEn	Master n	I	Master n upper address bus parity enable	47
Mn_wrBurst	Master n	I	Master n burst write transfer indicator	57
Mn_wrDBus	Master n	I	Master n write data bus	55
Mn_wrDBusPar	Master n	I	Master n write data bus parity	55
Mn_wrDBusParEn	Master n	I	Master n write data bus parity enable	56
PLB_abort	Arbiter	O	PLB abort bus request indicator	32
PLB_ABus(0:31)	Arbiter	O	PLB address bus	46
PLB_ABusPar	Arbiter	O	PLB address bus parity	47
PLB_ABusParEn	Arbiter	O	PLB address bus parity enable	47
PLB_BE	Arbiter	O	PLB byte enables	35
PLB_BEPar	Arbiter	O	PLB byte enables parity	41
PLB_BEParEn	Arbiter	O	PLB byte enables parity enable	41
PLB_busLock	Arbiter	O	PLB bus lock	28

Table 2-1. Summary of PLB Signals (Sheet 2 of 3)

Signal Name	Interface	I/O	Description	Page
PLB_lockErr	Arbiter	O	PLB lock error indicator	46
PLB_masterID	Arbiter	O	PLB current master identifier	34
PLB_MBusy(n)	Master n	O	PLB master n slave busy indicator	59
PLB_MIRQ(n)	Master n	O	PLB master n slave interrupt indicator	60
PLB_MRdErr(n)	Master n	O	PLB master n slave read error indicator	59
PLB_MWrErr(n)	Master n	O	PLB master n slave write error indicator	59
PLB_Mn_WrBTerm	Master n	O	PLB master n terminate write burst indicator	58
PLB_Mn_WrDAck	Master n	O	PLB master n write data acknowledgment	56
PLB_MnAddrAck	Master n	O	PLB master n address acknowledgment	32
PLB_MnRdBTerm	Master n	O	PLB master n terminate read burst indicator	53
PLB_MnRdDAck	Master n	O	PLB master n read data acknowledge	51
PLB_MnRdDBus	Master n	O	PLB master n read data bus	48
PLB_MnRdDBusPar	Master n	O	PLB master n read data bus parity	49
PLB_MnRdDBusParEn	Master n	O	PLB master n read data bus parity enable	49
PLB_MnRdWdAddr(0:3)	Master n	O	PLB master n read word address	49
PLB_MnRearbitrate	Master n	O	PLB master n bus rearbitrate indicator	32
PLB_MnSSize(0:1)	Master n	O	PLB slave data bus size	43
PLB_MnTimeout	Arbiter	O	PLB master n bus timeout	35
PLB_Msize(0:1)	Arbiter	O	PLB master data bus size	43
PLB_PAValid	Arbiter	O	PLB primary address valid indicator	29
PLB_rdBurst	Arbiter	O	PLB burst read transfer indicator	51
PLB_rdPendPri(0:1)	Arbiter	O	PLB pending read request priority	33
PLB_wrPendPri(0:1)	Arbiter	O	PLB pending write request priority	34
PLB_rdPendReq	Arbiter	O	PLB pending read bus request indicator	33
PLB_rdPrim	Arbiter	O	PLB secondary to primary read request indicator	54
PLB_reqPri(0:1)	Arbiter	O	PLB current request priority	34
PLB_RNW	Arbiter	O	PLB read not write	35
PLB_SAValid	Arbiter	O	PLB secondary address valid indicator	30
PLB_size(0:3)	Arbiter	O	PLB transfer size	41
PLB_TAttribute	Arbiter	O	PLB transfer attribute bus	44
PLB_type(0:2)	Arbiter	O	PLB transfer type	42
PLB_UABus(0:31)	Arbiter	O	PLB upper address bus	47
PLB_UABusPar	Arbiter	O	PLB upper address bus parity	47
PLB_UABusParEn	Arbiter	O	PLB upper address bus parity enable	47
PLB_wrBurst	Arbiter	O	PLB burst write transfer indicator	57
PLB_wrDBus	Arbiter	O	PLB write data bus	55
PLB_wrDBusPar	Arbiter	O	PLB write data bus parity	55

Table 2-1. Summary of PLB Signals (Sheet 3 of 3)

Signal Name	Interface	I/O	Description	Page
PLB_wrDBusParEn	Arbiter	O	PLB write data bus parity enable	56
PLB_wrPendReq	Arbiter	O	PLB pending write bus request indicator	33
PLB_wrPrim	Arbiter	O	PLB secondary to primary write request indicator	58
Sl_ABusParErr	Slave	I	Slave address bus parity error	60
Sl_addrAck	Slave	I	Slave address acknowledgment	32
Sl_MBusy(0:n)	Slave	I	Slave busy indicator	59
Sl_MRdErr(0:n)	Slave	I	Slave read error indicator	59
Sl_MWrErr(0:n)	Slave	I	Slave write error indicator	59
Sl_MIRQ(0:n)	Slave	I	Slave interrupt indicator	60
Sl_rdBTerm	Slave	I	Slave terminate read burst transfer	53
Sl_rdComp	Slave	I	Slave read transfer complete indicator	51
Sl_rdDAck	Slave	I	Slave read data acknowledgment	51
Sl_rdDBus	Slave	I	Slave read data bus	48
Sl_rdDBusPar	Slave	I	Slave read data bus parity	49
Sl_rdDBusParEn	Slave	I	Slave read data bus parity enable	49
Sl_rdWdAddr(0:3)	Slave	I	Slave read word address	49
Sl_rearbitrate	Slave	I	Slave rearbitrate bus indicator	32
Sl_SSize(0:1)	Slave	I	Slave data bus size	43
Sl_wait	Slave	I	Slave wait indicator	31
Sl_wrBTerm	Slave	I	Slave terminate write burst transfer	58
Sl_wrComp	Slave	I	Slave write transfer complete indicator	56
Sl_wrDAck	Slave	I	Slave write data acknowledgment	56
SYS_plbClk	System	I	System C2 clock	26
SYS_plbReset	System		System PLB reset	27
SYS_2xplbClk	System	I	System clock 2x the frequency of PLB clock (edge aligned). Only required for cores that support double data rate (DDR) protocol.	165

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 and the PLB arbiter. All PLB master output signals, slave output signals, and arbiter output signals are asserted or negated relative to the rising edge of the SYS_plbClk signal. All PLB master input signals, slave input signals, and arbiter input signals are sampled relative to this edge.

Note: The master and slave that are attached to the PLB are expected to operate at the frequency of the PLB. Thus, any matching speed that is required because of I/O constraints are handled in the PLB interfaces of masters and slaves. Likewise, any matching speed that is required because of units that run at different frequencies are handled in the PLB interfaces of masters and slaves. Processor cores that run at speeds significantly greater than that of the PLB 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 power-on reset signal for the PLB arbiter. 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 with the following characteristics:

- No read or write bus requests are pending; that is, all Mn_request signals are negated.
- The bus is not locked; that is, all Mn_busLock signals and PLB_busLock signals are negated.
- The bus is not granted or being granted to any master; that is, the PLB_PAValid signal is negated.
- The read and write data buses are not being used; that is, all Sl_rdDAck signals and Sl_wrDAck signals are negated and all Sl_rdDbus signals 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 the SYS_plbClk signal. The duration of the assertion when forcing the PLB to an idle state in a system depends on the actual implementation of the PLB arbiter, master, and slave devices of that system.

Note: In addition to the SYS_plbReset input, a PLB master can have other means by which it can force itself into a reset state without affecting the state of other masters and slaves that are attached to the PLB, or the PLB arbiter. However, if currently involved in a PLB transfer, the master must allow the transfer to be completed, or correctly terminate it by using the Mn_abort signal. Otherwise, if a slave acknowledges the request of a master, and the master needs to enter its reset state before all the data that is associated with that request is transferred, the master must tolerate receiving the data acknowledgments while entering, during, and after the reset state. Furthermore, the master must negate the Mn_busLock and Mn_rdBurst signals if they are 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.

When 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 Sl_wait signal of the slave and by the PLB arbiter address cycle timeout 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 in a single cycle arbitration implementation of the bus.

2.3.1 Mn_request (Bus Request)

The master asserts this signal to request a data transfer across the PLB. When the Mn_request signal has been asserted, this signal, the address, and all of the transfer qualifiers must retain their values until one of the following events occur:

- The slave terminates the address cycle.
- The master aborts the request.
- The PLB arbiter asserts the PLB_MnTimeout signal.

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

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

2.3.2 Mn_priority(0:1) (Request Priority)

The master drives these signals to indicate to the PLB arbiter the priority of the request of the master 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 before the slave that asserts the Sl_addrAck signal or the Sl_rearbitrate signal, or the master aborts the request through the Mn_abort signal.

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

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

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

2.3.3 Mn_busLock, PLB_busLock (Bus Arbitration Lock)

The current master can use the busLock signal to lock bus arbitration and force the PLB arbiter to continue to grant the bus to that master and ignore all other requests that are pending. The master asserts this signal with the assertion of the Mn_request signal as a transfer qualifier. It must remain asserted until the PLB arbiter samples it in the clock cycle in which the slave asserts the Sl_addrAck signal. At this point, the master has now locked both the read and write buses. If the master negates the Mn_busLock signal before the assertion of the Sl_addrAck signal, the bus is not locked. Also, if the master asserts the Mn_abort signal in

the same clock cycle in which the Sl_addrAck signal is asserted, the bus is not locked. The PLB can only be locked by requesting a transfer with the Mn_busLock signal asserted and being the highest priority request that is presented to the PLB arbiter.

When the current master has successfully locked the bus, it is not necessary for that master to continuously drive the request signal that is asserted. If the master negates the Mn_request signal, but does not negate the Mn_busLock signal, the bus continues to be locked to that master and remains locked until the master negates the Mn_busLock signal. More specifically, the bus continues 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 rearbitrates and grants the bus to the highest priority request.

Mn_busLock must remain deasserted in the absence of a locked request or transfer.

Note: A master request with the Mn_busLock signal asserted is a special case. This is because the PLB arbiter waits for both the read data bus and the write data bus to be available before granting the PLB to a master and presenting the address and the transfer qualifiers of the master to the slaves. See *Section 2.3.4 PLB_PAValid (PLB Primary Address Valid)* on page 29 and *Section 2.3.5 PLB_SAValid (Secondary Address Valid)* on page 30 for more detailed information about how the PLB arbiter handles a master request with the Mn_busLock signal asserted.

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

2.3.4 PLB_PAValid (PLB Primary Address Valid)

The PLB arbiter asserts this signal in response to the assertion of the Mn_request signal and to indicate that a valid primary address and transfer qualifiers are on the PLB outputs. The cycle in which the PLB_PAValid signal is asserted, relative to the assertion of the Mn_request signal, 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. All slaves must sample the PLB_PAValid signal. If the PLB_PAValid signal is asserted, the address is within the address range of the slaves, and the slaves are capable of performing the transfer, the slaves must 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, it must assert the Sl_wait signal to require the PLB arbiter to wait for the request to be correctly terminated. Otherwise, it must assert the Sl_rearbitrate signal to require the PLB arbiter to rearbitrate the bus.

When the PLB_PAValid signal is asserted, it remains asserted until any of the following conditions occur:

- A slave asserts the Sl_addrAck signal.
- The requesting master aborts the request.
- A slave asserts the Sl_rearbitrate signal.
- The PLB arbiter times out.

In the clock cycle following the occurrence of one of these conditions, the PLB_PAValid signal is deasserted in the absence of a master request. In the clock cycle following the occurrence of one of these conditions, in the presence of a master request, the PLB arbiter rearbitrates the bus and the PAValid signal can remain asserted with the address and transfers qualifiers for a subsequent transfer. This occurs only in single cycle acknowledgment implementations. See *Section 4 PLB Timing Guidelines* on page 67 for more detailed information.

Notes:

1. When the PLB_PAValid signal is asserted, the PLB arbiter waits for sixteen clock cycles for the request to be correctly terminated. If no slave responds by the 16th clock cycle with an Sl_wait signal, an Sl_AddrAck signal, or an Sl_rearbitrate signal, or the master does not abort the request, the PLB arbiter times out and asserts the correct PLB_MnTimeout signal to the master in the 17th clock cycle. See *Section 5.1.21 Bus Timeout Transfer* on page 106 for more detailed information.
2. When a slave has asserted the Sl_addrAck signal, the PLB arbiter waits 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 because of an address acknowledgment occurring without the corresponding data acknowledgments.

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

2.3.5 PLB_SAValid (Secondary Address Valid)

The PLB arbiter asserts this signal to indicate to a PLB slave that there is a valid secondary, or pipelined, address and transfer qualifiers on the PLB outputs. The clock cycle in which the PLB_SAValid signal is asserted, relative to the assertion of the Mn_request signal, 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 request is considered a pipelined request because the requested data bus is busy. For the read data bus, the busy state corresponds to the window of time starting with the clock cycle that follows the assertion of the Sl_addrAck signal and ending with the clock cycle in which the Sl_rdComp signal is asserted. For the write data bus, the busy state corresponds to the window of time starting with the clock cycle that follows the assertion of the Sl_addrAck signal and ending with the clock cycle in which the Sl_wrComp signal is asserted.

When the PLB_SAValid signal has been asserted for a pipelined read request, the PLB arbiter waits indefinitely for any of the following conditions to occur:

- A slave asserts the Sl_addrAck signal.
- A slave asserts the Sl_rearbitrate signal.
- The requesting master aborts the request.
- The Sl_rdComp signal is asserted for the primary read request.

In the first, second, and third case, the PLB arbiter rearbitrates the bus in the following clock cycle. In the fourth case, the PLB arbiter does not rearbitrate the bus. Instead, the PLB arbiter negates the PLB_SAValid signal and asserts the PLB_PAValid signal. This negation and assertion indicates that there is a new valid primary address and transfer qualifiers on the PLB outputs. The PLB_SAValid signal is deasserted and the PLB_PAValid signal is asserted in the same clock cycle the Sl_rdComp signal is asserted or in a subsequent clock cycle.

When the PLB_SAValid signal has been asserted for a secondary write request, the PLB arbiter waits indefinitely for any of the following conditions to occur:

- A slave asserts the Sl_addrAck signal.
- A slave asserts the Sl_rearbitrate signal.
- The requesting master aborts the request.
- The Sl_wrComp signal is asserted for the primary write request.

In the first, second, and third cases, the PLB arbiter rearbitrates the bus in the following clock cycle. In the fourth case, the PLB arbiter does not rearbitrate the bus. Instead, the PLB arbiter negates the PLB_SAValid signal and asserts the PLB_PAValid signal in one or more clock cycles that follow the assertion of the Sl_wrComp signal. This negation and assertion indicates that there is a new valid primary address and transfer qualifiers on the PLB outputs.

Notes:

1. It is not possible for a pipelined request to timeout 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, the slave must assert the Sl_rearbitrate signal. This assertion allows the arbiter to potentially move on to a transfer that can be acknowledged. It is up to the slave to determine under what conditions it must assert the Sl_rearbitrate signal or wait to acknowledge the pipelined request.
2. When a master has a valid bus lock condition established, the PLB_SAValid signal is only asserted for pipelined requests that the locking master generates. All other requests from other masters are ignored.
3. The PLB_SAValid signal is asserted once per pipelined request. It can be asserted more than once before completion of the primary data transfer for a given data bus. Each subsequent assertion of this signal for a particular data bus is a new transfer or level of pipelining. There is no limit to the depth of read or write pipelining possible.

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

2.3.6 Sl_wait (Wait for Address Acknowledgment)

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 can assert this signal anytime it recognizes a valid address and type on the PLB. The slave is not required to negate it before asserting the Sl_addrAck signal or the Sl_rearbitrate signal.

Note: The PLB arbiter qualifies the Sl_wait signal with the PLB_PAValid signal. Therefore, the slaves are not required to qualify the assertion of Sl_wait with the PLB_PAValid signal.

When the SI_wait signal is asserted in response to the assertion of the PLB_PAValid signal, the PLB arbiter uses the responding slave device's Sl_wait signal to disable the timeout mechanism for its address cycle bus and waits indefinitely for the slave to assert its Sl_addrAck or Sl_rearbitrate signals. Otherwise, the PLB arbiter waits a maximum of sixteen clock cycles for the Sl_addrAck signal or the Sl_rearbitrate signal to be asserted before timing out.

As long as the Sl_wait signal is asserted, the PLB timeout counter is inhibited from counting. The slave must assert this signal and it must remain asserted until the assertion of the Sln_addrAck signal, the PLB_abort signal, or the Sln_rearbitrate signal.

When the Sl_wait signal is asserted in response to the assertion of the PLB_SAValid signal, the PLB arbiter ignores the SI_wait signal and waits indefinitely for any of the following conditions to occur:

- The slave assert its Sl_addrAck signal.
- The master aborts the request.
- The slave asserts the Sl_rearbitrate signal.
- The secondary request becomes a primary request.

The Sl_wait signal is an input to the PLB arbiter only; it is not driven to the PLB masters.

2.3.7 Sl_addrAck, PLB_MnAddrAck (Address Acknowledgment)

This signal is asserted to indicate that the slave has acknowledged the address and latches the address and all of the transfer qualifiers at the end of the current clock cycle. The slave asserts this signal only while the PLB_PAValid signal or the PLB_SAValid signal is asserted. This signal must remain negated at all other times.

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. When this signal is asserted, the PLB arbiter must rearbitrate the bus. The slave asserts this signal only while the PLB_PAValid signal or the PLB_SAValid signal are asserted. This signal must remain negated at all other times.

When it is asserted in response to the assertion of the PLB_PAValid signal or the PLB_SAValid signal, the PLB arbiter passes this signal to the masters. The PLB arbiter then rearbitrates and grants the bus to the highest priority request. Furthermore, to avoid a possible deadlock scenario, the PLB arbiter ignores the original master request during rearbitration.

Even though a secondary or pipelined operation cannot immediately gain access to the data bus, slaves must, when the opportunity exists, assert the Sl_rearbitrate signal to allow subsequent pipelined transfers to propagate to the various slaves that are connected to the bus. This assertion increases overall throughput and decreases the latency of the bus.

Masters that do not lock the bus need not monitor the PLB_MnRearbitrate signal because the arbiter is required to ignore the original master request during rearbitration.

Notes:

1. Slaves must never assert the Sl_addrAck signal *and* Sl_rearbitrate in the same clock cycle. Slaves must either acknowledge a transfer or indicate their inability to respond by forcing rearbitration.
2. If the bus was previously locked, the Sl_rearbitrate signal is ignored by the PLB arbiter to prevent the interruption of an atomic operation. Hence, to prevent deadlock, the locking master must negate its Mn_request and Mn_busLock signals for a minimum of two clock cycles following the sampling of a PLB_MnRearbitrate assertion. See *Section 5.1.20 Slave Requested Rearbitration With Bus Locked* on page 105 for detailed information.

2.3.9 Mn_abort, PLB_abort (Abort Request)

The master asserts this signal 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 can only be used to abort a request that has not been acknowledged or is being acknowledged in the current clock cycle. In the clock cycle following the assertion of the Mn_abort signal, the master must either negate the Mn_request signal or make a new request. However, starting with the clock cycle following the assertion of the Sl_addrAck signal, the master can no longer abort a request and the slave is required to perform the necessary handshaking to complete the transfer. This signal has a minimum amount of set-up time to allow for its assertion late in the clock cycle.

Notes:

1. A slave can assert the Sl_wrDAck signal and the Sl_wrComp signal with the Sl_addrAck signal for a primary write request, even if the master is aborting the request clock cycle. In this case, the master is required to ignore these signals and the slave stores no data.

2. If the Sl_rearbitrate signal is asserted in the same clock cycle as the Mn_abort signal, the PLB arbiter ignores the Sl_rearbitrate signal and the master is not be backed-off during rearbitration. In the clock cycle following the assertion of the Mn_abort signal, the PLB arbiter rearbitrates and grants the highest priority request.

The slaves sample the PLB_abort signal only while the PLB_PAValid signal or the PLB_SAValid signal is asserted.

2.4 PLB Status Signals

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

2.4.1 PLB_rdPendReq (PLB Read Pending Bus Request)

The PLB arbiter asserts this signal to indicate that a master has a read request that is pending on the PLB or that a secondary read transfer has been acknowledged and is pending. This signal is a combined logic OR of all the master request inputs for reads and the secondary read bus status. This signal is combinatorially asserted with the request, or is asserted the clock cycle after the assertion of the Sl_addrAck signal for a secondary transfer. This signal can be sampled by any PLB master, or it slave or can be used by itself. Also, when another master requests the bus, this signal can be used with the PLB_rdPendPri(0:1) signal to determine when to negate the Mn_busLock or Mn_rdBurst signals. This signal is negated combinatorially with the negation of the request or, in the case of a secondary transfer, in the clock cycle following the assertion of an rdPrim signal.

This signal is always valid and is not negated during a clock cycle in which the master is aborting a request.

2.4.2 PLB_wrPendReq (PLB Write Pending Bus Request)

The PLB arbiter asserts this signal to indicate that a master has a write request that is pending on the PLB or to indicate that a secondary write transfer has been acknowledged and is pending. This signal is a combined logic OR of all the master request inputs for writes and for the secondary read bus status. This signal is combinatorially asserted with the request, or is asserted in the clock cycle after the assertion of the Sl_addrAck signal for a secondary transfer.

This signal is always valid and is not negated during a clock cycle in which the master is aborting a request.

2.4.3 PLB_rdPendPri(0:1) (PLB Read Pending Request Priority)

These signals are driven by the PLB arbiter and are valid any time the PLB_rdPendReq signal is asserted. These signals indicate the highest priority of any active read request input from all masters that are attached to the PLB or a pipelined read transfer that has been acknowledged and is pending. Masters can use these signals to determine when to negate the Mn_busLock or Mn_rdBurst signals because of another master requesting with higher priority. Slaves can also use these signals.

2.4.4 PLB_wrPendPri(0:1) (PLB Write Pending Request Priority)

These signals are driven by the PLB arbiter and are valid any time the PLB_wrPendReq signal is asserted. These signals indicate the highest priority of any active write request input from all masters attached to the PLB or a pipelined write transfer that has been acknowledged and is pending. Masters can use these signals to determine when to negate the Mn_busLock or Mn_wrBurst signals because of another master requesting with higher priority. Slaves can also use these signals.

2.4.5 PLB_reqPri(0:1) (PLB Current Request Priority)

These signals are driven by the PLB arbiter and are valid any time the PLB_rdPendReq or PLB_wrPendReq signals are asserted. These signals indicate the priority of the current request that the PLB arbiter has granted and is gating to the slaves. This priority remains valid from the clock cycle in which the PLB_PAValid signal or the PLB_SAValid signal is asserted until the clock cycle in which the slave has acknowledged the request. Slaves can also use these signals to resolve arbitration when requesting access to other buses.

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

These signals are driven by the PLB arbiter and are valid in any clock cycle in which the PLB_PAValid signal or the PLB_SAValid signal is asserted. These signals indicate to the slaves the identification of the master of the current transfer. The slave must use these signals to determine to which master the Sl_MBusy, Sl_MRdErr, and Sl_MWrErr signals must be driven on the PLB. The slave can also latch the master ID in an error syndrome register to indicate which master request caused the error.

Note: The width of the PLB_masterID signal (as shown in *Table 2-3*) is determined by the maximum number of masters the particular PLB arbiter implementation supports.

Table 2-3. PLB Master Identification

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

2.4.7 PLB_MnTimeout (PLB Master Bus Timeout)

These signals are driven by the PLB arbiter to each master and are asserted in the 17th clock cycle after the assertion of the PLB_PAValid signal with no response from the slave. The PLB arbiter begins counting in the first clock cycle in which the PLB_PAValid signal is asserted and after one of the following events:

- Sixteen clock cycles occur without the assertion of the Sl_wait, the Sl_addrAck, or the Sl_rearbitrate signals.
- The master asserts the Mn_abort signal in the 17th clock cycle.
- The arbiter asserts PLB_MnTimeout signal in the 17th clock cycle.

The assertion of Sl_wait, Sl_addrAck, Sl_rearbitrate, and the Mn_abort signals is ignored after the 16th clock cycle. The PLB_MnTimeout signal is asserted for one clock cycle only and indicates to the master that its transfer request has timed out on the bus. The master must immediately deassert its Mn_request signal for the current transfer that has timed out. No additional handshaking for the timed-out transfer occurs. The master can immediately request another transfer to a different address; however, repeated attempts to the timed-out address must be avoided.

The PLB asserts these signals only while the PLB_PAValid signal is asserted. These signals remain negated at all other times.

2.5 PLB Transfer Qualifier Signals

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

2.5.1 Mn_RNW, PLB_RNW (Read/NotWrite)

The master drives this signal to indicate whether the request is for a read or a write transfer. If Mn_RNW = ‘1’, the request is for the slave to supply data to be read into the master. If Mn_RNW ‘0’, the request is for the master to supply data to be written to the slave.

2.5.2 Mn_BE, PLB_BE (Byte Enables)

The master drives these signals. For a nonline transfer and a nonburst transfer, they identify which bytes of the target that is being addressed are to be read from or written to. Each bit corresponds to a byte lane on the read or write data bus.

Table 2-4. Byte Enables for Various Bus Widths

PLB Data Bus Width	Number of Byte Enables	PLB_BE Size
32-Bit	4	0:3
64-Bit	8	0:7
128-Bit	16	0:15

For a read transfer, the slaves must access the indicated bytes and place them on the Sl_rdDbus in the correct memory alignment. These bytes are then steered to the PLB_MnRdDBus. For a write transfer, the slaves must only write out the indicated bytes from the Mn_wrDBus 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 *Table 2-5*.

Table 2-5. Byte Enable Signals Transfer Request (Sheet 1 of 4)

32-Bit Bus PLB_BE(0:3)	64-Bit Bus PLB_BE(0:7)	128-Bit Bus PLB_BE(0:15)	Transfer Request	Mn_ABus (28:31)
1000	1000_0000	1000_0000_0000_0000	Byte 0	0000
0100	0100_0000	0100_0000_0000_0000	Byte 1	0001
0010	0010_0000	0010_0000_0000_0000	Byte 2	0010
0001	0001_0000	0001_0000_0000_0000	Byte 3	0011
1000	0000_1000	0000_1000_0000_0000	Byte 4	0100
0100	0000_0100	0000_0100_0000_0000	Byte 5	0101
0010	0000_0010	0000_0010_0000_0000	Byte 6	0110
0001	0000_0001	0000_0001_0000_0000	Byte 7	0111
1000	1000_0000	0000_0000_1000_0000	Byte 8	1000
0100	0100_0000	0000_0000_0100_0000	Byte 9	1001
0010	0010_0000	0000_0000_0010_0000	Byte 10	1010
0001	0001_0000	0000_0000_0001_0000	Byte 11	1011
1000	0000_1000	0000_0000_0000_1000	Byte 12	1100
0100	0000_0100	0000_0000_0000_0100	Byte 13	1101
0010	0000_0010	0000_0000_0000_0010	Byte 14	1110
0001	0000_0001	0000_0000_0000_0001	Byte 15	1111
1100	1100_0000	1100_0000_0000_0000	Halfword 0, 1	0000
0110	0110_0000	0110_0000_0000_0000	Unaligned halfword 1, 2	0001
0011	0011_0000	0011_0000_0000_0000	Halfword 2, 3	0010
N/A	0001_1000	0001_1000_0000_0000	Unaligned halfword 3, 4	0011
1100	0000_1100	0000_1100_0000_0000	Halfword 4, 5	0100
0110	0000_0110	0000_0110_0000_0000	Unaligned halfword 5, 6	0101
0011	0000_0011	0000_0011_0000_0000	Halfword 6, 7	0110
N/A	N/A	0000_0001_1000_0000	Unaligned halfword 7, 8	0111
1100	1100_0000	0000_0000_1100_0000	Halfword 8, 9	1000
0110	0110_0000	0000_0000_0110_0000	Unaligned halfword 9, 10	1001
0011	0011_0000	0000_0000_0011_0000	Halfword 10, 11	1010
N/A	0001_1000	0000_0000_0001_1000	Unaligned halfword 11, 12	1011
1100	0000_1100	0000_0000_0000_1100	Halfword 12, 13	1100
0110	0000_0110	0000_0000_0000_0110	Unaligned halfword 13, 14	1101
0011	0000_0011	0000_0000_0000_0011	Halfword 14, 15	1110

Table 2-5. Byte Enable Signals Transfer Request (Sheet 2 of 4)

32-Bit Bus PLB_BE(0:3)	64-Bit Bus PLB_BE(0:7)	128-Bit Bus PLB_BE(0:15)	Transfer Request	Mn_ABus (28:31)
1110	1110_0000	1110_0000_0000_0000	Bytes 0,1, 2	0000
0111	0111_0000	0111_0000_0000_0000	Bytes 1, 2, 3	0001
N/A	0011_1000	0011_1000_0000_0000	Bytes 2, 3, 4	0010
N/A	0001_1100	0001_1100_0000_0000	Bytes 3, 4, 5	0011
1110	0000_1110	0000_1110_0000_0000	Bytes 4, 5, 6	0100
0111	0000_0111	0000_0111_0000_0000	Bytes 5, 6, 7	0101
N/A	N/A	0000_0011_1000_0000	Bytes 6, 7, 8	0110
N/A	N/A	0000_0001_1100_0000	Bytes 7, 8, 9	0111
1110	1110_0000	0000_0000_1110_0000	Bytes 8, 9, 10	1000
0111	0111_0000	0000_0000_0111_0000	Bytes 9, 10, 11	1001
N/A	0011_1000	0000_0000_0011_1000	Bytes 10, 11, 12	1010
N/A	0001_1100	0000_0000_0001_1100	Bytes 11, 12, 13	1011
1110	0000_1110	0000_0000_0000_1110	Bytes 12, 13, 14	1100
0111	0000_0111	0000_0000_0000_0111	Bytes 13, 14, 15	1101
1111	1111_0000	1111_0000_0000_0000	Word 0, 1, 2, 3	0000
N/A	0111_1000	0111_1000_0000_0000	Unaligned Word 1, 2, 3, 4	0001
N/A	0011_1100	0011_1100_0000_0000	Unaligned Word 2, 3, 4, 5	0010
N/A	0001_1110	0001_1110_0000_0000	Unaligned Word 3, 4, 5, 6	0011
1111	0000_1111	0000_1111_0000_0000	Word 4, 5, 6, 7	0100
N/A	N/A	0000_0111_1000_0000	Unaligned Word 5, 6, 7, 8	0101
N/A	N/A	0000_0011_1100_0000	Unaligned Word 6, 7, 8, 9	0110
N/A	N/A	0000_0001_1110_0000	Unaligned Word 7, 8, 9, 10	0111
1111	1111_0000	0000_0000_1111_0000	Word 8-11	1000
N/A	0111_1000	0000_0000_0111_1000	Unaligned Word 9-12	1001
N/A	0011_1100	0000_0000_0011_1100	Unaligned Word 10-13	1010
N/A	0001_1110	0000_0000_0001_1110	Unaligned Word 11-14	1011
1111	0000_1111	0000_0000_0000_1111	Word 12, 13, 14, 15	1100
N/A	1111_1000	1111_1000_0000_0000	Bytes 0, 1, 2, 3, 4	0000
N/A	0111_1100	0111_1100_0000_0000	Bytes 1, 2, 3, 4, 5	0001
N/A	0011_1110	0011_1110_0000_0000	Bytes 2, 3, 4, 5, 6	0010
N/A	0001_1111	0001_1111_0000_0000	Bytes 3, 4, 5, 6, 7	0011
N/A	N/A	0000_1111_1000_0000	Bytes 4, 5, 6, 7, 8	0100
N/A	N/A	0000_0111_1100_0000	Bytes 5, 6, 7, 8, 9	0101
N/A	N/A	0000_0011_1110_0000	Bytes 6, 7, 8, 9, 10	0110
N/A	N/A	0000_0001_1111_0000	Bytes 7, 8, 9, 10, 11	0111
N/A	1111_1000	0000_0000_1111_1000	Bytes 8, 9, 10, 11, 12	1000

Table 2-5. Byte Enable Signals Transfer Request (Sheet 3 of 4)

32-Bit Bus PLB_BE(0:3)	64-Bit Bus PLB_BE(0:7)	128-Bit Bus PLB_BE(0:15)	Transfer Request	Mn_ABus (28:31)
N/A	0111_1100	0000_0000_0111_1100	Bytes 9, 10, 11, 12, 13	1001
N/A	0011_1110	0000_0000_0011_1110	Bytes 10, 11, 12, 13, 14	1010
N/A	0001_1111	0000_0000_0001_1111	Bytes 11, 12, 13, 14, 15	1010
N/A	1111_1100	1111_1100_0000_0000	Bytes 0, 1, 2, 3, 4, 5	0000
N/A	0111_1110	0111_1110_0000_0000	Bytes 1, 2, 3, 4, 5, 6	0001
N/A	0011_1111	0011_1111_0000_0000	Bytes 2, 3, 4, 5, 6, 7	0010
N/A	N/A	0001_1111_1000_0000	Bytes 3, 4, 5, 6, 7, 8	0011
N/A	N/A	0000_1111_1100_0000	Bytes 4, 5, 6, 7, 8, 9	0100
N/A	N/A	0000_0111_1110_0000	Bytes 5, 6, 7, 8, 9, 10	0101
N/A	N/A	0000_0011_1111_0000	Bytes 6, 7, 8, 9, 10, 11	0110
N/A	N/A	0000_0001_1111_1000	Bytes 7, 8, 9, 10, 11, 12	0111
N/A	1111_1100	0000_0000_1111_1100	Bytes 8, 9, 10, 11, 12, 13	1000
N/A	0111_1110	0000_0000_0111_1110	Bytes 9, 10, 11, 12, 13, 14	1001
N/A	0011_1111	0000_0000_0011_1111	Bytes 10, 11, 12, 13, 14, 15	1010
N/A	1111_1110	1111_1110_0000_0000	Bytes 0, 1, 2, 3, 4, 5, 6	0000
N/A	0111_1111	0111_1111_0000_0000	Bytes 1, 2, 3, 4, 5, 6, 7	0001
N/A	N/A	0011_1111_1000_0000	Bytes 2, 3, 4, 5, 6, 7, 8	0010
N/A	N/A	0001_1111_1100_0000	Bytes 3, 4, 5, 6, 7, 8, 9	0011
N/A	N/A	0000_1111_1110_0000	Bytes 4, 5, 6, 7, 8, 9, 10	0100
N/A	N/A	0000_0111_1111_0000	Bytes 5, 6, 7, 8, 9, 10, 11	0101
N/A	N/A	0000_0011_1111_1000	Bytes 6, 7, 8, 9, 10, 11, 12	0110
N/A	N/A	0000_0001_1111_1100	Bytes 7, 8, 9, 10, 11, 12, 13	0111
N/A	1111_1110	0000_0000_1111_1110	Bytes 8, 9, 10, 11, 12, 13, 14	1000
N/A	0111_1111	0000_0000_0111_1111	Bytes 9, 10, 11, 12, 13, 14, 15	1001
N/A	1111_1111	1111_1111_0000_0000	Doubleword	0000
N/A	N/A	0111_1111_1000_0000	Bytes 1, 2, 3, 4, 5, 6, 7, 8	0001
N/A	N/A	0011_1111_1100_0000	Bytes 2, 3, 4, 5, 6, 7, 8, 9	0010
N/A	N/A	0001_1111_1110_0000	Bytes 3, 4, 5, 6, 7, 8, 9, 10	0011
N/A	N/A	0000_1111_1111_0000	Bytes 4, 5, 6, 7, 8, 9, 10, 11	0100
N/A	N/A	0000_0111_1111_1000	Bytes 5, 6, 7, 8, 9, 10, 11, 12	0101
N/A	N/A	0000_0011_1111_1100	Bytes 6, 7, 8, 9, 10, 11, 12, 13	0110
N/A	N/A	0000_0001_1111_1110	Bytes 7 14	0111
N/A	1111_1111	0000_0000_1111_1111	Doubleword	1000
N/A	N/A	1111_1111_1000_0000	Bytes 0 8	0000
N/A	N/A	0111_1111_1100_0000	Bytes 1 9	0001
N/A	N/A	0011_1111_1110_0000	Bytes 2 10	0010

Table 2-5. Byte Enable Signals Transfer Request (Sheet 4 of 4)

32-Bit Bus PLB_BE(0:3)	64-Bit Bus PLB_BE(0:7)	128-Bit Bus PLB_BE(0:15)	Transfer Request	Mn_ABus (28:31)
N/A	N/A	0001_1111_1111_0000	Bytes 3 11	0011
N/A	N/A	0000_1111_1111_1000	Bytes 4 12	0100
N/A	N/A	0000_0111_1111_1100	Bytes 5 13	0101
N/A	N/A	0000_0011_1111_1110	Bytes 6 14	0110
N/A	N/A	0000_0001_1111_1111	Bytes 7 15	0111
N/A	N/A	1111_1111_1100_0000	Bytes 0 9	0000
N/A	N/A	0111_1111_1110_0000	Bytes 1 10	0001
N/A	N/A	0011_1111_1111_0000	Bytes 2 11	0010
N/A	N/A	0001_1111_1111_1000	Bytes 3 12	0011
N/A	N/A	0000_1111_1111_1100	Bytes 4 13	0100
N/A	N/A	0000_0111_1111_1110	Bytes 5 14	0101
N/A	N/A	0000_0011_1111_1111	Bytes 6 15	0110
N/A	N/A	1111_1111_1110_0000	Bytes 0 10	0000
N/A	N/A	0111_1111_1111_0000	Bytes 1 11	0001
N/A	N/A	0011_1111_1111_1000	Bytes 2 12	0010
N/A	N/A	0001_1111_1111_1100	Bytes 3 13	0011
N/A	N/A	0000_1111_1111_1110	Bytes 4 14	0100
N/A	N/A	0000_0111_1111_1111	Bytes 5 15	0101
N/A	N/A	1111_1111_1111_0000	Bytes 0 11	0000
N/A	N/A	0111_1111_1111_1000	Bytes 1 12	0001
N/A	N/A	0011_1111_1111_1100	Bytes 2 13	0010
N/A	N/A	0001_1111_1111_1110	Bytes 3 14	0011
N/A	N/A	0000_1111_1111_1111	Bytes 4 15	0100
N/A	N/A	1111_1111_1111_1000	Bytes 0 12	0000
N/A	N/A	0111_1111_1111_1100	Bytes 1 13	0001
N/A	N/A	0011_1111_1111_1110	Bytes 2 14	0010
N/A	N/A	0001_1111_1111_1111	Bytes 3 15	0011
N/A	N/A	1111_1111_1111_1100	Bytes 0 13	0000
N/A	N/A	0111_1111_1111_1110	Bytes 1 14	0001
N/A	N/A	0011_1111_1111_1111	Bytes 2 15	0010
N/A	N/A	1111_1111_1111_1110	Bytes 0 14	0000
N/A	N/A	0111_1111_1111_1111	Bytes 1 15	0001
N/A	N/A	1111_1111_1111_1111	Quadword	0000

For masters whose data bus width is less than that of the PLB bus, mirroring of the byte enables (BE) signal occurs. Because of this mirroring, noncontiguous BE signals might occur. Slave implementations must account for this occurrence. For example, when 32-bit masters are attached to a 64-bit PLB, the PLB_BE(4:7)

signals are a replica of the PLB_BE(0:3) signals. This replication might result in a noncontiguous BE signal combination from the viewpoint of a 64-bit slave. For example, if the master drives PLB_BE(0:3) with ‘0001’, PLB_BE(0:7) is ‘0001 0001’. Slave designs must account for the fact that this replication can occur and base all decodes of BE signals from a combination of the PLB_BEs and the PLB_Abus.

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

Note: For burst transfers, the Mn_BE signals can optionally indicate the number of transfers that the master is requesting. *Table 2-6* shows the definition of the Mn_BE(0:3) signals for 32-bit PLB implementations during burst transfers.

Table 2-6. Byte Enable Signals during Burst Transfers (32-bit PLB)

Mn_BE(0:3)	Burst Length
0000	The PLB_rdBurst signal or the PLBwrBurst signal determines burst length.
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

The burst length refers to the number of transfers of the data type that are selected by the Mn_size signals. The Mn_size ‘1000’ and Mn_BE(0:3) ‘1111’ transfers 16 bytes, Mn_size ‘1001’ and Mn_BE(0:3) ‘1111’ transfer 16 halfwords, and Mn_BE(0:3 ‘1111’ and Mn_size ‘1010’ transfer 16 words.

Table 2-7. Byte Enable Signals during Burst Transfers for (64-Bit and above PLB) (Sheet 1 of 2)

Mn_BE(4:7)_Mn_BE(0:3)	Burst Length
0000_0000	The PLB_rdBurst signal or the PLBwrBurst signal determines burst length.
0000_0001	Burst of 2
0000_0010	Burst of 3
.	.
.	.
0000_1111	Burst of 16

Table 2-7. Byte Enable Signals during Burst Transfers for (64-Bit and above PLB) (Sheet 2 of 2)

Mn_BE(4:7)_Mn_BE(0:3)	Burst Length
0001_0000	Burst of 17
0001_0001	Burst of 18
0001_0010	Burst of 19
.	.
.	.
.	.
1111_1110	Burst of 255
1111_1111	Burst of 256

Burst length refers to the number of transfers of the data type that the Mn_size signals select. Mn_size ‘1000’ and Mn_BE(0:7) ‘1111 0001’ transfer 32 bytes. Mn_size ‘1001’ and Mn_BE(0:7) ‘1111 0001’ transfer 32 halfwords and Mn_BE(0:7) ‘1111 0001’. Mn_size ‘1010’ transfers 32 words.

Masters which do not implement the fixed-length transfer must drive all 0’s on the BE signals during burst transfers to be compatible with slaves that have implemented this feature. Slaves that do not implement the fixed-length transfer ignore the PLB_BE signals during a burst transfer and continue bursting until the master negates the PLB_rdBurst signal or the PLB_wrBurst signal. Slave that are 32 bits and are connected to implementations that are 64 bits or higher do not sample the PLB_BE(4:7) signal. Also, these slaves do not know the full transfer length that masters that are 64 bits or higher request. These slaves terminate the requested transfer prematurely when PLB_BE(4:7) is not ‘0000’. Also, fixed-length burst transfers to these 32-bit slaves with the PLB_BE(0:3) ‘0000’ and PLB_BE(4:7) not ‘0000’ cause the slave to continue bursting until the PLB_rdBurst or PLB_wrBurst signal is negated by the master. See *Section 5.1.15 Fixed-Length Burst Read Transfer* on page 100 for a detailed description.

2.5.3 Mn_BEPar, PLB_BEPar (Byte Enables Parity)

The master drives this signal if the master supports parity for the Mn_BE signal. The master generates odd parity, where the number of 1’s across the Mn_BE bus and the parity bit is an odd number. The PLB_BEPar signal arrives at the slave at the same time as the PLB_BE signal. The slave must use the signal to check the parity of the BEs to verify that the byte enables are correct.

2.5.4 Mn_BEParEn, PLB_BEParEn (Byte Enables Parity Enable)

The master drives this signal if the master supports parity for the Mn_BE signal. It indicates that the Mn_BEPar signal is valid, and must be used by the slave to check parity on the PLB_BE signals. The PLB_BEParEn signal arrives at the slave at the same time as the PLB_BE signals. The system integrator must hard wire this signal to a ‘0’ if the master does not support parity on the Mn_BE signal.

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

The Mn_size(0:3) signals are driven by the master to indicate the size of the requested transfer. *Table 2-8 PLB Transfer Size Signals* on page 42 defines all PLB transfer size signals.

Table 2-8. PLB Transfer Size Signals

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

Note:

1. A ‘0000’ value indicates that the request is to transfer 1 4 bytes on a 32-bit PLB, one to eight bytes on a 64-bit PLB, or 1 16 bytes on a 128-bit PLB starting at the target address. The number of bytes to be transferred are indicated on the Mn_BE signals.
2. For line read transfers, the target word might or might not be the first word transferred, depending on the design of the slave. For line read transfers, the Sl_rdWdAddr(0:3) signals 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: Mn_ABus(28:31) ‘00XX’ for a 4-word line write, Mn_ABus(27:31) ‘000XX’ for an 8-word line write, and Mn_ABus(26:31) ‘0000 XX’ for a 16-word line write.
3. The Mn_BE signals are ignored during the data tenure for a burst transfer.
4. If the Mn_size(0:3) signal is ‘1000’, the request is to burst read or write bytes. If Mn_size(0:3) is ‘1001’, the request is to burst. If Mn_size(0:3) is ‘1010’, the request is to burst words. The slave must start transferring data at the address indicated by the PLB_ABus(0:31) and width as indicated by the size bits. The slave must then continue to read or write bytes, halfwords, or words, until the Mn_burst signal is negated indicating that the master no longer needs additional data.
5. For the PLB and devices supporting wider data paths, doubleword encodings are used to transfer 64 bits. For PLB and devices supporting wider data paths, quadword encodings are used to transfer 128-bits. Slaves must be designed to support all these transfer sizes, even though their data bus width might be less than the requested transfer size. Slaves with data bus widths that are less than the requested burst transfer, must provide for or accept their full data bus width. For example, when a 64-bit master requests a doubleword (64-bit) burst read transfer from a word (32-bit) slave, the slave must decode the type and must provide word (32-bit) data. The master is responsible for negating the Mn_burst signal at the appropriate time to fulfill its transfer request.

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

The Mn_type signals are driven by the master and are used to indicate to the slave, through the PLB_type signals, the type of transfer that is being requested. *Table 2-9 PLB Transfer Type Signals* on page 43 defines all of the PLB transfer type signals. Two categories of transfer types exist: memory transfer and direct memory access (DMA) transfer. Only DMA masters attempt transfers with the Mn_type signal not equal to ‘000’. Memory slaves that do not accept special DMA peripheral transfers must implement type ‘000’ Memory Transfer and type ‘110’ DMA buffered memory transfer. The operation of the memory slave for transfer type ‘110’ can be the same as for type ‘000’. If it can be guaranteed that a memory slave is never required to respond to a type ‘110’ transfer, it does not need to be supported at all.

Table 2-9. PLB Transfer Type Signals

Mn_type(0:2)	Definition
000	Memory transfer.
001	Optional. DMA flyby transfer[1].
010	Optional. DMA buffered external peripheral transfer[2].
011	Optional. DMA buffered onchip peripheral bus peripheral transfer[2].
100	Optional. PLB slave buffered memory to memory transfer[2].
101	Optional. PLB slave buffered peripheral to and from memory transfer[2].
110	Optional. DMA buffered memory transfer[3].
111	Optional. PLB slave buffered memory to memory transfer with sideband signals.

1. Must be used with Mn_size(0:3) values of ‘0000’ and ‘1000’ ‘1101’ only.
2. Must be used with Mn_size(0:3) values of ‘0000’ only.
3. Memory slaves not supporting DMA peripheral transfers must also decode Mn_type(0:2) ‘110’ as a memory transfer to support DMA buffered memory-to-memory transfers.

2.5.6.1 Memory Transfers (Mn_type = ‘000’)

This transfer type is used to read data from or write data to a device in the memory address space. Each PLB slave must decode the address on the PLB address bus to determine if the transfer is to or from the memory area that the slave controls.

2.5.7 Mn_MSize(0:1), PLB_MSize(0:1) (Master Size))

These signals are inputs to the PLB core for each master port. These signals indicate the data bus width of the associated master. These signals do not need to be outputs of the master because the inputs to the PLB might be tied. The MSize signals are valid with the master’s active request for the duration of the PLB_PAValid signal or the PLB_SAValid signal. The PLB_MSize(0:1) signals are inputs to the 64-bit PLB slaves and specify the data bus width of the valid master request that is currently being broadcast to the PLB slaves. It is possible to change the master size with each request.

Table 2-10. Mn_MSize(0:1) Master Size

Mn_MSize(0:1)	Master Data Bus Size
00	32-Bit
01	64-Bit
10	128-Bit
11	128-Bit DDR

2.5.8 Sl_SSize(0:1), PLB_MnSSize(0:1) (Slave Size))

The Sl_SSize(0:1) signals are outputs of all non 32-bit PLB slaves. The slaves activate these signals with the assertion of the PLB_PAValid signal or the SAValid signal, and a valid slave address decode and must remain negated at all other times. These signals routed through the PLB to the requesting master as PLB_MnSSize(0:1). The master only samples these signals in the clock cycle in which the PLB_MnaddrAck signal is asserted.

Table 2-11. Sl_SSize(0:1) Slave Size

Sl_SSize(0:1)	Slave Data Bus Size
00	32-Bit
01	64-Bit
10	128-Bit
11	128-Bit DDR

2.5.9 Mn_TAttribute(0:15), PLB_TAttribute(0:15) (Transfer Attributes)

The master drives these signals to present specific transfer information to slaves which monitor these signals. Masters which implement this bus are required to assert the Mn_TAttributes signal with the Mn_request signal. Also, they must remain valid until they are latched by the slave with the assertion of the Sl_addrAck signal. The functionality of these generic signals is independent of the PLB architecture. However, recommended bit assignments follow.

2.5.9.1 Mn_TAttribute(0), PLB_TAttribute(0) (W - Write Through Storage Attribute)

This attribute controls the write-through versus copy-back behavior of any caches in the system. If this signal is asserted, any write accesses must write-through to memory instead of only writing into the cache. If this signal is negated, write accesses must only write in the cache.

2.5.9.2 Mn_TAttribute(1), PLB_TAttribute(1) (I) - Caching Inhibited Storage Attribute)

This attribute controls whether a given access is cacheable or not. If this signal is asserted, a PLB slave which implements a cache must bypass the cache and perform the access, read, or write operations directly to memory. If this signal is negated, the cache can be accessed instead of memory.

2.5.9.3 Mn_TAttribute(2), PLB_TAttribute(2) (M - Memory Coherent Storage Attribute)

This attribute specifies whether the access must be performed in a fashion which maintains memory coherency with the rest of the system. The specific requirements for performing accesses when this signal is asserted are highly dependent on the bus architecture of the system.

2.5.9.4 Mn_TAttribute(3), PLB_TAttribute(3) (G - Guarded Storage Attribute)

This attribute indicates that the requested transfer can be for a non-well-behaved memory. If a master is requesting a nonburst transfer (Mn_size(0) ‘00’), and this signal is negated, the master is indicating that the 1 KB page of memory that corresponds to the requested address is well behaved and that the slave can access all of the 1 KB page, but might stop at the 1 KB page boundary. If the master is requesting a nonburst transfer with this signal asserted, the master is indicating that the 1 KB page of memory that corresponds to the requested address might not be well behaved. Therefore, the slave must restrict itself to accessing only exactly what was requested by the master.

If the master is requesting a burst transfer (Mn_size(0) ‘01’), and the signal is negated, the master is indicating that the 1 KB page that corresponds to the requested address is well behaved and all subsequent 1 KB pages of memory are also well behaved. Therefore, the slave can access all memory on this page and subsequent pages. If the master is requesting a burst transfer with this signal asserted, the master is indi-

cating that the 1 KB page of memory corresponding to the requested address is well behaved. However, the next 1 KB page of memory might not be well behaved. Therefore, the slave must restrict itself to accessing only within the 1 KB page that corresponds to the initial requested address.

When stopping a burst transfer at a 1 KB page boundary, the slave can 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 acknowledgment and negate the Mn_rdBurst or Mn_wrBurst signals in the following clock cycle to guarantee that the 1 KB page boundary is not crossed. Following the detection of the second-to-last data acknowledge, if a master decides that it is allowable to cross into the next page, it can indicate so by leaving the Mn_rdBurst or 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 it is not explicitly required by a master. If the *wait* technique is used, slaves must not cross the 1 KB page boundary until they have returned the second-to-last read/write data acknowledgment and have given the masters the opportunity to negate their Mn_rdBurst signals or their Mn_wrBurst signals in the following clock cycle. Following the detection of the second-to-last data acknowledgment, if the Mn_rdBurst signals or the Mn_wrBurst signals are still asserted, the slave can assume that the master has requested data from the next page. Therefore the page can be accessed.

2.5.9.5 Mn_TAttribute(4), PLB_TAttribute(4) (U0 - User Defined Storage Attribute)

The master drives this signal, formerly known as PLB_compress, to indicate whether the requested transfer is for a memory area that contains compressed data. If the master is requesting a read data transfer and this signal is asserted, the master is indicating that the data that corresponds to the requested address is compressed. Therefore, the slave must decompress the data before transferring it back to the master. If the master is requesting a write data transfer and this is asserted, the master is indicating that the data corresponding to the requested address must be compressed. Therefore, the slave must compress the data before writing it to memory.

2.5.9.6 Mn_TAttribute(5:7), PLB_TAttribute(5:7) (U1-U3 User Defined Storage Attributes)

These signals are meant to be used as processor-dependent storage attributes. In general, masters other than CPUs must not use these bits. Masters that want to convey specific transfer information to slaves must use bits 9 15 of the Mn_TAttribute bus.

2.5.9.7 Mn_TAttribute[8], PLB_TAttribute[8] (Ordered Transfer)

The master drives this signal, formerly known as PLB_ordered, for a write request to indicate whether 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 (logic 0) indicating a write transfer. PLB slaves must ignore the PLB_TAttribute[8] signal during read transfers.

When the slave acknowledges a write request with the Mn_TAttribute[8] 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_TAttribute[8] signal negated, the slave can decide to hold this request in a buffer and perform subsequent requests (reads or writes) before completing the unordered write request.

Although the Mn_TAttribute[8] signal can 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 the data.

This signal is used a master that needs to ensure that a write-transfer operation is complete before the data that is being written can be accessed by any other system resource.

2.5.9.8 Mn_TAttribute(9:15), PLB_TAttributes(9:15) (Transfer Attributes)

The master drives these signals to present specific transfer information to slaves that monitor these signals. Masters that implement these signals are required to assert the Mn_TAttributes signal with the Mn_request signal, and these signals must remain valid until they are latched by the slave with the assertion of the Sl_addrAck signal. The functionality of these generic signals is independent of the PLB architecture. The system designer can assign these bits in a particular fashion to convey information between master and slaves. No conventional assignment of these bits exists; they can be considered reserved for future use.

2.5.10 Mn_lockErr, PLB_lockErr (Lock Error Status)

The master asserts this signal to indicate whether 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 must not lock the SEAR and SESR when the error occurs. Instead, the error address and syndrome must 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 are overwritten. If the value of this signal is high (logic 1), the slave must lock the SEAR and SESR when errors occur as a result of this transfer. Errors that occur after the SEAR and SESR are locked, however, do not override the values that were written with the first error. When the SESR and SEAR registers are locked with an error, they remain locked until software clears the SESR.

2.5.11 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 then arbitrates the requests and allows the highest priority master's address to be gated onto the PLB_ABus signal. For nonline transfers, this 32-bit bus indicates the lowest numbered byte address of the target data to be read or written over the PLB. The Mn_BE(0:3) signals indicate which bytes of the word are read or written for this transfer. See *Section 2.5.2 Mn_BE, PLB_BE (Byte Enables)* on page 35 for a detailed description of the Mn_BE signals.

For line read transfers, the address bus can indicate the target byte address within the line of data that the master is requesting. Slaves can read the data in any order and can use the target address to optimize performance by transferring the target word first. For line write transfers, the line word address must be zero because line writes transfers must be performed in sequential order across the PLB, starting with the first word of the line. *Table 2-12* indicates the bits of the Mn_ABus that must be zeroed for line write transfers.The slave must latch the address at the end of the clock cycle in which it asserts the Sl_addrAck signal.

Table 2-12. 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)

2.5.12 Mn_ABusPar, PLB_ABusPar (Address Bus Parity)

The master drives this signal if the master supports parity for the Mn_ABus signal. One parity bit is used for the Mn_ABus(0:31) signal. The master generates odd parity, where the number of 1's across the Mn_ABus signal and the parity bit is an odd number. The PLB_ABusPar signal arrives at the slave at the same time as PLB_ABus signal. The signal must be used by the slave to check the parity of the PLB_ABus signal to verify that the address is correct.

2.5.13 Mn_ABusParEn, PLB_ABusParEn (Address Bus Parity Enable)

The master drives this signal if the master supports parity for the Mn_ABus signal. It indicates that the Mn_ABusPar signal is valid, and it must be used by the slave to check parity on the PLB_ABus signals. The PLB_ABusParEn signal arrives at the slave at the same time as the PLB_ABus signals. The system integrator must hard wire this signal to a '0' if the master does not support parity on the Mn_ABus signal.

2.5.14 Mn_UABus(0:31), PLB_UABus(0:31) (Upper Address Bus)

This upper address bus allows for address expansion above the 4 GB 32-bit address limit by expanding the address bus from 32-bits to 64-bits. Masters are required to provide a valid address when their request signal is asserted. The PLB then arbitrates the requests and allows the upper address of the highest priority master to be gated onto the PLB_UABus signal.

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

Systems might use only 32-bit addressing in which case this bus is not necessary. Systems might also only partially implement the upper address bus to marginally increase the ease with which the bus can be addressed without additional unused signals. It is recommended that masters only implement bits that are required, for example, such as Mn_UABus(28:31) to create a 36-bit address.

Note: Some logic gating might be required to interconnect slaves with only 32-bit addressing. This can be accomplished by gating the slaves' the PLB_PAValid signal and SAValid inputs with a decode of the PLB_UABus bits implemented. Ensure that all PLB signal timings are met for any external logic that is added.

2.5.15 Mn_UABusPar, PLB_UABusPar (Upper Address Bus Parity)

The master drives this signal if the master supports parity for Mn_UABus. One parity bit is used for Mn_UABus(0:31). The master generates odd parity where the number of 1's across Mn_UABus and the parity bit is an odd number. The PLB_UABusPar signal arrives at the slave at the same time as the PLB_UABus signal. The slave must use the PLB_UABusPar signal to check the parity of the PLB_UABus signal to verify that the upper address is correct.

2.5.16 Mn_UABusParEn, PLB_UABusParEn (Upper Address Bus Parity Enable)

The master drives this signal if the master supports parity for Mn_UABus. It indicates that the Mn_UABusPar signal is valid and must be used by the slave to check parity on the PLB_UABus signal. The PLB_UABusParEn signal arrives at the slave at the same time as the PLB_UABus signal. The system must hard wire this signal to a '0' if the master does not support parity on Mn_UABus.

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 acknowledgment phase.

Note: A single-beat transfer has one transfer phase and one data acknowledgment phase associated with it. A line or burst transfer has a multiple number of transfer and data acknowledgment 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 by placing a high value on the Mn_RNW signal. When it has granted the bus to the master, the PLB arbiter gates the data on the Sl_rdDBus onto the PLB_MnRdDBus. The master waits for the slave to assert the Sl_rdDAck signal to acknowledge that the data on the read data bus is valid.

For single-beat transfers, the slave asserts the Sl_rdDAck signal for one clock cycle only. For 4-beat line transfers, a 32-bit slave asserts the Sl_rdDAck signal for four clock cycles. For 8-beat line transfers, a 32-bit slave asserts the Sl_rdDAck signal for eight clock cycles. For 16-beat line transfers, a 32-bit slave asserts the Sl_rdDAck signal for 16 clock cycles. For burst transfers, the slave asserts the Sl_rdDAck signal for as many clock cycles as the master requires through the Mn_rdBurst signal. However, in all cases, the slave indicates the end of the current transfer by asserting the Sl_rdComp signal for one clock cycle.

A slave can request the termination of a read burst transfer by asserting the Sl_rdBTerm signal during the read data cycle.

In the case of address-pipelined read transfers, the PLB arbiter asserts 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, PLB_MnRdDBus (Read-Data Bus)

This data bus transfers data between a slave and a master during a PLB read transfer. For the various bus widths, this signal is defined as shown in *Table 2-13*.

Table 2-13. PLB Read Data Bus Width

Signal Definition	Read Data Bus Width
Sl_rdDBus(0:31), PLB_rdDBus(0:31)	32-Bit
Sl_rdDBus(0:63), PLB_rdDBus(0:63)	64-Bit
Sl_rdDBus(0:127), PLB_rdDBus(0:127)	128-Bit

For a primary read request, the slave can begin to drive data on the Sl_rdDBus two clock cycles following the assertion of the Sl_addrAck signal. For a secondary read request, the slave can begin to drive data on the Sl_rdDBus two cycles following the assertion of the PLB_rdPrim signal. In both cases, the slave can drive data on the Sl_rdDBus through one clock cycle following the assertion of the Sl_rdComp signal.

Also, for a primary read request, the master must begin to sample the PLB_MnRdDAck signal two clock cycles following the assertion of the PLB_MnAddrAck signal. 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 the PLB_MnRdDBus at the end of the clock cycle in which the PLB_MnRdDAck signal is sampled asserted.

For non-line-read transfers, data must always be transferred at the requested width. Byte transfers, halfword transfers, 3-byte transfers, word transfers, up to octalword transfers are possible. However, in the case of a read transfer involving a slave device whose data path width is smaller than the width of the requested PLB transfer, the slave must first accumulate at least one word of the requested data internally and then perform the read data transfer on the PLB at the requested width.

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

2.6.2 Sl_rdDBusPar, PLB_MnRdDBusPar (Read Data Bus Parity)

For the various bus widths this bus is defined as follows.

Figure 2-1. PLB Read Data Bus Parity Width

Signal Definition	Read Data Bus Width
Sl_rdDBusPar(0:3), PLB_rdDBusPar(0:3)	32-Bit
Sl_rdDBusPar(0:7), PLB_rdDBusPar(0:7)	64-Bit
Sl_rdDBusPar(0:15), PLB_rdDBusPar(0:15)	128-Bit

The slave drives this bus if the slave supports parity for the Sl_rdDBus. One parity bit is used for each byte of the Sl_rdDBus. The slave generates odd parity, where the number of 1's across each byte of the Sl_rdDBus and the corresponding parity bit is an odd number. The PLB_MnRdDBusPar bus arrives at the master at the same time as PLB_MnRdDBus. The master must use the signals to check the parity of the PLB_MnRdDBus to verify that the read data is correct.

2.6.3 Sl_rdDBusParEn, PLB_MnRdDBusParEn (Read Data Bus Parity Enable)

The slave drives this signal if the slave supports parity for the Sl_rdDBus. It indicates that the Sl_rdDBusPar bus is valid, and it must be used by the master to check parity on the PLB_MnRdDBus. The PLB_MnRdDBusParEn signal arrives at the master at the same time as the PLB_MnRdDBus. The system integrator must hard wire this signal to a '0' if the slave does not support parity on the Sl_rdDBus.

2.6.4 Mn_rdDBusParErr (Read Data Bus Parity Error)

The master drives this signal when a parity error is detected on the PLB_MnRdDBus using the PLB_MnRdDBusPar bus. The master must latch the PLB address for the data byte with bad parity and prevent the use or distribution of the data with bad parity. When the master has asserted this signal, it must remain asserted until it is cleared by a subsequent access to the master through the device control register (DCR) interface. All of the Mn_RdDBusParErr signals must be ORed together, and the output must be connected to a central interrupt controller.

2.6.5 Sl_rdWdAddr(0:3), PLB_MnRdWdAddr(0:3) (Read Word Address)

The slave drives these signals to indicate the word address of a data word that is transferred as part of a read line transfer. Masters sample these signals in the clock cycle in which the Sl_rdDAck signal is asserted for a read line transfer. Because the PLB read data bus is a shared bus, the Sl_rdWdAddr(0:3) signals must be driven low, with logic 0's, by the slave any time that the slave is not selected for a read transfer or the SYS_plbReset signal is asserted. If it is selected for a read transfer, the slave can begin to drive the

Sl_rdWdAddr(0:3) signals with nonzero logic values two clock cycles after it has asserted the Sl_addrAck signal through one clock cycle after the assertion of the Sl_rdComp signal. Masters must only sample the necessary PLB_MnRdWdAddr signals associated with the line transfer size that is requested.

Table 2-14. 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			

For optimal performance, slaves must operate in a target-word-first mode of operation where the word that is being requested through the ABus signal is returned in the first read-data acknowledgment cycle. For transfer sizes greater than 32 bits, the target word can be within the doubleword, quadword, or octalword that is returned. The Sl_rdWdAddr(0:3) signals reflect the value of the first word in the data that is returned.

Table 2-15. PLB Sl_rdWdAddr(0:3) Signals for Target-Word-First 16-Word Transfers

ABus(26:31)	Resolved Transfer Size[1]			
	32-Bit	64-Bit	128-Bit	256-Bit
0000XX	0000	0000	0000	0000
0001XX	0001	0000	0000	0000
0010XX	0010	0010	0000	0000
0011XX	0011	0010	0000	0000
0100XX	0100	0100	0100	0000
0101XX	0101	0100	0100	0000
0110XX	0110	0110	0100	0000
0111XX	0111	0110	0100	0000
1000XX	1000	1000	1000	1000
1001XX	1001	1000	1000	1000
1010XX	1010	1010	1000	1000
1011XX	1011	1010	1000	1000
1100XX	1100	1100	1100	1000
1101XX	1101	1100	1100	1000
1110XX	1110	1110	1100	1000
1111XX	1111	1110	1100	1000

1. Resolved transfer size is the minimum size of either the slave or the master. The slave ignores the ABus(30:31) signals for line read transfers, which are therefore don't care bits. For 4-word line read transfers only, PLB_MnRdWdAddr(2:3) signals are valid. For 8-word line read transfers only, PLB_MnRdWdAddr(1:3) signals are valid.

2.6.6 Sl_rdDAck, PLB_MnRdDAck (Read Data Acknowledgment)

The slave drives this signal to indicate that the data on the Sl_rdDBus bus is valid and must be latched at the end of the current clock cycle. For a primary read request, the slave can begin to assert the Sl_rdDAck signal two clock cycles following the assertion of the Sl_addrAck signal. For a secondary read request, the slave can begin to assert the Sl_rdDAck signal two clock cycles following the assertion of the PLB_rdPrim signal.

For single-beat read transfers, the signal is asserted for one clock cycle only. For line read transfers, the signal is asserted for multiple clock cycles to fulfill the line request. For burst read transfers, this signal is asserted for as many clock cycles as the length of the burst requires, as indicated through the Mn_rdBurst signal.

Note: For line and burst transfers, the slave is not required to assert the Sl_rdDAck signal back-to-back until transfer completion. The slave can insert read data bus “wait states”. It is preferred, however, that slaves assert the Sl_rdDAck signal back-to-back until the line or burst transfer is completed.

Note: The slave must drive this signal to a low value any time that the slave is not selected, or when the slave is selected but not ready to transfer read data on the read data bus.

2.6.7 Sl_rdComp (Data Read Complete)

The slave drives this signal to indicate to the PLB arbiter that the read transfer is either complete, or will be complete by the end of the next clock cycle. The assertion of this signal causes the PLB arbiter to gate the next read request to the slaves.

For optimal performance for single or line reads, the slave must assert this signal one clock cycle before the data acknowledgment 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, this signal must 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.

For optimal performance during read burst transfers, the Sl_rdComp signal must be asserted in the cycle before the last Sl_rdDAck signal, but only if the Sl_rdBTerm signal is also asserted or has previously been asserted by the slave. If the Sl_rdBTerm is not asserted, the Sl_rdComp signal must be asserted either in the clock cycle in which the last Sl_rdDAck is asserted, or in a subsequent clock cycle (see *Section 5.1.14 Fixed-Length Burst Transfer* on page 97 for more details).

The definition of fixed length burst suggests that slaves must assert Sl_rdBTerm to allow the Sl_rdComp signal to be asserted in the clock before the last Sl_rdDAck. This leads to maximum throughput on the read data bus.

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

2.6.8 Mn_rdBurst, PLB_rdBurst (Read Burst)

The master drives this signal to control the length of a burst read transfer. A burst read of sequential bytes up to octalwords can be requested by a master on the PLB by indicating a transfer size shown in *Table 2-16* and a high value on the Mn_RNW signal.

Table 2-16. Read Burst Size

PLB_size(0:3)	Burst Size
1000	Bytes
1001	Halfwords
1010	Words
1011	Doublewords
1100	Quadwords
1101	Octalwords

The burst transfer address must be aligned on a boundary of the requested burst size. For example, a half-word burst request must start on an even or halfword address alignment.

The master can assert the Mn_rdBurst signal at any time that it is not actively signaling the end of a read burst transfer and the SYS_plbReset signal is deasserted. The Mn_rdBurst signal can be asserted whether a read or write is being requested. The PLB arbiter ensures that the PLB_rdBurst signal is valid during a primary read burst.

Typically, the Mn_rdBurst signal is asserted with or following the assertion of the Mn_request signal for a primary read transfer. To request a burst of more than one transfer, the master must guarantee that the Mn_rdBurst signal is asserted by the clock cycle following the assertion of the PLB_MnAddrAck signal. When a read burst transfer has been acknowledged by a slave, the slave starts sampling the PLB_rdBurst signal in the clock cycle following the assertion of the Sl_addrAck signal, or the PLB_rdPrim signal in the case of a read burst transfer that is acknowledged as a pipelined request. Sampling PLB_rdBurst asserted indicates to the slave that the master requires additional sequential transfers of data. Sampling PLB_rdBurst negated indicates to the slave that the master requires one, and only one, additional transfer of data.

When the first data transfer has been completed for a read 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 in which PLB_rdBurst signal is sampled negated, the slave also samples its Sl_rdDAck signal. If it is asserted, the slave terminates the transfer by asserting its Sl_rdComp signal in the following clock cycle or in a later clock cycle. If it is negated, the slave supplies one, and only one, additional data transfer in the following clock cycle (or in a later clock cycle, depending on the number of wait states) and terminates the transfer by asserting its Sl_rdComp signal. If the Sl_rdBTerm signal is or was previously asserted for the primary read burst, the Sl_rdComp signal can be asserted as early as the clock cycle before the assertion of the final Sl_rdDAck or in a subsequent clock cycle. If the Sl_rdBTerm signal is not or was not previously asserted for the primary read burst, the Sl_rdComp signal must be asserted at the same time as the final Sl_rdDAck or in a subsequent clock cycle. After the master has negated the Mn_rdBurst signal to terminate a read burst transfer, this signal must remain negated until the clock cycle following the assertion of the last Sl_rdDAck for the read burst transfer, unless the Sl_rdBTerm signal was asserted. See note 2 for more information.

Notes:

1. In the case of the same master requesting two read burst requests, where the second request is acknowledged before all of the data has been transferred for the first request (the second request is a pipelined read request), and the Sl_rdBTerm signal is not asserted, the master must guarantee that the Mn_rdBurst signal is negated during the last data transfer for the first request before this signal is reasserted for the second request. Furthermore, for the second request, the Mn_rdBurst signal must be asserted in the clock cycle immediately following the last read data acknowledgment for the first request. The PLB arbiter

guarantees that the PLB_rdBurst signal is driven to the appropriate value in the clock cycle following the assertion of the PLB_rdPrim signal. This is illustrated in *Figure 5-26 Pipelined Back-to-Back Read Burst Transfers* on page 112.

If, during a read burst transfer, the master samples Sl_rdBTerm asserted and it *does not* have a pipelined read burst transfer acknowledged or is not currently being acknowledged, it must negate the Mn_rdBurst signal in the following clock cycle. The *master* must then continue to negate the Mn_rdBurst signal until a read burst transfer is acknowledged. If a pipelined read burst transfer is acknowledged after the Sl_rdBTerm signal is sampled asserted for the primary read burst, the master must then drive the Mn_rdBurst signal to the appropriate value for the next read burst transfer in the clock cycle following the assertion of the Sl_addrAck signal.

If, during a read burst transfer, the master samples Sl_rdBTerm asserted and it *has* a pipelined read burst transfer acknowledged or is currently being acknowledged, it must drive the Mn_rdBurst signal to the appropriate value for the *next* read burst transfer in the clock cycle following the assertion of the Sl_rdBTerm signal. This is illustrated in *Figure 5-27 Pipelined Back-to-Back Fixed-Length Read Burst Transfers* on page 113.

2. In the case of a pipelined master with a nonburst cycle on the primary request and an acknowledged read burst transfer on the secondary request, the master can assert the Mn_rdBurst signal any time, including before the primary and secondary requests, However, the master must drive the Mn_rdBurst signal to the appropriate value in the clock cycle following the assertion of the Sl_addrAck signal for the pipelined read burst transfer. The PLB arbiter guarantees that the PLB_rdBurst signal is driven to the appropriate value in the clock cycle following the assertion of the PLB_rdPrim signal.
3. The Mn_rdBurst signal can remain asserted at any time the master is not signaling the end to a read burst transfer and the SYS_plbReset signal is deasserted. Slaves must ignore the PLB_rdBurst signal unless performing a primary read burst.
4. Slaves must decode and accept all values of Burst Size and provide their full data bus width of data per transfer. An example of this is illustrated in *Figure 5-51 64-Bit Master 4-Doubleword Burst Read from a 32-Bit Slave* on page 152.

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

2.6.9 Sl_rdBTerm, PLB_MnRdBTerm (Read Burst Terminate)

The slave asserts this signal to indicate to a master that the current burst read transfer in progress must be terminated. The slave can assert this signal starting the clock cycle following the assertion of the Sl_addrAck signal (for a primary read burst request) or the PLB_rdPrim signal (for a secondary read burst request), up to and including the clock cycle in which the Sl_rdComp signal or the final Sl_rdDAck signal is asserted, whichever occurs first. 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. When the slave asserts the Sl_rdBTerm signal, it must no longer sample the PLB_rdBurst signal for the current transfer. If the PLB_rdBurst signal is active when the slave asserts the Sl_rdBTerm signal, the slave supplies one, and only one, additional piece of data. The transfer is then completed when the Sl_rdDAck signal is asserted.

Notes:

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.

2. In the case of a master requesting two read burst requests, where the second request is acknowledged before all the data that is being transferred for the first request (the second request is a secondary read request), if the PLB_MnRdBTerm signal is or has been previously asserted, the master must sample the PLB_MnRdBTerm signal 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, because the Sl_rdComp signal can be asserted in the clock before the last Sl_rdDAck signal of the first read request.

2.6.10 PLB_rdPrim (Read Secondary to Primary Indicator)

The PLB arbiter asserts this signal to indicate that a secondary read request, which has already been acknowledged by a slave, can now be considered a primary read request. Slaves that support address pipelining must begin to sample this signal in the clock cycle following the assertion of the Sl_addrAck signal in response to the assertion of the PLB_SAValid signal. When transferring data for a secondary read request, the slave can begin to drive the Sl_rdDBus(0:31) bus two clock cycles after it has sampled asserted the PLB_rdPrim signal. For arbiters that implement only a single level of read pipelining, this signal only needs to be 1 bit wide. For arbiters that implement more than one level of read pipelining, for each pipelined read, the arbiter must keep track of which slave acknowledged the pipelined request. This tracking is accomplished by the arbiter sampling all slave Sl_addrAck signals independently. In this case, each slave then receives its own PLB_rdPrim(n) signal so that the arbiter can notify only the slave that responded to a particular pipelined request. The arbiter notifies that slave that it can now be considered the primary transfer and to begin driving data onto the read data bus two clock cycles after it has received it PLB_rdPrim(n) signal.

Note: If there is no secondary read request on the PLB or a slave has not acknowledged a secondary read request, the PLB_rdPrim signal is not asserted.

2.7 PLB Write Data Bus Signals

The PLB write data cycle is divided into two phases: transfer and data acknowledgment. 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 acknowledgment phase.

Note: A single-beat transfer has one transfer phase and one data acknowledgment phase associated with it. A line or burst transfer has a multiple number of transfer and data acknowledgment 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 bus. When it has granted the bus to the master, the PLB arbiter gates the data on Mn_wrDBus onto the PLB_wrDBus bus. The master then waits for the slave to assert the Sl_wrDAck signal to acknowledge the latching of the write data.

For single-beat transfers, the slave asserts the Sl_wrDAck signal for one clock cycle only. For 2-beat, transfers, the slave asserts the Sl_wrDAck signal for two clock cycles. For 4-beat transfers, the slave asserts the Sl_wrDAck signal for four clock cycles. For 8-beat transfers, the slave asserts the Sl_wrDAck signal for eight clock cycles. And for 16-beat transfers, the slave asserts the Sl_wrDAck signal for 16 clock cycles. For burst transfers, the slave asserts the Sl_wrDAck signal for as many clock cycles as the master that is using the Mn_wrBurst signal requires. But in all cases, the slave indicates the end of the current transfer by asserting the Sl_wrComp signal for one clock cycle.

A slave can request the termination of a write burst transfer by asserting the Sl_wrBTerm signal during the write data cycle.

In the case of address-pipelined write transfers, the PLB arbiter asserts 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.

2.7.1 Mn_wrDBus, PLB_wrDBus (Write Data Bus)

This data bus is used to transfer data between a master and a slave during a PLB write transfer. For the various bus widths, this signal is defined as shown in *Table 2-17*.

Table 2-17. PLB Write Data Bus Width

Signal Definition	Write Data Bus Width
Mn_wrDBus(0:31), PLB_wrDBus(0:31)	32-Bit
Mn_wrDBus(0:63), PLB_wrDBus(0:63)	64-Bit
Mn_wrDBus(0:127), PLB_wrDBus(0:127)	128-Bit

For a primary write request, the master must place the first bytes of data to be written on the Mn_wrDBus bus in the same clock cycle in which the Mn_request signal is first asserted. For a secondary write request, the master must place the first bytes of data to be written on Mn_wrDBus in the clock cycle immediately following the last data acknowledgment for the primary write request.

When a master has requested a write transfer, it must begin to sample the PLB_Mn_WrDAck signal continuously.The master must then retain the data on Mn_wrDBus until the end of the clock cycle in which the PLB_MnWrDAck signal is sampled asserted.

For nonline, nonburst transfers (Mn_size(0:3) ‘0000’), the master must retain the data on the Mn_wrDBus until the end of the clock cycle in which the PLB_MnWrDAck signal is first sampled asserted, at which time the master considers the transfer to be complete.

For line write transfers, the master must retain the first words of data (word 0 in the 32-bit case) on the Mn_wrDBus until the end of the clock cycle in which the PLB_MnWrDAck signal is first sampled asserted. The master then continues to place a new word of data, such as word 1, word 2, and so on, in the 32-bit case, on Mn_wrDBus every time the PLB_MnWrDAck signal is sampled asserted, until this signal is sampled asserted for the last word of the line, at which time the master considers the transfer to be complete.

For burst write transfers, the master must retain the first byte, halfword, word, doubleword, or quadword of data (data 0) on Mn_wrDBus until the end of the clock cycle in which the PLB_MnWrDAck signal is first sampled asserted. The master then continues to place a new byte, halfword, word, doubleword, or quadword of data (data 1, data 2, and so on) on Mn_wrDBus every time the PLB_MnWrDAck signal is sampled asserted, until the burst transfer is completed.

Note: In the case of write burst transfers that are a smaller size than the bus width, the master is required to place the write data on the correct memory alignment on 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), and so on. In general, masters must perform burst transfers at the full size of the their write data bus for optimal PLB throughput.

2.7.2 Mn_wrDBusPar, PLB_wrDBusPar (Write Data Bus Parity)

For the various bus widths, the wrDBusPar bus is defined as shown in *Table 2-18*.

Table 2-18. PLB Write Data Bus Parity Width

Signal Definition	Write Data Bus Width
Mn_wrDBusPar(0:3), PLB_wrDBusPar(0:3)	32-Bit
Mn_wrDBusPar(0:7), PLB_wrDBusPar(0:7)	64-Bit
Mn_wrDBusPar(0:15), PLB_wrDBusPar(0:15)	128-Bit

The master drives the wrDBusPar bus if the master supports parity for Mn_wrDBus. One parity bit is used for each byte of the Mn_wrDBus. The master generates odd parity, where the number of 1's across each byte of the Mn_wrDBus and the corresponding parity bit is an odd number. The PLB_wrDBusPar bus arrives at the slave at the same time as PLB_wrDBus. The slave must use these signals to check the parity of the PLB_wrDBus to verify that the write data is correct.

2.7.3 Mn_wrDBusParEn, PLB_wrDBusParEn (Write Data Bus Parity Enable)

The master drives this signal if the master supports parity for the Mn_wrDBus. It indicates that the Mn_wrDBusPar bus is valid, and must be used by the slave to check parity on the PLB_wrDBus. The PLB_wrDBusParEn signal arrives at the slave at the same time as the PLB_wrDBus. The system integrator must hard wire this signal to a '0' if the master does not support parity on Mn_wrDBus.

2.7.4 Sl_wrDAck, PLB_MnWrDAck (Write Data Acknowledge)

The slave drives this signal for a write transfer to indicate that the slave no longer requires the data that is currently on the PLB_wrDBus. The slave no longer requires the data because the slave has either already latched the data or will latch the data at the end of the current clock cycle. For a primary write request, the slave can begin to assert the Sl_wrDAck signal in the clock cycle in which the Sl_addrAck signal is asserted. For a secondary write request, the slave can begin to assert the Sl_wrDAck signal in the clock cycle immediately following the assertion of the PLB_wrPrim signal.

For single-beat write transfers, the signal is asserted for one clock cycle only. For line write transfers, the signal is asserted for 1, 2, 4, 8, or 16 clock cycles depending on bus width and line size. For burst write transfers, this signal is asserted for as many clock cycles as the length of the burst requires, as indicated through the Mn_wrBurst signal.

Notes:

1. For line and burst transfers the slave is not required to assert the Sl_wrDAck signal back-to-back until transfer completion. The slave can insert write data bus "wait states." It is preferred, however, that slaves assert the Sl_wrDAck signal back-to-back until the line or burst transfer is completed.
2. 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 is not ready to transfer write data on the write data bus.

2.7.5 Sl_wrComp (Data Write Complete)

The slave asserts this signal 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 can assert this signal in the same clock cycle in which the request was granted if only one data transfer on the PLB is required and the address and data acknowledgment signals are also asserted.

2.7.6 Mn_wrBurst, PLB_wrBurst (Write Burst)

The master drives this signal to control the length of a burst write transfer. Unless the master is holding Mn_wrBurst inactive to signal the end of a previously acknowledged write burst, Mn_wrBurst must be asserted with the Mn_request signal when a write burst request (of more than one) is made. A burst write of sequential bytes up to octalwords can be requested by a master on the PLB by indicating a transfer size as shown in *Table 2-19*. The burst transfer address must be aligned on a boundary of the requested burst size. For example a halfword burst request must start on an even or halfword address alignment.

Table 2-19. Write Burst Size

PLB_size(0:3)	Burst Size
1000	Bytes
1001	Halfwords
1010	Words
1011	Doublewords
1100	Quadwords
1101	Octalwords

When a slave acknowledges write burst transfer, the slave samples 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 transfer of data. A low value indicates that the current transfer is the last sequential transfer that the master requires to write. This signal can be asserted only during a burst write transfer and must remain negated at all other times.

After 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 completes the burst transfer by asserting the Sl_wrComp signal. Because 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 when it has been negated and until the Sl_wrComp signal has been asserted for the current burst transfer.

Notes:

1. In the case of a master requesting two back-to-back write burst requests, where the second request is acknowledged before all the data being transferred for the first request (the second request is a secondary write request), the master must guarantee that Mn_wrBurst is reasserted for the second request in the clock cycle immediately following the last data acknowledge for the first request. This is illustrated in *Figure 5-28 Pipelined Back-to-Back Write Burst Transfers* on page 114.
2. In the case of a master requesting two back-to-back write requests, where the first request is a nonburst and the second request is a burst and is acknowledged before all the data being transferred for the first request (the second request is a secondary write request), the master must assert Mn_wrBurst with the second request.

3. Slaves must decode all values of burst size and accept their full data bus width of data per transfer. An example of this is illustrated in *Figure 5-54 64-Bit Master 4-Doubleword Burst Write to a 32-Bit Slave* on page 155.

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

2.7.7 Sl_wrBTerm, PLB_MnWrBTerm (Write Burst Terminate)

The slave asserts this signal to indicate that the current primary burst write transfer in progress must be terminated by the master. The slave can assert this signal with the Sl_addrAck signal, or during any clock cycle thereafter, up to and including the clock cycle in which the last the Sl_wrDAck signal is asserted for the current transfer. This signal must only be asserted for one clock cycle per termination request. For secondary acknowledges, the slave needs to wait until the clock after the assertion of the PLB_wrPrim signal for the start of the window which it can assert Sl_wrBterm.

If the Sl_wrBTerm signal is asserted and the Mn_wrBurst signal is asserted, 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 negated, before or coincident with the assertion of the final Sl_wrDAck signal, and the slave asserts the Sl_wrComp signal coincident with or following the assertion of the final Sl_wrDAck signal. This completes the burst write transfer.

If the Sl_wrBTerm signal is asserted and the Mn_wrBurst signal is negated then the master must ignore the Sl_wrBTerm signal and the slave asserts the Sl_wrComp signal incident with or following the final assertion of the Sl_wrDAck signal. This completes the burst write transfer.

2.7.8 PLB_wrPrim (0:n) (Write Secondary to Primary Indicator)

The PLB arbiter asserts this signal to indicate that a secondary, or pipelined, write request can be considered a primary write request in the clock cycle that follows. Slaves that support address pipelining must begin to sample this signal in the clock cycle in which the PLB_SAValid signal is asserted for a secondary write request. In the clock cycle following the assertion of the PLB_wrPrim signal, the PLB arbiter gates the secondary write data onto the PLB_wrDBus, provided the secondary write request has already been acknowledged, or is currently being acknowledged and the Mn_abort signal is not asserted. Accordingly, the slave can begin to assert its Sl_wrDAck signal for a secondary write request in the clock cycle following the assertion of the PLB_wrPrim signal. For arbiters that implement only a single level of write pipelining, this signal only needs to be 1 bit wide. For arbiters that implement greater than one level of write pipelining for each pipelined write transfer, the arbiter must keep track of which slave acknowledged the pipelined request. To keep track of which slave acknowledged the pipelined request, the arbiter samples all slave Sl_addrAck signals independently. Each slave then receives its own PLB_wrPrim(n) signal. This allows the arbiter to notify only the slave that responded to a particular pipelined request. That pipelined request can now be considered the primary transfer and begin latching data from the write data bus in the clock following the assertion of its PLB_wrPrim(n) signal.

Note: If a secondary write request is either aborted or not address acknowledged in the same clock cycle in which the Sl_wrComp signal is asserted for a primary write request, the PLB_wrPrim signal is asserted by the arbiter and must be ignored by the slaves.

2.8 Additional Slave Output Signals

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

2.8.1 Sl_MBusy(0:n), PLB_MBusy(0:n) (Master Busy)

The slave drives these signals 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: Sl_MBusy(0) corresponds to Master ID0, Sl_MBusy1 corresponds to master ID1, and so on. The slave must 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 is asserted in the clock cycle following the assertion of the Sl_addrAck signal and remains asserted until the final Sl_rdDAck is asserted by the slave. During write transfers, the Sl_MBusy signal is asserted in the clock cycle following the assertion of the Sl_addrAck signal. It must remain asserted until the write transfer is completed from the perspective of the slave. Usually, this is the completion of the write data transfer on the slave bus. Thus, the signal can remain asserted following the assertion of the last Sl_wrDAck signal.

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 acknowledgment cycle, while the request is in the queue and while the request is being transferred. If the queue can store multiple requests, the slave is 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 ORed together and sent to the appropriate master. The system integrator must OR together all of the slave busy outputs for each master and send one busy signal to each master on the PLB.

A master can use the master busy signals to determine if the slave has completed all of its transfers.

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

2.8.2 Sl_MRdErr(0:n), PLB_MRdErr(0:n) (Master Read Error)

The slave drives these signals to indicate that the slave has encountered an error during a read transfer that this master initiated. Each slave is required to drive a separate error signal for each master that is attached on the PLB bus: Sl_MRdErr(0) corresponds to master ID 0, Sl_MRdErr(1) corresponds to master ID 1, and so on. The slave drives this signal for one clock cycle for each error that is encountered while trying to complete the transfer.

The error signal is guaranteed to be asserted during the data acknowledge phase of the transfer coincident with the Sl_rdDack signal. Software must examine the Slave Error Address Register (SEAR) and Slave Error Status Register (SESR) in each slave to determine in which transfer the error occurred. The slave must latch the master ID input to determine which master's error line must be asserted. The system integrator must 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_MRdErr(0:n) and PLB_MRdErr(0:n) signals is determined by the number of masters supported by the particular PLB-based system.

2.8.3 Sl_MWrErr(0:n), PLB_MWrErr(0:n) (Master Write Error)

The slave drives these signals to indicate that the slave has encountered an error during a write 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: Sl_MWrErr(0) corresponds to master ID 0, Sl_MWrErr(1) corresponds to master ID 1, and so on. The slave drives this signal for one clock cycle for each error that is encountered while trying to complete the transfer.

The error signal is guaranteed to be asserted during the data acknowledgment phase of the transfer coincident with the Sl_wrDAck signal. Software must examine the SEAR and SESR in each slave to determine in which transfer the error occurred. The slave must latch the master ID input to determine which error line of the master must be asserted. The system integrator must 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_MWrErr(0:n) and PLB_MWrErr(0:n) signals is determined by the number of masters supported by the particular PLB-based system.

2.8.4 Sl_MIRQ(0:n), PLB_MIRQ(0:n) (Master Interrupt Request)

The slave drives these signals. When they are asserted, these signals indicate that the slave has encountered an event that it has considers important to the master. This can be because of an operation that the master did or did not initiate. Each slave is required to assert a separate interrupt request (IRQ) signal for each master which is attached on the PLB bus: Sl_MIRQ(0) corresponds to master ID 0, Sl_MIRQ(1) corresponds to master ID 1, and so on. When the slave has asserted this signal to the master, it must remain asserted until it is cleared by a subsequent access to the slave. This can be through the PLB or DCR interface.

The IRQ signal can be asserted at any time independent of the state of the master. One typical use of this signal is for latent, posted write cycles that did not complete gracefully on a slave bus that is attached to the PLB. This signal must be negated in response to reset. It might be necessary for software to examine the SEAR and SESR in each slave to determine in which transfer the IRQ occurred as result of the completion. Other implementations of these signals are possible, but it is up to the system integrator to ensure expected operation. The slave must latch the master ID input to determine which master IRQ line must be asserted. The system integrator must OR together all of the slave IRQ inputs for each master and send one IRQ signal to each master on the PLB.

Notes:

1. The width of the Sl_MIRQ(0:n) and PLB_MIRQ(0:n) signals is determined by the number of masters supported by the particular PLB-based system.
2. The slave drives this signal when a parity error is detected on the PLB_wrDBus using the PLB_wrDBusPar bus. The slave must latch the PLB address for the data byte with bad parity and prevent the use or distribution of the data with bad parity.

2.8.5 Sl_ABusParErr (Address Parity Error)

The slave drives this signal when a parity error is detected on any of the following buses; PLB_UABus with PLB_UABusPar, PLB_ABus with PLB_ABusPar, or PLB_BE with PLB_BEPar. When the slave has asserted this signal, it must remain asserted until it is cleared by a subsequent access to the slave. This access can be through the PLB or DCR interface. All of the Sl_ABusParErr signals must be ORed together, and the output must be connected to a central interrupt controller.

2.9 Summary of Signals That Can Be Considered Optional

Some signals that are defined for the master and slave interfaces are required by a particular master or slave device. It is up to the core developer to ultimately determine which signals to implement. The developer must perform a careful analysis in the context of system integration. Signals that are inputs to the PLB must be appropriately tied in the absence of an output from a master or slave device. These signal are listed in *Table 2-20*.

Table 2-20. Summary of Optional PLB Signals (Sheet 1 of 2)

Signal Name	Interface	Description	Page
Mn_abort	Master n	Master n abort bus request indicator	32
Mn_ABusPar	Master n	Master n address bus parity	47
Mn_ABusParEn	Master n	Master n address bus parity enable	47
Mn_UABus(0:31)	Master n	Master n upper address bus	47
Mn_UABusPar	Master n	Master n upper address bus parity	47
Mn_UABusParEn	Master n	Master n upper address bus parity enable	47
Mn_BEPar	Master n	Master n byte enables parity	41
Mn_BEParEn	Master n	Master n byte enables parity enable	41
Mn_busLock	Master n	Master n bus lock	28
Mn_lockErr	Master n	Master n lock error indicator	46
Mn_MSize(0:1)	Master n	Master data bus size	43
Mn_priority(0:1)	Master n	Master n bus request priority	28
Mn_rdDBusParErr	Master n	Master n read data bus parity error	49
Mn_rdBurst	Master n	Master n burst read transfer indicator	51
Mn_size(0:3)	Master n	Master n transfer size	41
Mn_TAttribute(0:15)	Master n	Master n transfer attribute bus	41
Mn_type(0:2)	Master n	Master n transfer type	42
Mn_wrDBusPar	Master n	Master n write data bus parity	55
Mn_wrDBusParEn	Master n	Master n write data bus parity enable	56
Mn_wrBurst	Master n	Master n burst write transfer indicator	57
Sl_ABusParErr	Slave	Slave address parity error	60
Sl_MRdErr(0:n)	Slave	Slave read error indicator	59
Sl_MWrErr(0:n)	Slave	Slave write error indicator	59
Sl_MIRQ(0:n)	Slave	Slave interrupt indicator	60
Sl_rdBTerm	Slave	Slave terminate read burst transfer	53
Sl_rdDBusPar	Slave	Slave read data parity	49
Sl_rdDBusParEn	Slave	Slave read data parity enable	49
Sl_rdWdAddr(0:3)	Slave	Slave read word address	49
Sl_rearbitrate	Slave	Slave rearbitrate bus indicator	32

Table 2-20. Summary of Optional PLB Signals (Sheet 2 of 2)

Signal Name	Interface	Description	Page
Sl_SSize(0:1)	Slave	Slave data bus size	43
Sl_wait	Slave	Slave wait indicator	31
Sl_wrBTerm	Slave	Slave terminate write burst transfer	58

3. PLB Interfaces

The processor local bus (PLB) I/O signals are grouped under the following interface categories depending on their function.

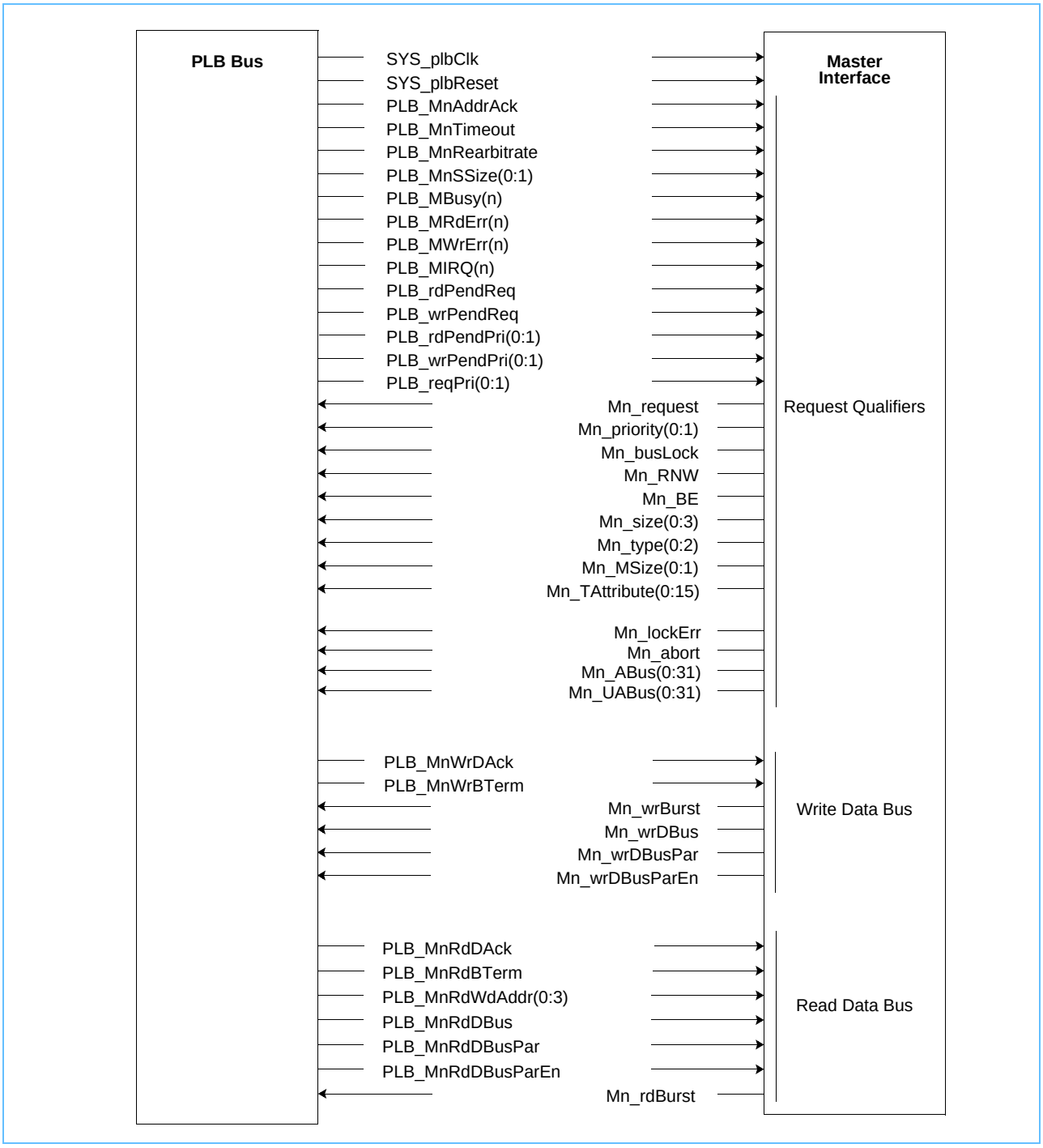
- PLB master interface
- PLB slave interface
- PLB arbiter interface

For a detailed functional description of various signals see *Section 2 PLB Signals* on page 23.

3.1 PLB Master Interface

Figure 3-1 PLB Master Interface on page 64 demonstrates all PLB master interface input/output signals. See *Section 2 PLB Signals* for a detailed functional description of the signals. The use of the PLB_rdPendReq, PLB_wrPendReq, PLB_rdPendPri and PLB_wrPendPri signals by a master is optional and not required by the PLB architecture.

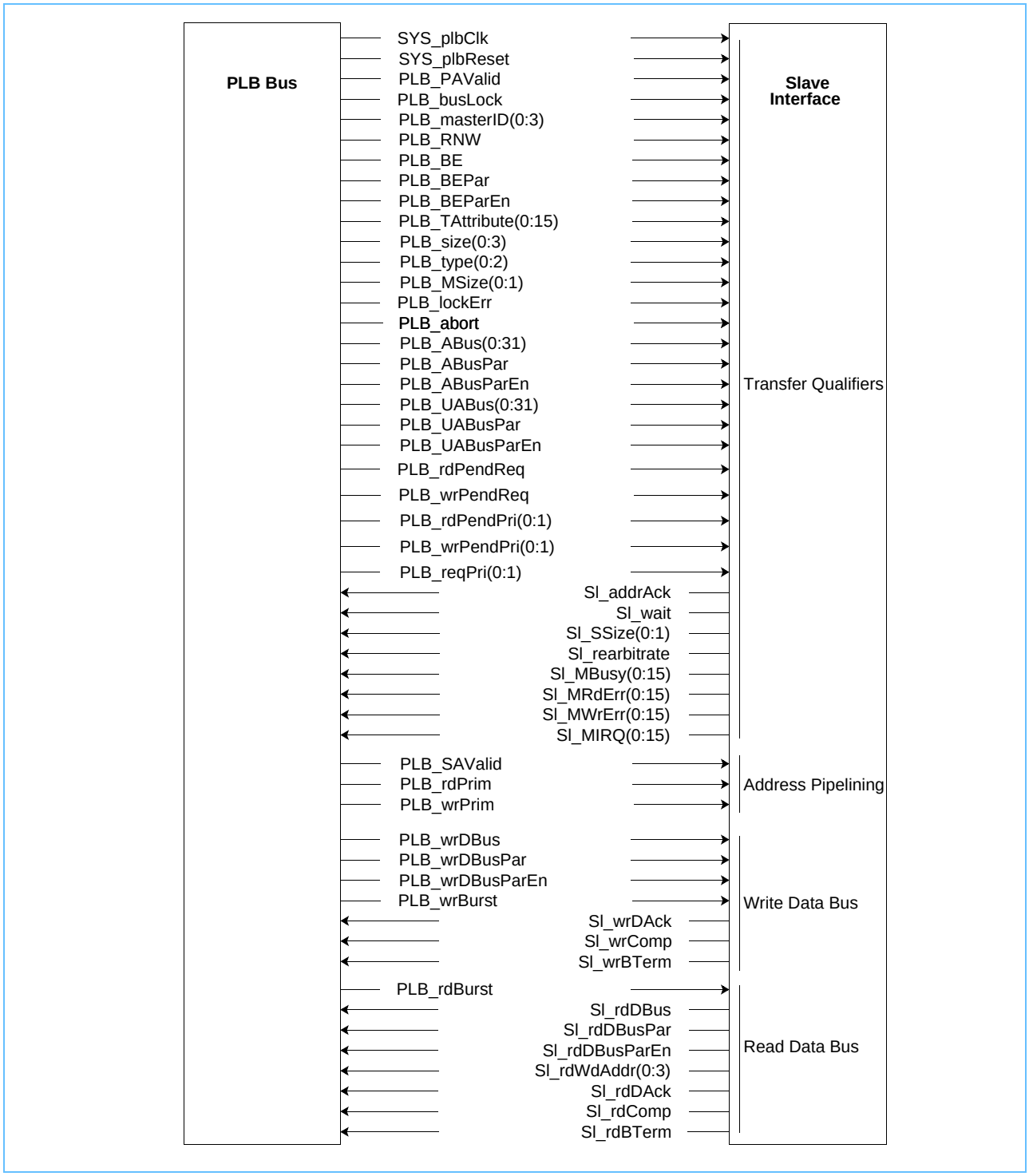
Figure 3-1. PLB Master Interface



3.2 PLB Slave Interface

Figure 3-2 demonstrates all PLB slave interface input/output signals. See *Section 2 PLB Signals* on page 23 for a detailed functional description of the signals.

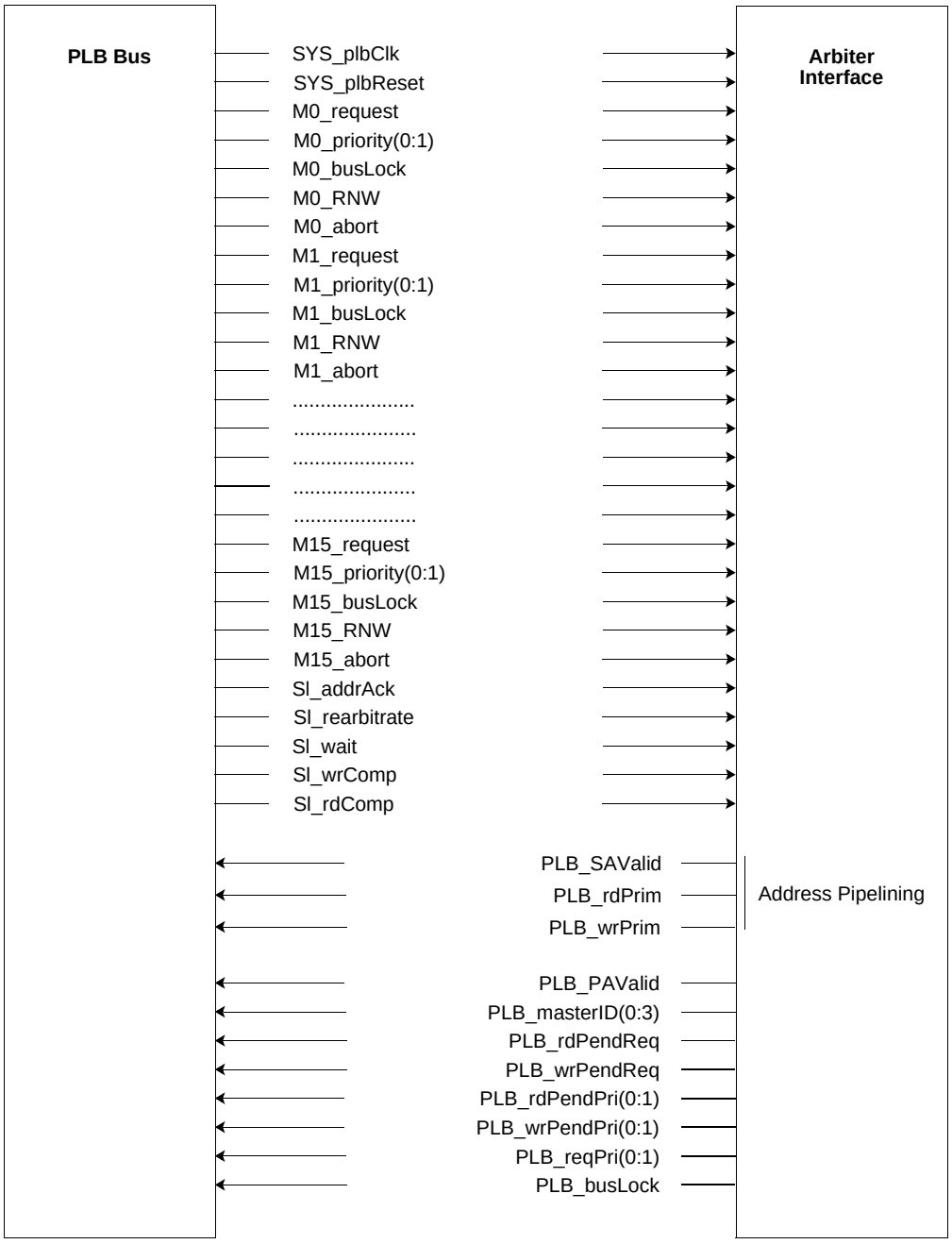
Figure 3-2. PLB Slave Interface



3.3 PLB Arbiter Interface

Figure 3-3 demonstrates all PLB arbiter interface input/output signals. See *Section 2 PLB Signals* on page 23 for a detailed functional description of the signals.

Figure 3-3. PLB Arbiter Interface



4. PLB Timing Guidelines

Multiple timing guidelines (1-cycle, 2-cycle acknowledgment, and 3-cycle acknowledgment) are described here to present a set of implementation options. For a given frequency and target technology, the single cycle guideline might not be achieved. The 2-cycle guideline breaks the single cycle request, the arbitration/acknowledgment path, into two clocks. The 3-cycle acknowledgment breaks the single cycle, the arbitration/acknowledgment path, into three clocks. This allows for higher frequency implementations of the processor local bus (PLB) but with increased address acknowledgment latency at each step. It is very important to consider which guideline is to be used and ensure that all components adhere, or are compatible with, the chosen guideline at the target frequency.

4.1 1-Cycle Acknowledgment 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. This guideline is provided in an attempt to maximize bus performance. Perform timing analysis early on all Core+ASIC chips using the PLB bus core 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 core user manuals for details.

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

Note: These definitions assume that there are 0 ns 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 core. For inputs, these delays represent the arrival time of the input relative to a 0 ns delayed clock.

4.1.1 PLB Master 1-Cycle Timing Guidelines

Table 4-1 describes PLB master signal timing guidelines.

Table 4-1. PLB Master 1-Cycle TIming Guidelines

Signal Name	Driven By	Output Valid	Received
Mn_abort	PLB master n	Late -	PLB arbiter
Mn_ABus	PLB master n	Early	PLB arbiter
Mn_ABusPar	PLB master n	Early	PLB arbiter
Mn_ABusParEn	PLB master n	Early	PLB arbiter
Mn_BE	PLB master n	Early	PLB arbiter
Mn_BEPar	PLB master n	Early	PLB arbiter
Mn_BEParEn	PLB master n	Early	PLB arbiter
Mn_busLock	PLB master n	Early	PLB arbiter
Mn_lockErr	PLB master n	Early	PLB arbiter
Mn_MSize	PLB master n	Early	PLB arbiter
Mn_priority	PLB master n	Begin	PLB arbiter
Mn_rdDBusParErr	PLB master n	Early	rdDBusParErr(n) OR
Mn_rdBurst	PLB master n	Early	PLB arbiter
Mn_request	PLB master n	Begin	PLB arbiter
Mn_RNW	PLB master n	Begin	PLB arbiter
Mn_TAttribute	PLB master n	Early	PLB arbiter
Mn_size	PLB master n	Early	PLB arbiter
Mn_type	PLB master n	Early	PLB arbiter
Mn_wrBurst	PLB master n	Early	PLB arbiter
Mn_wrDBus	PLB master n	Early	PLB arbiter
Mn_wrDBusPar	PLB master n	Early	PLB arbiter
Mn_wrDBusParEn	PLB master n	Early	PLB arbiter
Mn_UABus	PLB master n	Early	PLB arbiter
Mn_UABusParEn	PLB master n	Early	PLB arbiter

4.1.2 PLB Arbiter 1-Cycle Timing Guidelines

Table 4-2 describes PLB arbiter signal timing guidelines.

Table 4-2. PLB Arbiter 1-Cycle Timing Guidelines (Sheet 1 of 3)

Signal Name	Driven By	Output Valid	Received by
PLB_abort	PLB arbiter	Late	Slaves
PLB_ABus	PLB arbiter	Middle	Slaves
PLB_ABusPar	PLB arbiter	Middle	Slaves
PLB_ABusParEn	PLB arbiter	Middle	Slaves

Table 4-2. PLB Arbiter 1-Cycle Timing Guidelines (Sheet 2 of 3)

Signal Name	Driven By	Output Valid	Received by
PLB_BE	PLB arbiter	Middle	Slaves
PLB_BEPar	PLB arbiter	Middle	Slaves
PLB_BEParEn	PLB arbiter	Middle	Slaves
PLB_busLock	PLB arbiter	Middle +	Slaves
PLB_lockErr	PLB arbiter	Middle	Slaves
PLB_MasterID	PLB arbiter	Middle	Slaves
PLB_MnAddrAck	PLB arbiter	Late	PLB master n
PLB_MBusy(n)	OR Gate	Early +	PLB master n
PLB_MRdErr(n)	OR Gate	Early +	PLB master n
PLB_MWrErr(n)	OR Gate	Early +	PLB master n
PLB_MIRQ(n)	OR Gate	Early +	PLB master n
PLB_MnRdBTerm	PLB arbiter	Middle	PLB master n
PLB_MnRdDAck	PLB arbiter	Early +	PLB master n
PLB_MnRdDBus	PLB arbiter	Early +	PLB master n
PLB_MnRdDBusPar	PLB arbiter	Early +	PLB master n
PLB_MnRdDBusParEn	PLB arbiter	Early +	PLB master n
PLB_MnRdWdAddr	PLB arbiter	Early +	PLB master n
PLB_MnRearbitrate	PLB arbiter	Late	PLB master n
PLB_MnSSize	PLB arbiter	Late	PLB master n
PLB_MnTimeOut	PLB arbiter	Begin	PLB master n
PLB_MnWrBTerm	PLB arbiter	Late	PLB master n
PLB_MnWrDAck	PLB arbiter	Late	PLB master n
PLB_MSize	PLB arbiter	Middle	Slaves
PLB_PAValid	PLB arbiter	Middle	Slaves
PLB_rdPendPri	PLB arbiter	Early +	Masters/Slaves
PLB_rdPendReq	PLB arbiter	Early	Masters/Slaves
PLB_wrPendPri	PLB arbiter	Early +	Masters/Slaves
PLB_wrPendReq	PLB arbiter	Early	Masters/Slaves
PLB_rdBurst	PLB arbiter	Early +	Slaves
PLB_rdPrim	PLB arbiter	Middle	Slaves
PLB_reqPri	PLB arbiter	Middle	Slaves
PLB_RNW	PLB arbiter	Middle	Slaves
PLB_SAValid	PLB arbiter	Middle	Slaves
PLB_size	PLB arbiter	Middle	Slaves
PLB_type	PLB arbiter	Middle	Slaves
PLB_TAttribute	PLB arbiter	Middle	Slaves
PLB_wrBurst	PLB arbiter	Middle	Slaves

Table 4-2. PLB Arbiter 1-Cycle Timing Guidelines (Sheet 3 of 3)

Signal Name	Driven By	Output Valid	Received by
PLB_wrDBus	PLB arbiter	Middle +	Slaves
PLB_wrDBusPar	PLB arbiter	Middle +	Slaves
PLB_wrDBusParEn	PLB arbiter	Middle +	Slaves
PLB_wrPrim	PLB arbiter	End	Slaves
PLB_UABus	PLB arbiter	Middle	Slaves
PLB_UABusPar	PLB arbiter	Middle	Slaves
PLB_UABusParEn	PLB arbiter	Middle	Slaves

4.1.3 PLB Slave 1-Cycle Timing Guidelines

Table 4-3 describes PLB slave signal timing guidelines.

Table 4-3. PLB Slave 1-Cycle Timing Guidelines

Signal Name	Driven By	Output Valid	Received
Sl_addrAck	Slaves	Late -	PLB arbiter
Sl_ABusParErr	Slaves	Early	ABusParErr OR
Sl_MBusy(n)	Slaves	Early	MBusy(n) OR
Sl_MRdErr(n)	Slaves	Early	MRdErr(n) OR
Sl_MWrErr(n)	Slaves	Early	MWrErr(n) OR
Sl_MIRQ(n)	Slaves	Early	MIRQ(n) OR
Sl_rdBTerm	Slaves	Middle -	PLB arbiter
Sl_rdComp	Slaves	Early	PLB arbiter
Sl_rdDAck	Slaves	Early	PLB arbiter
Sl_rdDBus	Slaves	Early	PLB arbiter
Sl_rdDBusPar	Slaves	Early	PLB arbiter
Sl_rdDBusParEn	Slaves	Early	PLB arbiter
Sl_rdwdAddr	Slaves	Early	PLB arbiter
Sl_rearbitrate	Slaves	Late -	PLB arbiter
Sl_SSize	Slaves	Late -	PLB arbiter
Sl_wait	Slaves	Late -	PLB arbiter
Sl_wrBTerm	Slaves	Late -	PLB arbiter
Sl_wrComp	Slaves	Late	PLB arbiter
Sl_wrDAck	Slaves	Late -	PLB arbiter

4.2 2-Cycle Acknowledgment Timing Guidelines

The PLB signal timing guidelines described in this section are based on a single-clock-cycle arbitration, referred to as A1, followed by a single cycle address and data transfer, referred to as A2, across the PLB bus. These guidelines are provided in an attempt to maximize bus frequency and promote the reusability of PLB masters and slaves at higher frequencies and various technologies. Perform timing analysis early on all Core+ASIC chips using the PLB bus core 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 core user manuals for details.

Begin	Signal is valid within 8% of the clock cycle from the rise of the Sys_plbClk signal.
Early	Signal is valid within 18% of the clock cycle from the rise of the Sys_plbClk signal.
Middle	Signal is valid within 43% 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.

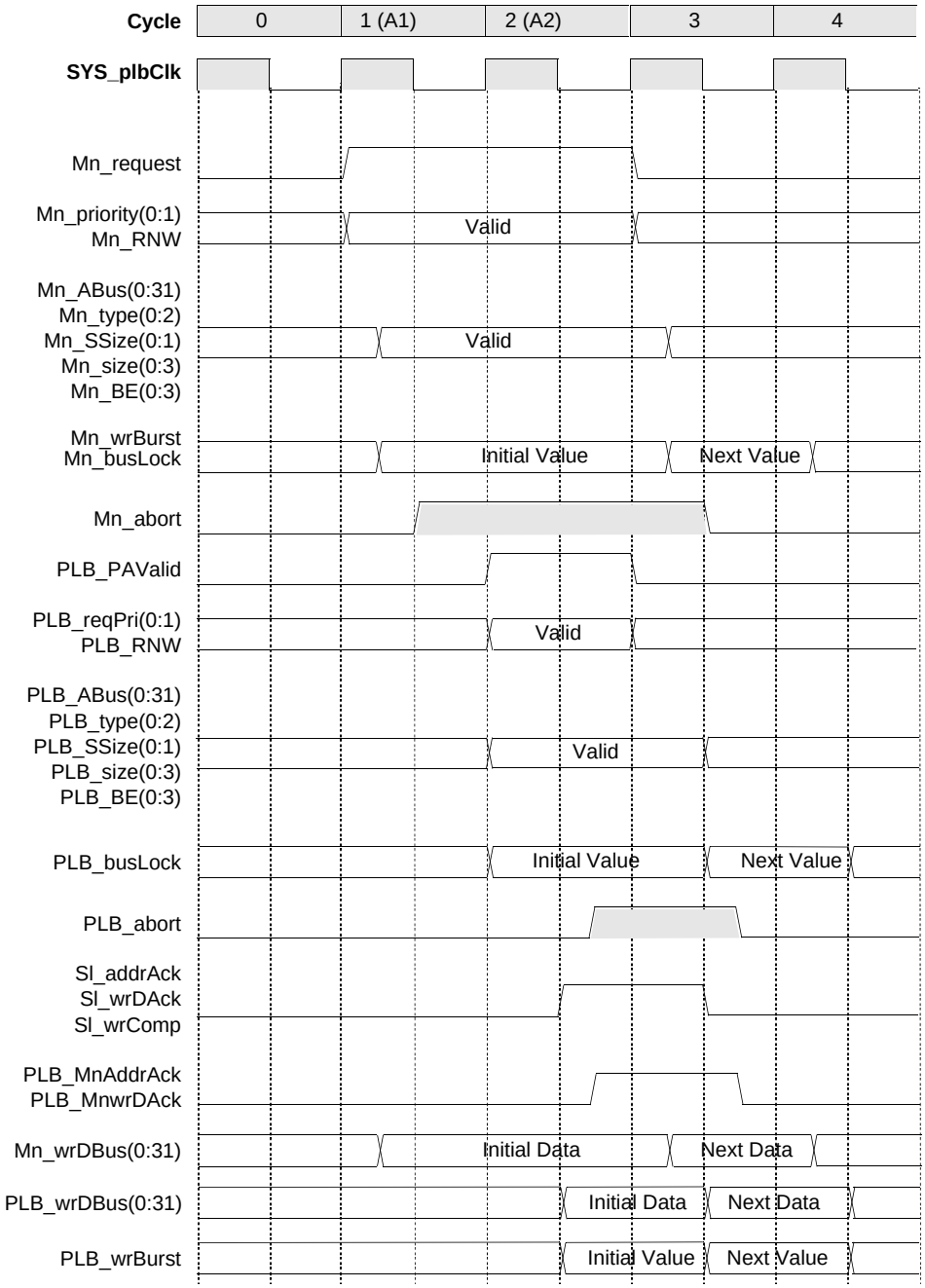
Notes:

1. These definitions assume that there are 0 ns 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 core. For inputs, these delays represent the arrival time of the input relative to a 0 ns delayed clock.
2. Signals that are identified as Begin(A2) can be considered. Begin timings during A2 and subsequent cycles before and including the cycle in which the slave acknowledges the master request. In the cycle following the assertion of the Sl_addrAck signal, all signals that are specified as Begin(A2), except the PLB_busLock signal, are no longer sampled by the slave. The PLB_busLock signal is then considered to be a middle signal until the completion of all data transfers that are associated with the initial master request.

4.2.1 Generic 2-Cycle Acknowledgment Arbitration

Figure 4-1 Generic 2-Cycle Acknowledgment PLB Arbitration on page 72 shows the operation of a generic 2-cycle acknowledgment PLB arbitration. The master asserts its request at the start of A1. The PLB then performs arbitration and, if the Mn_abort signal is deasserted at the end of A1, asserts the PLB_PAValid signal at the start of A2. All transfer qualifiers are valid at the start of A2, and the rest of the transfer proceeds as in a single cycle transfer.

Figure 4-1. Generic 2-Cycle Acknowledgment PLB Arbitration



4.2.2 PLB Master 2-Cycle Timing Guidelines

Table 4-4 describes PLB master signal timing guidelines.

Table 4-4. PLB Master 2-Cycle TIming Guidelines

Signal Name	Driven By	Output Valid	Received by
Mn_abort	PLB master n	Middle	PLB arbiter
Mn_ABus	PLB master n	Early	PLB arbiter
Mn_ABusPar	PLB master n	Early	PLB arbiter
Mn_ABusParEn	PLB master n	Early	PLB arbiter
Mn_BE	PLB master n	Early	PLB arbiter
Mn_BEPar	PLB master n	Early	PLB arbiter
Mn_BEParEn	PLB master n	Early	PLB arbiter
Mn_busLock	PLB master n	Early	PLB arbiter
Mn_lockErr	PLB master n	Early	PLB arbiter
Mn_MSize	PLB master n	Early	PLB arbiter
Mn_priority	PLB master n	Begin	PLB arbiter
Mn_rdDBusParErr	PLB master n	Early	rdDBusParErr OR
Mn_rdBurst	PLB master n	Early	PLB arbiter
Mn_request	PLB master n	Begin	PLB arbiter
Mn_RNW	PLB master n	Begin	PLB arbiter
Mn_size	PLB master n	Early	PLB arbiter
Mn_TAttribute	PLB master n	Early	PLB arbiter
Mn_type	PLB master n	Early	PLB arbiter
Mn_wrBurst	PLB master n	Early	PLB arbiter
Mn_wrDBus	PLB master n	Early	PLB arbiter
Mn_wrDBusPar	PLB master n	Early	PLB arbiter
Mn_wrDBusParEn	PLB master n	Early	PLB arbiter
Mn_UABus	PLB master n	Early	PLB arbiter
Mn_UABusPar	PLB master n	Early	PLB arbiter
Mn_UABusParEn	PLB master n	Early	PLB arbiter

4.2.3 PLB Arbiter 2-Cycle Timing Guidelines

Table 4-5 describes PLB arbiter signal timing guidelines.

Table 4-5. PLB Arbiter 2-Cycle Timing Guidelines (Sheet 1 of 3)

Signal Name	Driven By	Output Valid	Received by
PLB_abort	PLB arbiter	Late	Slaves
PLB_ABus	PLB arbiter	Begin(A2)	Slaves
PLB_ABusPar	PLB arbiter	Begin(A2)	Slaves
PLB_ABusParEn	PLB arbiter	Begin(A2)	Slaves
PLB_BE	PLB arbiter	Begin(A2)	Slaves

Table 4-5. PLB Arbiter 2-Cycle Timing Guidelines (Sheet 2 of 3)

Signal Name	Driven By	Output Valid	Received by
PLB_BEPar	PLB arbiter	Begin(A2)	Slaves
PLB_BEParEn	PLB arbiter	Begin(A2)	Slaves
PLB_busLock	PLB arbiter	Begin(A2)	Slaves
PLB_lockErr	PLB arbiter	Begin(A2)	Slaves
PLB_MasterID	PLB arbiter	Begin(A2)	Slaves
PLB_MnAddrAck	PLB arbiter	Late	PLB master n
PLB_MBusy(n)	OR Gate	Middle	PLB master n
PLB_MRdErr(n)	OR Gate	Middle	PLB master n
PLB_MWrErr(n)	OR Gate	Middle	PLB master n
PLB_MIRQ(n)	OR Gate	Middle	PLB master n
PLB_MnRdBTerm	PLB arbiter	Late	PLB master n
PLB_MnRdDAck	PLB arbiter	Middle	PLB master n
PLB_MnRdDBus	PLB arbiter	Middle	PLB master n
PLB_MnRdDBusPar	PLB arbiter	Middle	PLB master n
PLB_MnRdDBusParEn	PLB arbiter	Middle	PLB master n
PLB_MnRdWdAddr	PLB arbiter	Middle	PLB master n
PLB_MnRearbitrate	PLB arbiter	Late	PLB master n
PLB_MnTimeOut	PLB arbiter	Begin	PLB master n
PLB_MnSSize	PLB arbiter	Late	PLB master n
PLB_MnWrBTerm	PLB arbiter	Late	PLB Master n
PLB_MnWrDAck	PLB arbiter	Late	PLB master n
PLB_MSize	PLB arbiter	Begin(A2)	Slaves
PLB_PAValid	PLB arbiter	Begin	Slaves
PLB_rdPendPri	PLB arbiter	Middle	Masters/Slaves
PLB_rdPendReq	PLB arbiter	Middle	Masters/Slaves
PLB_wrPendPri	PLB arbiter	Middle	Masters/Slaves
PLB_wrPendReq	PLB arbiter	Middle	Masters/Slaves
PLB_rdBurst	PLB arbiter	Middle	Slaves
PLB_rdPrim	PLB arbiter	End	Slaves
PLB_reqPri	PLB arbiter	Begin(A2)	Slaves
PLB_RNW	PLB arbiter	Begin(A2)	Slaves
PLB_SAValid	PLB arbiter	Begin	Slaves
PLB_size	PLB arbiter	Begin(A2)	Slaves
PLB_TAttribute	PLB arbiter	Begin(A2)	Slaves
PLB_type	PLB arbiter	Begin(A2)	Slaves
PLB_wrBurst	PLB arbiter	Middle	Slaves
PLB_wrDBus	PLB arbiter	Middle	Slaves

Table 4-5. PLB Arbiter 2-Cycle Timing Guidelines (Sheet 3 of 3)

Signal Name	Driven By	Output Valid	Received by
PLB_wrDBusPar	PLB arbiter	Middle	Slaves
PLB_wrDBusParEn	PLB arbiter	Middle	Slaves
PLB_wrPrim	PLB arbiter	End	Slaves
PLB_UABus	PLB arbiter	Begin(A2)	Slaves
PLB_UABusPar	PLB arbiter	Begin(A2)	Slaves
PLB_UABusParEn	PLB arbiter	Begin(A2)	Slaves

4.2.4 PLB Slave 2-Cycle Timing Guidelines

Table 4-6 describes PLB slave signal timing guidelines.

Table 4-6. PLB Slave 2-Cycle Timing Guidelines

Signal Name	Driven By	Output Valid	Received by
Sl_addrAck	Slaves	Middle	PLB arbiter
Sl_ABusParErr	Slaves	Early	ABusParErr OR
Sl_MBusy(n)	Slaves	Early	MBusy(n) OR
Sl_MRdErr(n)	Slaves	Early	MRdErr(n) OR
Sl_MWrErr(n)	Slaves	Early	MWrErr(n) OR
Sl_MIRQ(n)	Slaves	Early	MIRQ(n) OR
Sl_rdBTerm	Slaves	Middle	PLB arbiter
Sl_rdComp	Slaves	Early	PLB arbiter
Sl_rdDAck	Slaves	Early	PLB arbiter
Sl_rdDBus	Slaves	Early	PLB arbiter
Sl_rdDBusPar	Slaves	Early	PLB arbiter
Sl_rdDBusParEn	Slaves	Early	PLB arbiter
Sl_rdwdAddr	Slaves	Early	PLB arbiter
Sl_rearbitrate	Slaves	Middle	PLB arbiter
Sl_SSize	Slaves	Middle	PLB arbiter
Sl_wait	Slaves	Middle	PLB arbiter
Sl_wrBTerm	Slaves	Middle	PLB arbiter
Sl_wrComp	Slaves	Middle	PLB arbiter
Sl_wrDAck	Slaves	Middle	PLB arbiter

4.3 3-Cycle Acknowledgment Timing Guidelines

The 3-cycle PLB signal timing guidelines described in this section are based on a single clock cycle arbitration, referred to as A1, followed by the first address valid cycle, referred to as A2, and completed with the assertion of address acknowledgment in the third clock cycle. These guidelines are provided in an attempt to

maximize bus frequency and promote the reusability of PLB masters and slaves at higher frequencies and various technologies. Perform timing analysis early on all Core+ASIC chips using the PLB bus core 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 core user manuals for details.

Begin(A2)	Signal is valid within 5% of the clock cycle from the rise of the Sys_plbClk signal.
Begin	Signal is valid within 15% of the clock cycle from the rise of the Sys_plbClk signal.
Early	Signal is valid within 30% of the clock cycle from the rise of the Sys_plbClk signal.
Middle	Signal is valid within 50% of the clock cycle from the rise of the Sys_plbClk signal
Late	Signal is valid within 65% of the clock cycle from the rise of the Sys_plbClk signal.
End	Signal is valid within 75% of the clock cycle from the rise of the Sys_plbClk signal.

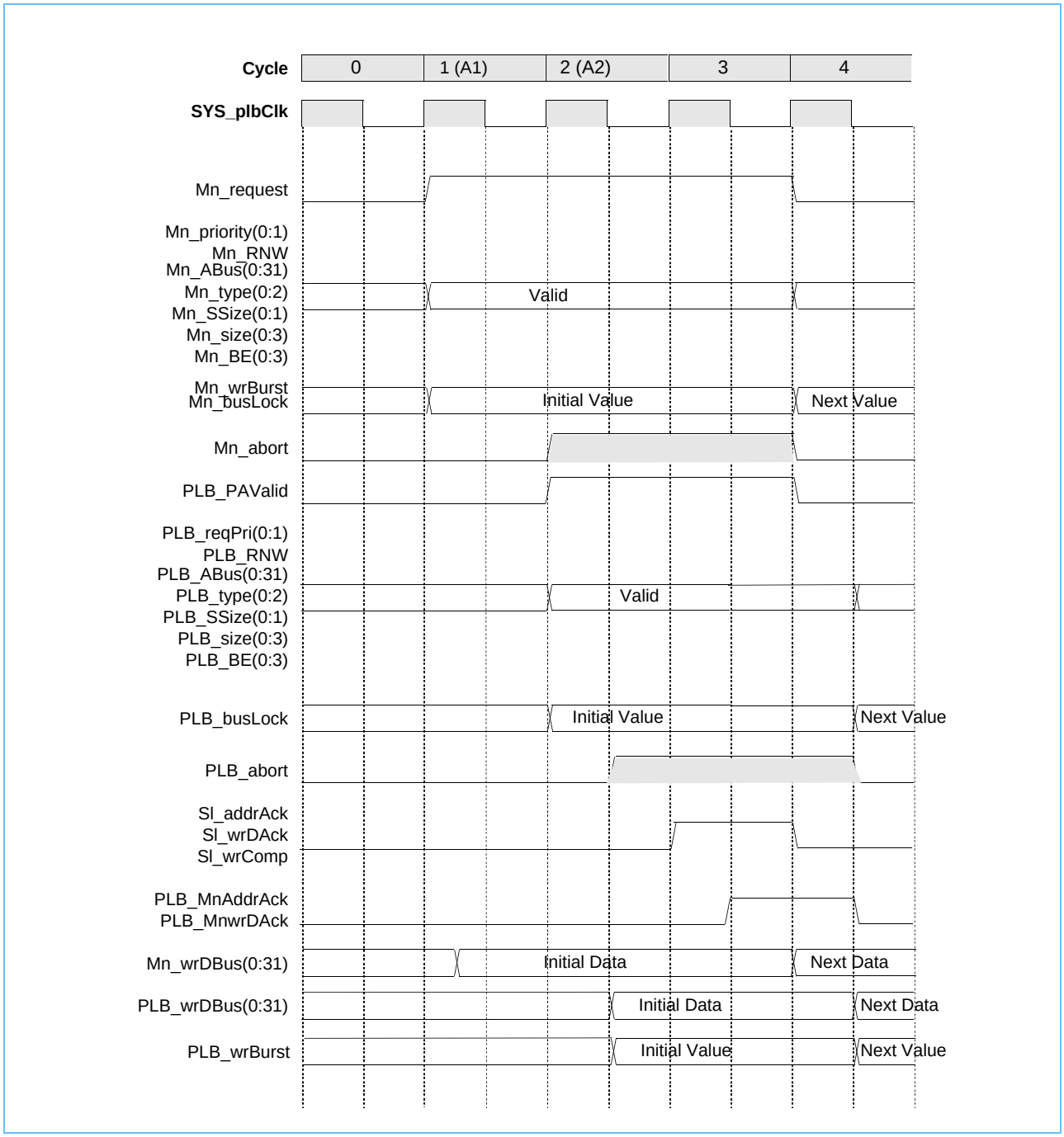
Notes:

1. These definitions assume that there are 0 ns 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 core. For inputs, these delays represent the arrival time of the input relative to a 0 ns delayed clock.
2. Signals that are identified as Begin(A2) can be considered during A2 and subsequent cycles before and including the cycle in which the slave acknowledges the master request. In the cycle following the assertion of the Sl_addrAck signal, all signals that are specified as Begin(A2), except the PLB_busLock signal, are no longer sampled by the slave. The PLB_busLock signal is then considered to be a middle signal until the completion of all data transfers that are associated with the initial master request.

4.3.1 Generic 3-Cycle Acknowledgment Arbitration

Figure 4-2 shows the operation of a generic 3-cycle PLB acknowledgment. The master asserts its request at the start of A1. The PLB then performs arbitration, and if the Mn_abort signal is deasserted at the end of A1, the PLB asserts the PLB_PAValid signal at the start of A2. All transfer qualifiers are valid at the start of A2. The slave decodes the access and asserts the addrAck signal at the start of the next clock.

Figure 4-2. Generic 3-Cycle PLB Arbitration



4.3.2 PLB Master 3-Cycle Timing Guidelines

Table 4-7 describes PLB master signal timing guidelines.

Table 4-7. PLB Master 3-Cycle TIming Guidelines

PLB Signal Name	Driven By	Output Valid	Received
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	Begin	PLB arbiter
Mn_BE	PLB master n	Begin	PLB arbiter
Mn_BEPar	PLB master n	Begin	PLB arbiter
Mn_BEParEn	PLB master n	Begin	PLB arbiter
Mn_size	PLB master n	Begin	PLB arbiter
Mn_TAttribute	PLB master n	Begin	PLB arbiter
Mn_type	PLB master n	Begin	PLB arbiter
Mn_MSize	PLB master n	Begin	PLB arbiter
Mn_lockErr	PLB master n	Begin	PLB arbiter
Mn_ABus	PLB master n	Begin	PLB arbiter
Mn_ABusPar	PLB master n	Begin	PLB arbiter
Mn_ABusParEn	PLB master n	Begin	PLB arbiter
Mn_UABus	PLB master n	Begin	PLB arbiter
Mn_UABusPar	PLB master n	Begin	PLB arbiter
Mn_UABusParEn	PLB master n	Begin	PLB arbiter
Mn_wrDBus	PLB master n	Begin	PLB arbiter
Mn_wrDBusPar	PLB master n	Begin	PLB arbiter
Mn_wrDBusParEn	PLB master n	Begin	PLB arbiter
Mn_wrBurst	PLB master n	Begin	PLB arbiter
Mn_rdBurst	PLB master n	Begin	PLB arbiter
Mn_rdDBusParErr	PLB master n	Begin	PLB arbiter
Mn_abort	PLB master n	Begin	PLB arbiter

4.3.3 PLB Arbiter 3-Cycle Timing Guidelines

Table 4-8 describes PLB arbiter signal timing guidelines.

Table 4-8. PLB Arbiter 3-Cycle Timing Guidelines (Sheet 1 of 3)

PLB Signal Name	Driven By	Output Valid	Received
PLB_MBusy(n)	OR Gate	Early	PLB master n
PLB_MRdErr(n)	OR Gate	Early	PLB master n
PLB_MWrErr(n)	OR Gate	Early	PLB master n

Table 4-8. PLB Arbiter 3-Cycle Timing Guidelines (Sheet 2 of 3)

PLB Signal Name	Driven By	Output Valid	Received
PLB_MIRQ(n)	OR Gate	Early	PLB master n
PLB_MnRdDBus	PLB arbiter	Middle	PLB master n
PLB_MnRdDBusPar	PLB arbiter	Middle	PLB master n
PLB_MnRdDBusParEn	PLB arbiter	Middle	PLB master n
PLB_MnRdWdAddr	PLB arbiter	Middle	PLB master n
PLB_MnRdDAck	PLB arbiter	Middle	PLB master n
PLB_rdBurst	PLB arbiter	Middle	Slaves
PLB_rdPendReq	PLB arbiter	Middle	Masters/Slaves
PLB_rdPendPri	PLB arbiter	middle	Masters/Slaves
PLB_wrPendReq	PLB arbiter	Middle	Masters/Slaves
PLB_wrPendPri	PLB arbiter	middle	Masters/Slaves
PLB_busLock	PLB arbiter	Begin(A2)	Slaves
PLB_reqPri	PLB arbiter	Begin(A2)	Slaves
PLB_MasterID	PLB arbiter	Begin(A2)	Slaves
PLB_PAValid	PLB arbiter	Begin	Slaves
PLB_SAValid	PLB arbiter	Begin	Slaves
PLB_rdPrim	PLB arbiter	Middle	Slaves
PLB_wrPrim	PLB arbiter	Middle	Slaves
PLB_RNW	PLB arbiter	Begin(A2)	Slaves
PLB_BE	PLB arbiter	Begin(A2)	Slaves
PLB_BEPar	PLB arbiter	Begin(A2)	Slaves
PLB_BEParEn	PLB arbiter	Begin(A2)	Slaves
PLB_size	PLB arbiter	Begin(A2)	Slaves
PLB_TAttribute	PLB arbiter	Begin(A2)	Slaves
PLB_type	PLB arbiter	Begin(A2)	Slaves
PLB_MSize	PLB arbiter	Begin(A2)	Slaves
PLB_lockErr	PLB arbiter	Begin(A2)	Slaves
PLB_ABus	PLB arbiter	Begin(A2)	Slaves
PLB_ABusPar	PLB arbiter	Begin(A2)	Slaves
PLB_ABusParEn	PLB arbiter	Begin(A2)	Slaves
PLB_UABus	PLB arbiter	Begin(A2)	Slaves
PLB_UABusPar	PLB arbiter	Begin(A2)	Slaves
PLB_UABusParEn	PLB arbiter	Begin(A2)	Slaves
PLB_wrDBus	PLB arbiter	Middle	Slaves
PLB_wrDBusPar	PLB arbiter	Middle	Slaves
PLB_wrDBusParEn	PLB arbiter	Middle	Slaves
PLB_wrBurst	PLB arbiter	Middle	Slaves

Table 4-8. PLB Arbiter 3-Cycle Timing Guidelines (Sheet 3 of 3)

PLB Signal Name	Driven By	Output Valid	Received
PLB_MnSSize	PLB arbiter	Middle	PLB master n
PLB_MnWrBTerm	PLB arbiter	Middle	PLB Master n
PLB_MnRdBTerm	PLB arbiter	Middle	PLB master n
PLB_MnAddrAck	PLB arbiter	Middle	PLB master n
PLB_MnTimeOut	PLB arbiter	Begin	PLB master n
PLB_MnRearbitrate	PLB arbiter	Middle	PLB master n
PLB_MnWrDAck	PLB arbiter	Middle	PLB master n
PLB_abort	PLB arbiter	Middle	Slaves

4.3.4 PLB Arbiter 3-Cycle Timing Guidelines

Table 4-9 describes PLB arbiter input signal timing guidelines.

Table 4-9. PLB Arbiter 3-Cycle Timing Guidelines

PLB Signal Name	Driven By	Output Valid	Received
Sl_rdwdAddr	OR logic	Early	PLB arbiter
Sl_rdDAck	OR logic	Early	PLB arbiter
Sl_rdComp	OR logic	Early	PLB arbiter
Sl_rdDBus	OR logic	Early	PLB arbiter
Sl_rdDBusPar	OR logic	Early	PLB arbiter
Sl_rdDBusParEn	OR logic	Early	PLB arbiter
Sl_rdBTerm	OR logic	Early	PLB arbiter
Sl_wrBTerm	OR logic	Early	PLB arbiter
Sl_wrDAck	OR logic	Early	PLB arbiter
Sl_wrComp	OR logic	Early	PLB arbiter
Sl_addrAck	OR logic	Early	PLB arbiter
Sl_wait	OR logic	Early	PLB arbiter
Sl_SSize	OR logic	Early	PLB arbiter
Sl_rearbitrate	OR logic	Early	PLB arbiter

4.3.5 PLB Slave 3-Cycle Timing Guidelines

Table 4-10 describes PLB slave signal timing guidelines.

Table 4-10. PLB Slave 3-Cycle Timing Guidelines (Sheet 1 of 2)

PLB Signal Name	Driven By	Output Valid	Received
Sln_MBusy(n)	Slaves	Begin	MBusy(n) OR
Sln_MRdErr(n)	Slaves	Begin	MRdErr(n) OR
Sln_MWrErr(n)	Slaves	Begin	MWrErr(n) OR

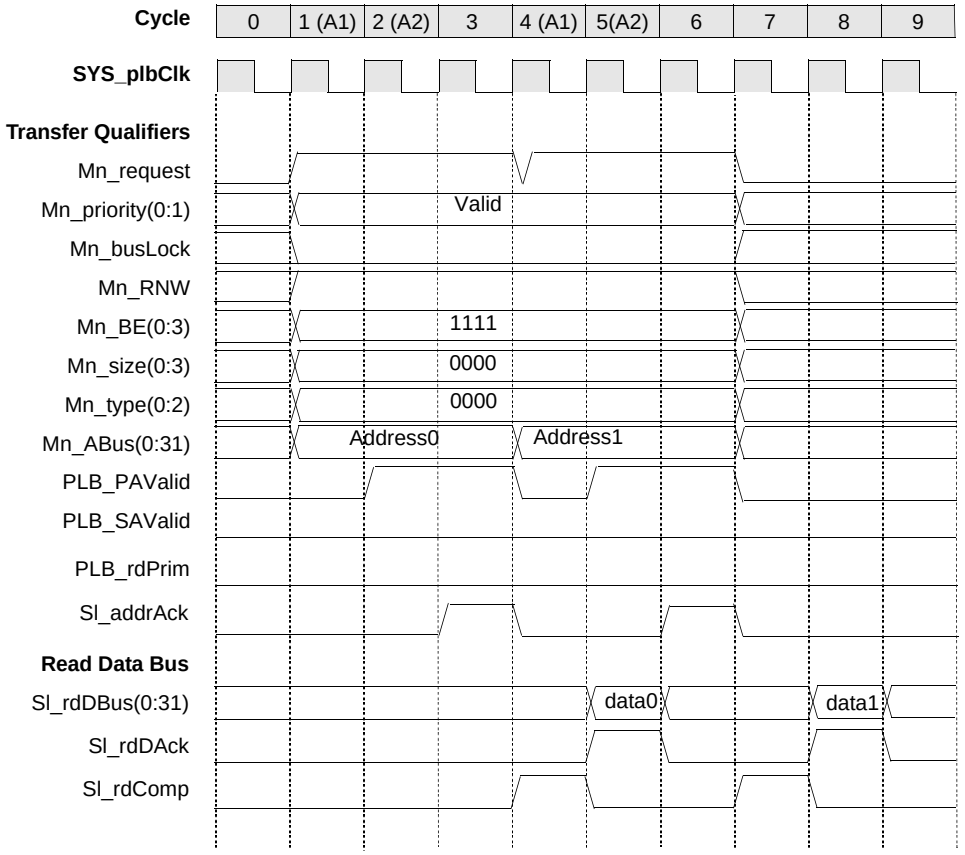
Table 4-10. PLB Slave 3-Cycle Timing Guidelines (Sheet 2 of 2)

PLB Signal Name	Driven By	Output Valid	Received
Sln_MIRQ(n)	Slaves	Begin	MIRQ(n) OR
Sln_rdwdAddr	Slaves	Begin	OR logic
Sln_rdDAck	Slaves	Begin	OR logic
Sln_rdComp	Slaves	Begin	OR logic
Sln_rdDBus	Slaves	Begin	OR logic
Sln_rdDBusPar	Slaves	Begin	OR logic
Sln_rdDBusParEn	Slaves	Begin	OR logic
Sln_rdBTerm	Slaves	Begin	OR logic
Sln_wrBTerm	Slaves	Begin	OR logic
Sln_wrDAck	Slaves	Begin	OR logic
Sln_wrComp	Slaves	Begin	OR logic
Sl_addrAck	Slaves	Begin	OR logic
Sl_ABusParErr	Slaves	Begin	OR logic
Sl_wait	Slaves	Begin	OR logic
Sl_SSize	Slaves	Begin	OR logic
Sl_rearbitrate	Slaves	Begin	OR logic

4.3.6 Back-to-Back Read Operation with 3-Cycle Acknowledgment

Figure 4-3 illustrates the operation of back-to-back PLB read operations with 3-cycle PLB arbitration. Arbitration for the initial request occurs in clock cycle 1. The PLB_PAValid signal for Address 0 is asserted in clock cycle 2. The slave acknowledges this request, and the master drives a subsequent read request for address1 in clock cycle 3. Arbitration for the second request occurs and the PLB_SAValid signal is asserted in clock 4. The Sl_rdComp signal is asserted, the PLB_SAValid signal is deasserted, and the PLB_PAValid signal is asserted in clock cycle 5. The slave provides data 0 with the Sl_rdDAck signal and acknowledges the second request. The second read cycle completes normally.

Figure 4-3. Back-to-Back Read Operation with 3-Cycle Acknowledgment



5. PLB Operations

This section on processor local bus (PLB) operations describes in detail the following topics with appropriate timing diagrams:

- PLB nonaddress pipelining
- 2 deep PLB address pipelining
- PLB bandwidth and latency

All signals on the PLB are positive active and are either direct outputs of edge-triggered latches that 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 must be captured in the masters or slaves on the rising edge of the SYS_plbClk signal.

5.1 PLB Nonaddress Pipelining

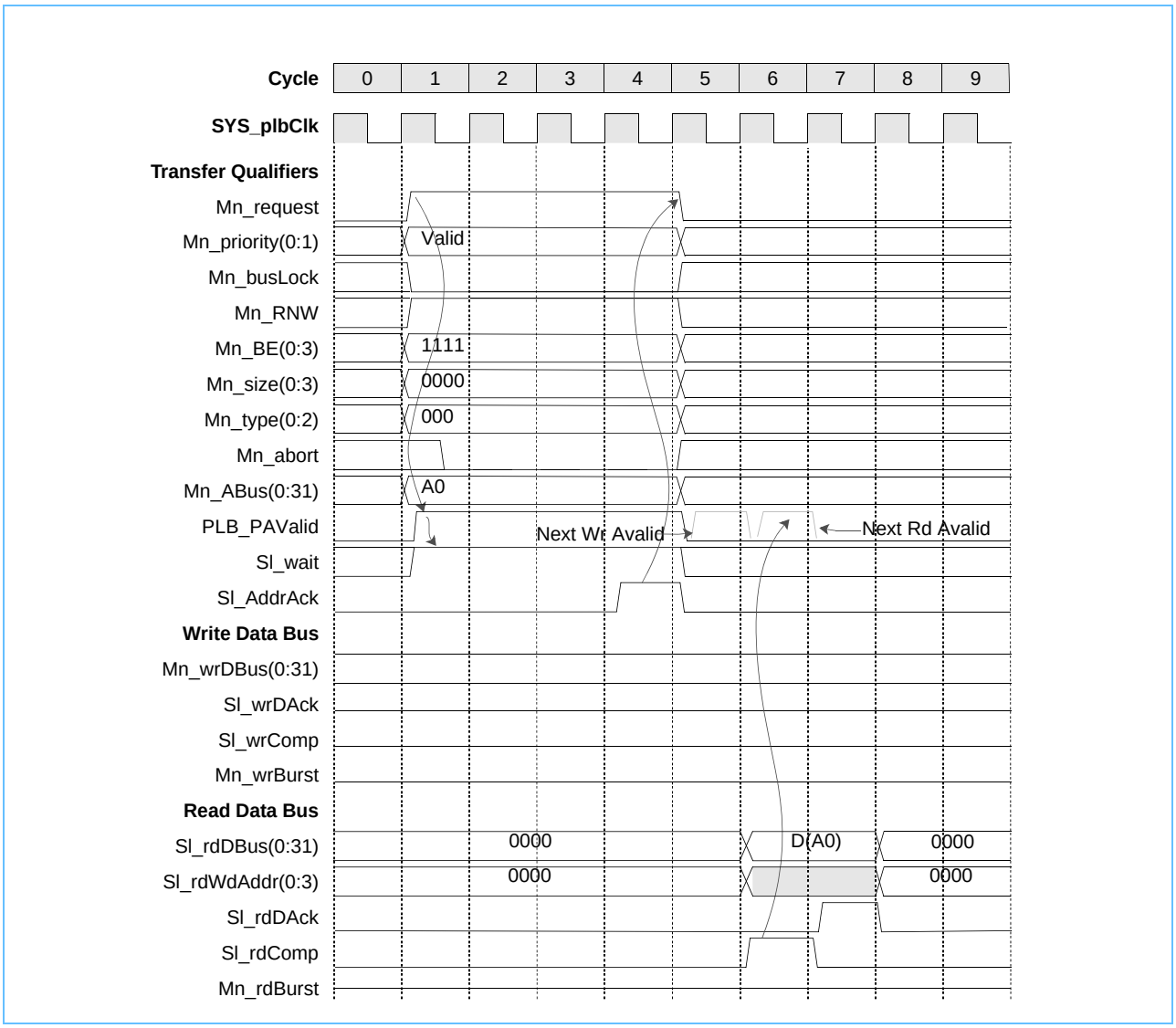
The timing diagrams included in this section are examples of nonaddress pipelined read and write transfers on the PLB. However, the signal assertion and negation times as shown in these diagrams are only meant to illustrate their dependency on the rising edge of SYS_plbClk, and in no way are they intended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

Furthermore, because set-up and hold times for the PLB inputs are dependent on the technology that is used and on the physical implementation of the bus, these parameters are specified as a percentage of the bus clock cycle relative to the rise of the SYS_plbClk signal. A set of signal timing guidelines to be used in the design of PLB masters and slaves has been developed and described in *Section 4 PLB Timing Guidelines* on page 67.

5.1.1 Read Transfers

Figure 5-1 shows the operation of a single-read data-transfer on the PLB. The slave asserts the Sl_wait signal 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 continues to drive the PLB_PAValid signal, the address signal, and transfer qualifier signal until the slave device asserts the Sl_addrAck signal. The slave then asserts the Sl_rdComp signal in the clock cycle preceding the data acknowledgment phase to indicate that the transfer will complete in the following clock cycle and that the arbiter can arbitrate the next read request.

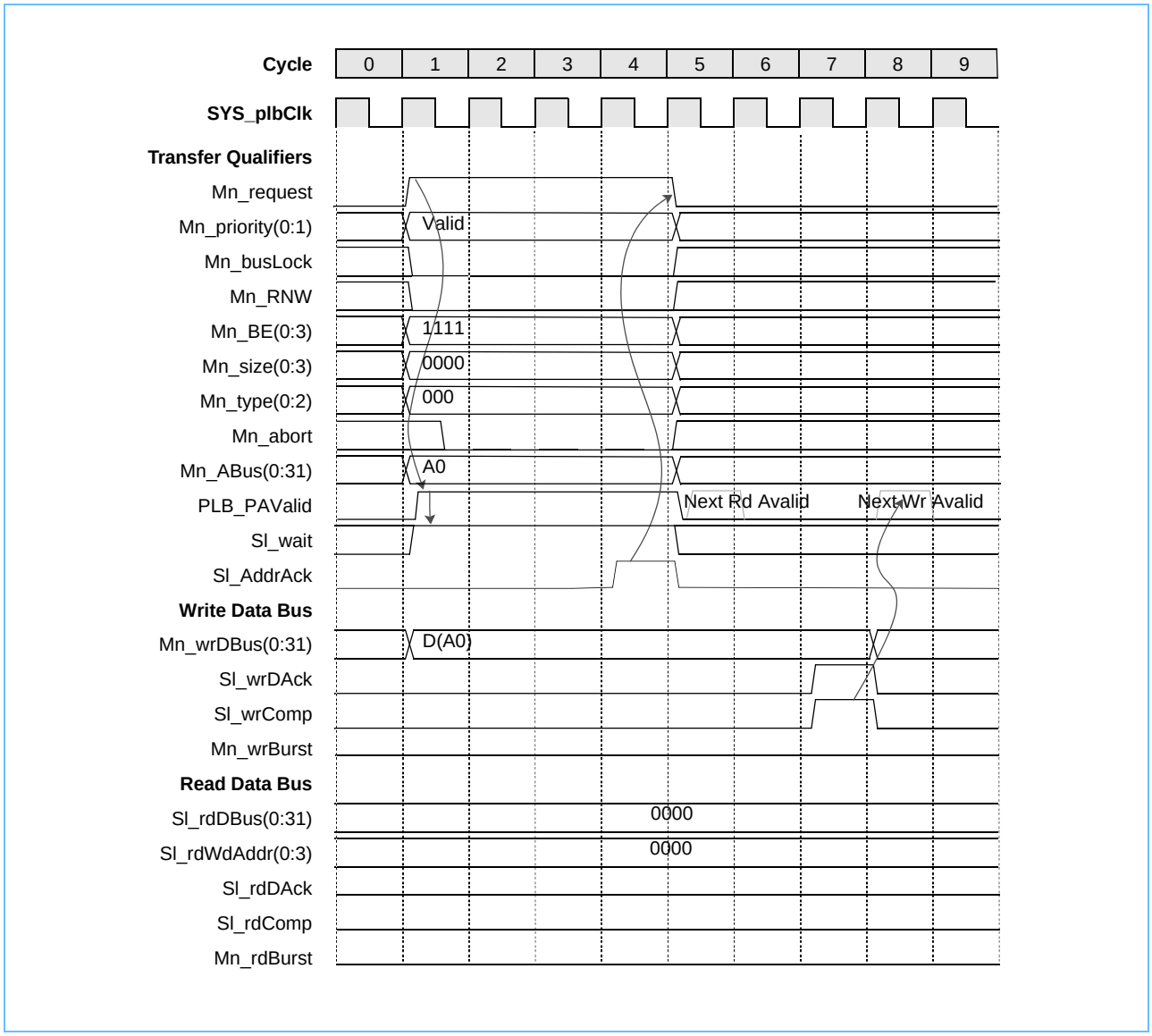
Figure 5-1. Read Transfers



5.1.2 Write Transfers

Figure 5-2 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 continues to drive the PLB_PAValid signal, the address signal, and transfer qualifier signal 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. 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 the Sl_wrDAck signal is asserted.

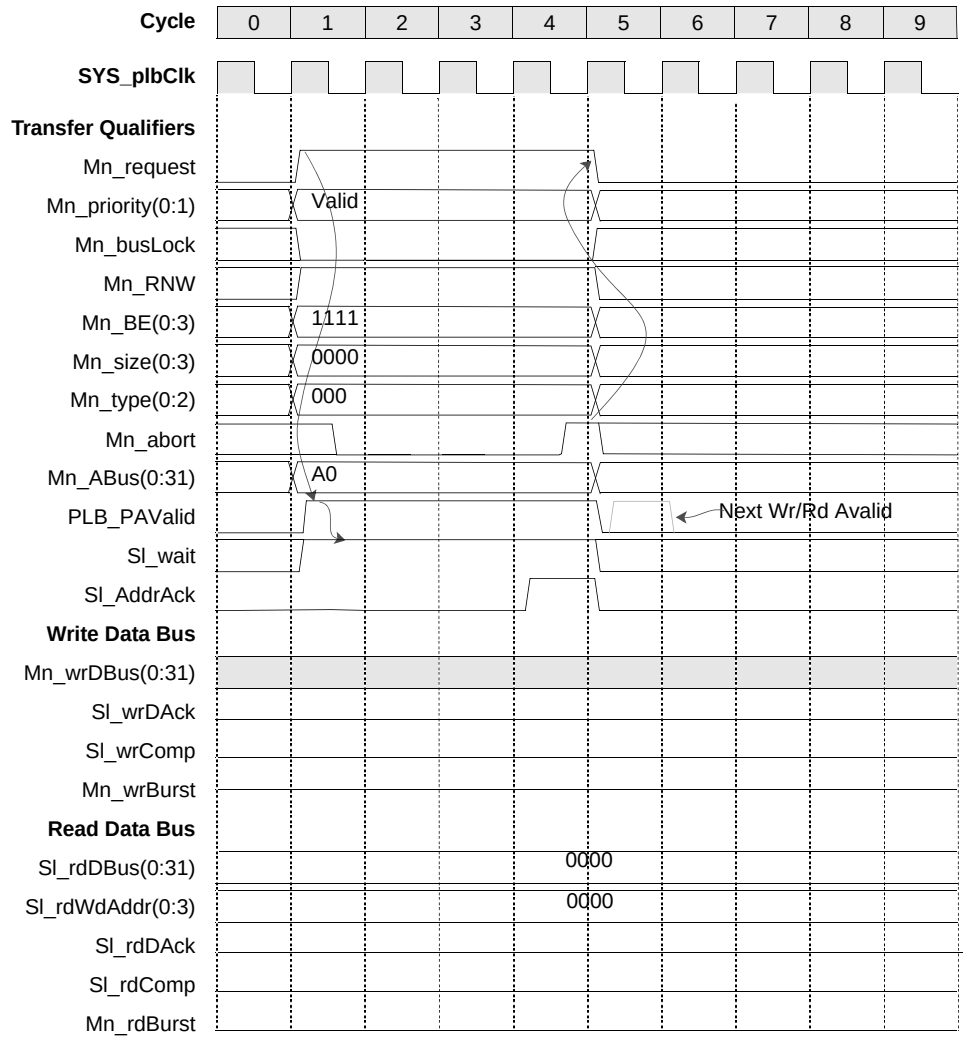
Figure 5-2. Write Transfers



5.1.3 Transfer Abort

Figure 5-3 shows a transfer that the master aborted in the same clock cycle in which the request was being acknowledged by the slave. When the master asserts the Mn_abort signal in clock cycle 4, the PLB arbiter and PLB slaves ignore the address acknowledgment and abort the requested transfer. All active requests are then sampled in the next clock cycle when the PLB arbiter rearbitrates. The Mn_abort signal has a minimal amount of set-up time to allow this signal to be asserted late in a clock cycle. The data handshaking is not completed by the assertion of the data acknowledgment signals. The master can either negate its request signal or make a new request in the clock cycle following the aborted request.

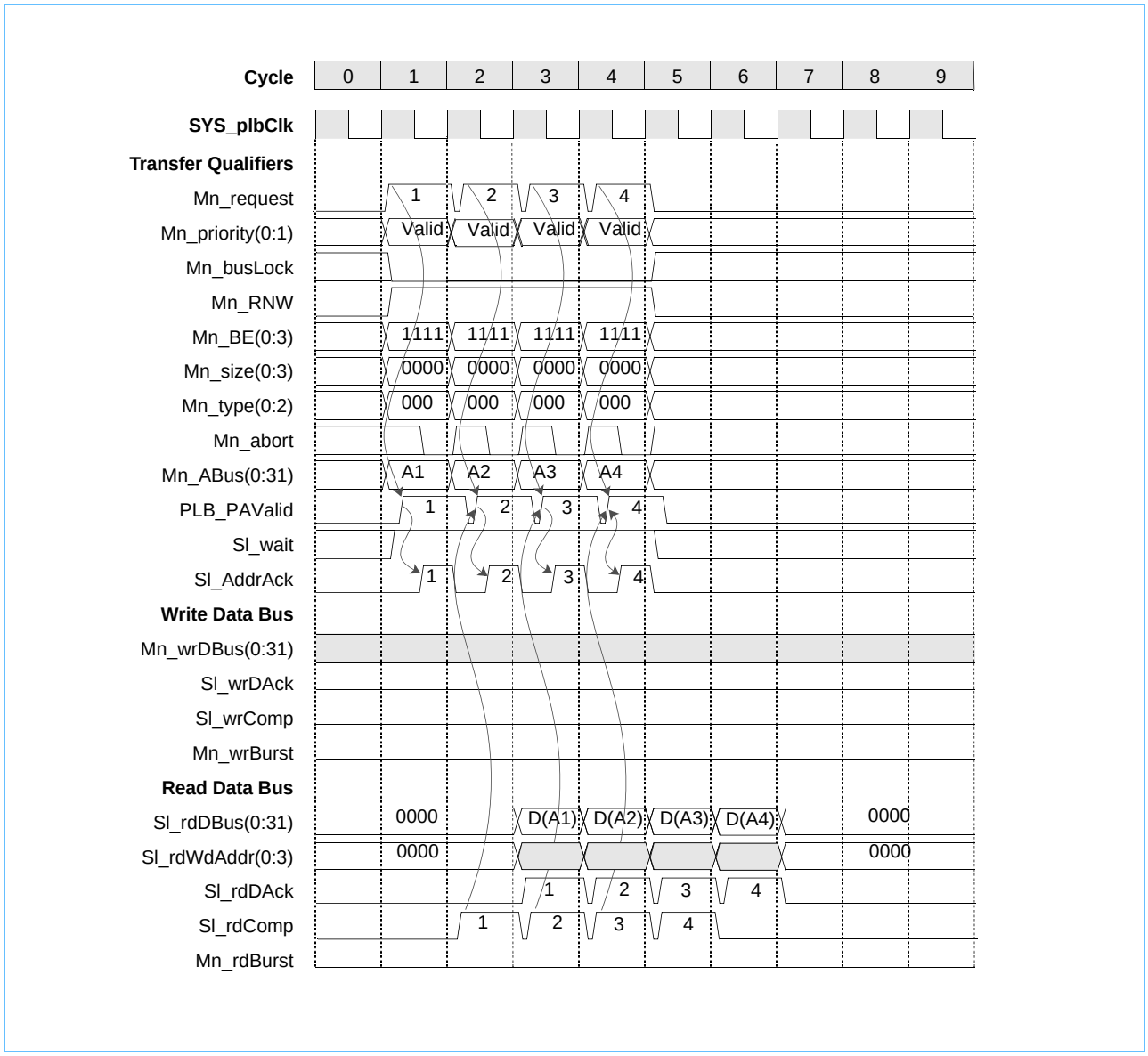
Figure 5-3. Transfer Abort



5.1.4 Back-to-Back Read Transfers

Figure 5-4 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 signal. This assertion allows the read request of the next master to be sent to slaves in the clock cycle preceding the data acknowledgment phase on the PLB. The slave cannot assert its Sl_rdDAck signal for the data read until two clock cycles following the assertion of the corresponding Sl_addrAck signal. 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 can read data every clock cycle from a slave that is capable of providing data in a single clock cycle.

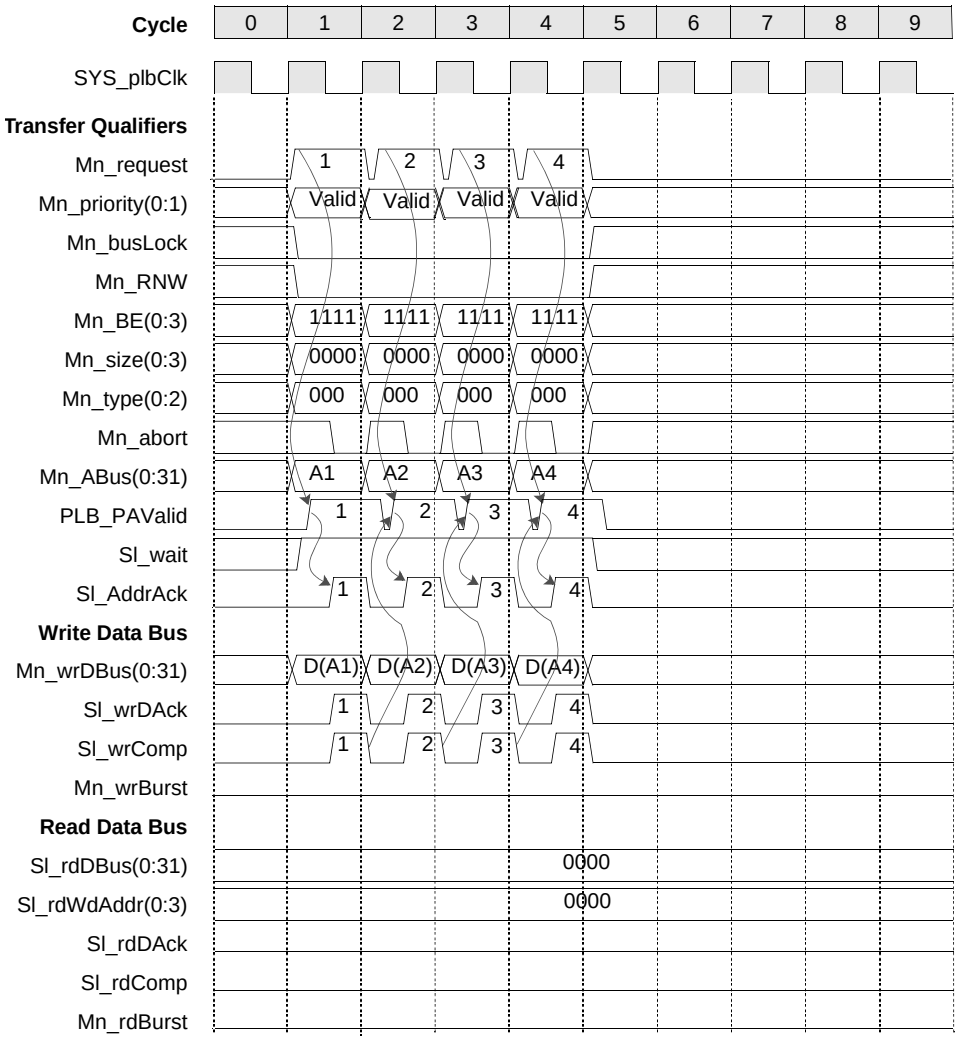
Figure 5-4. Back-to-Back Read Transfers



5.1.5 Back-to-Back Write Transfers

Figure 5-5 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. This completes the transfer within a single clock cycle on the PLB. The next valid write address cycle occurs in the clock cycle following the assertion of the Sl_wrComp signal.

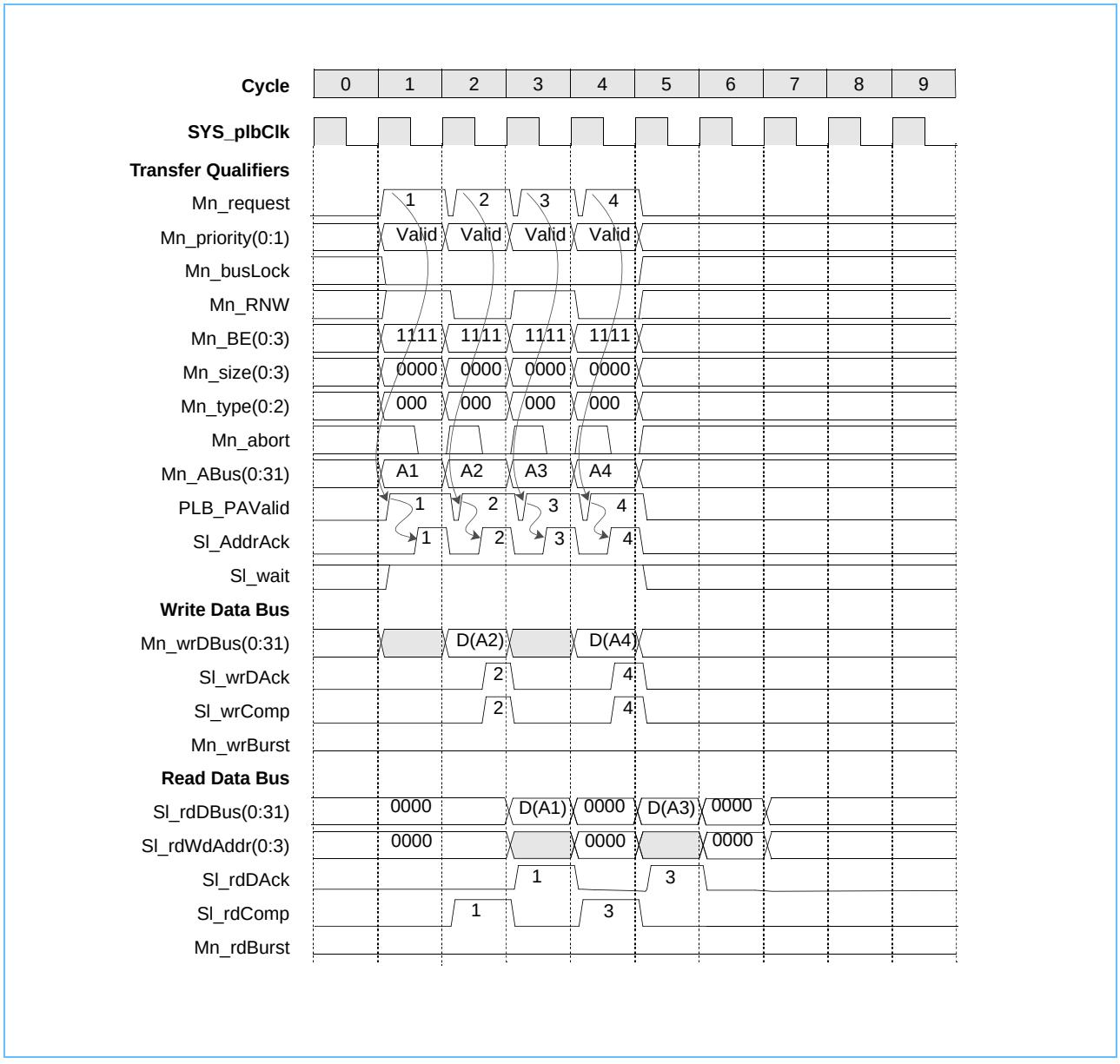
Figure 5-5. Back-to-Back Write Transfers



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

Figure 5-6 shows the operation of several back-to-back single read and write transfers on the PLB. Although the PLB arbiter grants the requests in the order that they are presented, the data transfer for the write transfers occurs in the clock cycle before the data transfers for the read transfers. Using this protocol, a slave can be continuously read or written at a rate of one transfer per clock cycle.

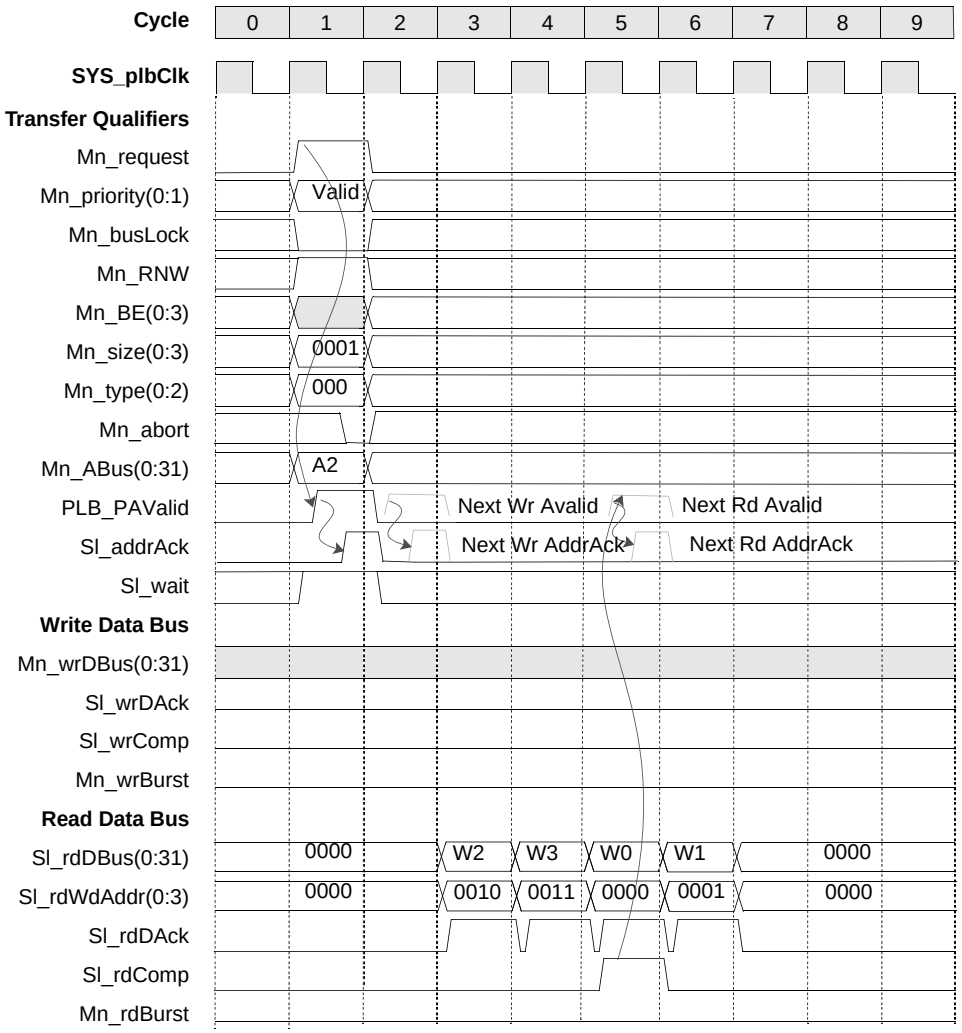
Figure 5-6. Back-to-Back Read/Write - Read/Write Transfers



5.1.7 4-word Line Read Transfers

Figure 5-7 shows the operation of a single 4-word line-read from a slave device that can provide data in a single clock cycle. For line transfers, the words within the line can be transferred in any order. The Sl_rdWdAddr(0:3) outputs of the slave 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 that indicates to the PLB arbiter that the line transfer will be completed in the following clock cycle.

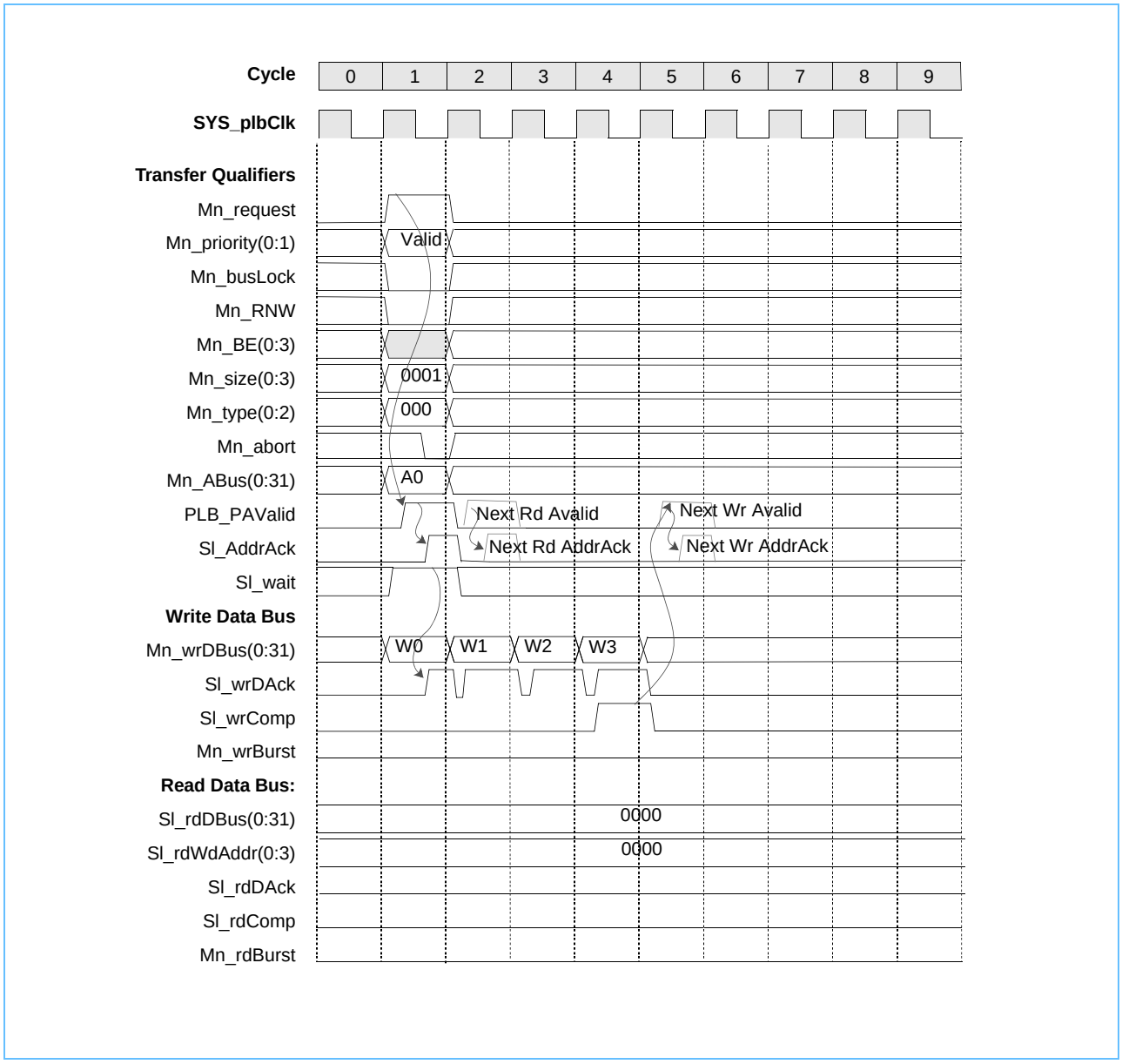
Figure 5-7. 4-Word Line Read Transfers



5.1.8 4-Word Line Write Transfers

Figure 5-8 shows the operation of a single 4-word line write to a slave device that can latch 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.

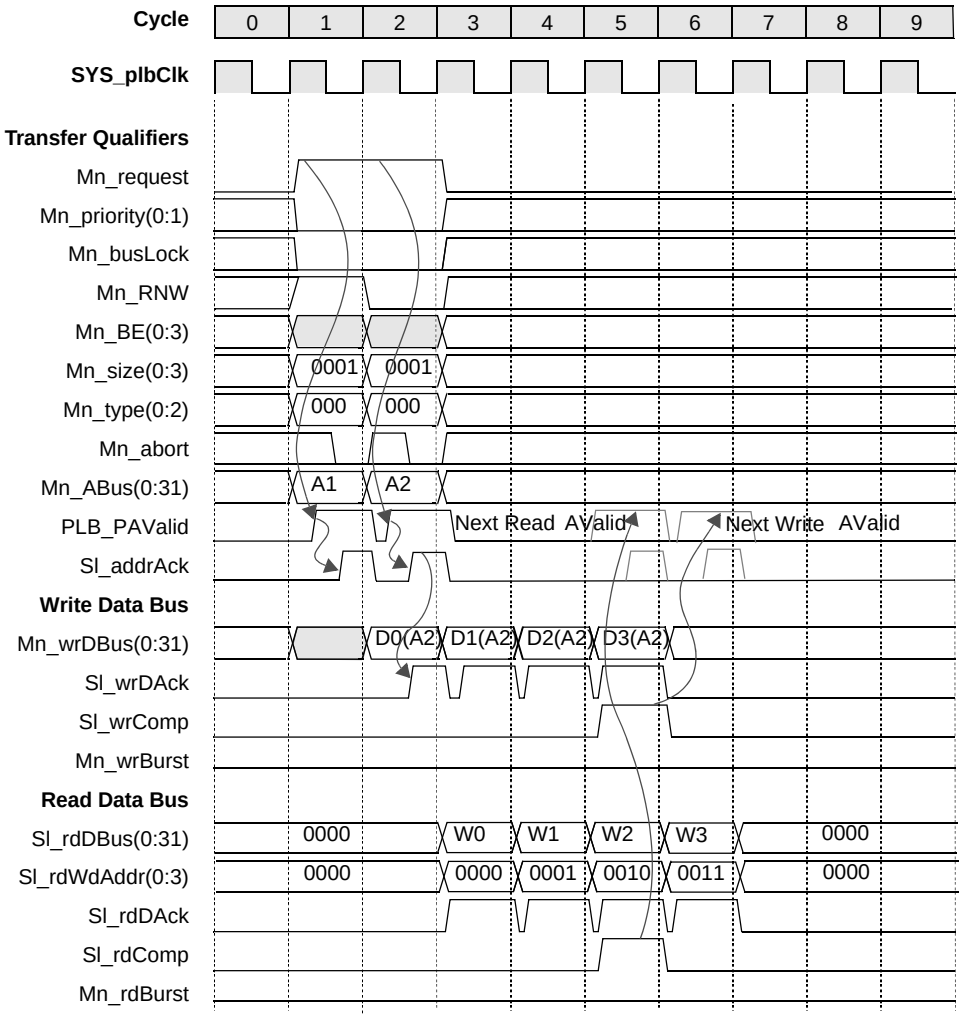
Figure 5-8. 4-Word Line Write Transfers



5.1.9 4-Word Line Read Followed by 4-Word Line Write Transfers

Figure 5-9 shows the operation of a 4-word line read followed immediately by a 4-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.

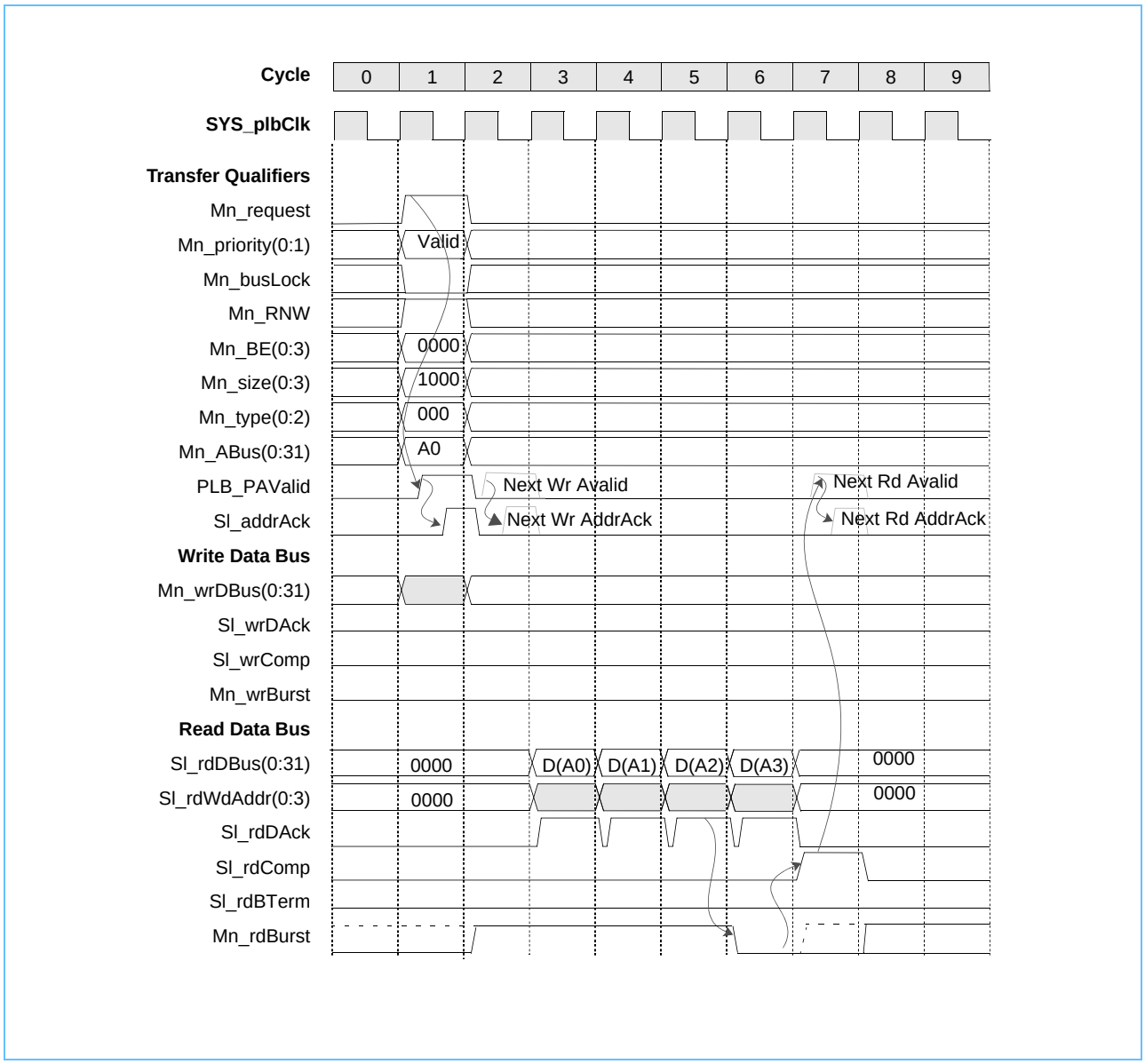
Figure 5-9. 4-word Line Read Followed by 4-Word Line Write Transfers



5.1.10 Sequential Burst Read Transfer Terminated by Master

Figure 5-10 shows the operation of a burst read from a slave device on the PLB. A master can 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 internally increments the address sequentially for each data transfer and continues to fetch data until it detects a low value on the Mn_rdBurst signal. The burst transfer is then completed by the slave device that is asserting the Sl_rdComp in the data acknowledgment phase of the last data transfer cycle following the negation of the Mn_rdBurst signal.

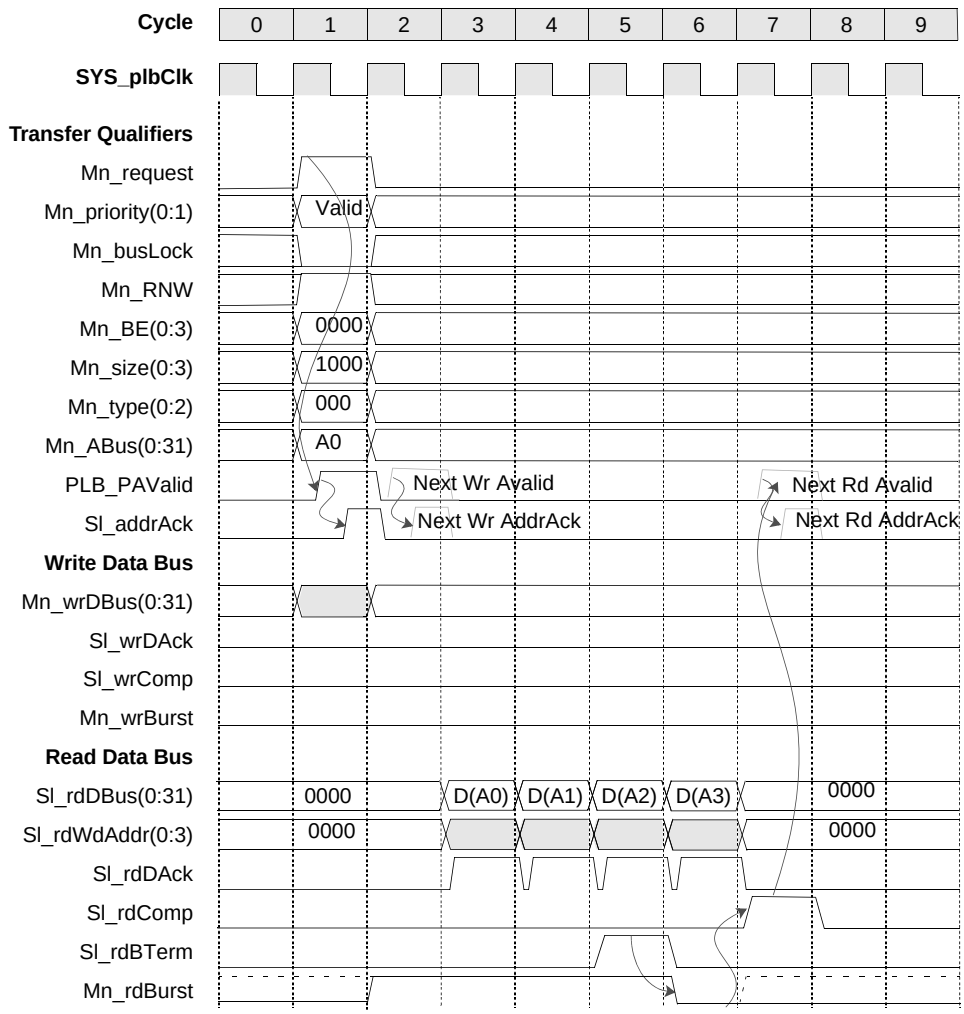
Figure 5-10. Sequential Burst Read Transfer Terminated by Master



5.1.11 Sequential Burst Read Transfer Terminated by Slave

Figure 5-11 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 5-12 Burst Write Transfer Terminated by Master* on page 95 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 acknowledgment phase of the last data transfer cycle.

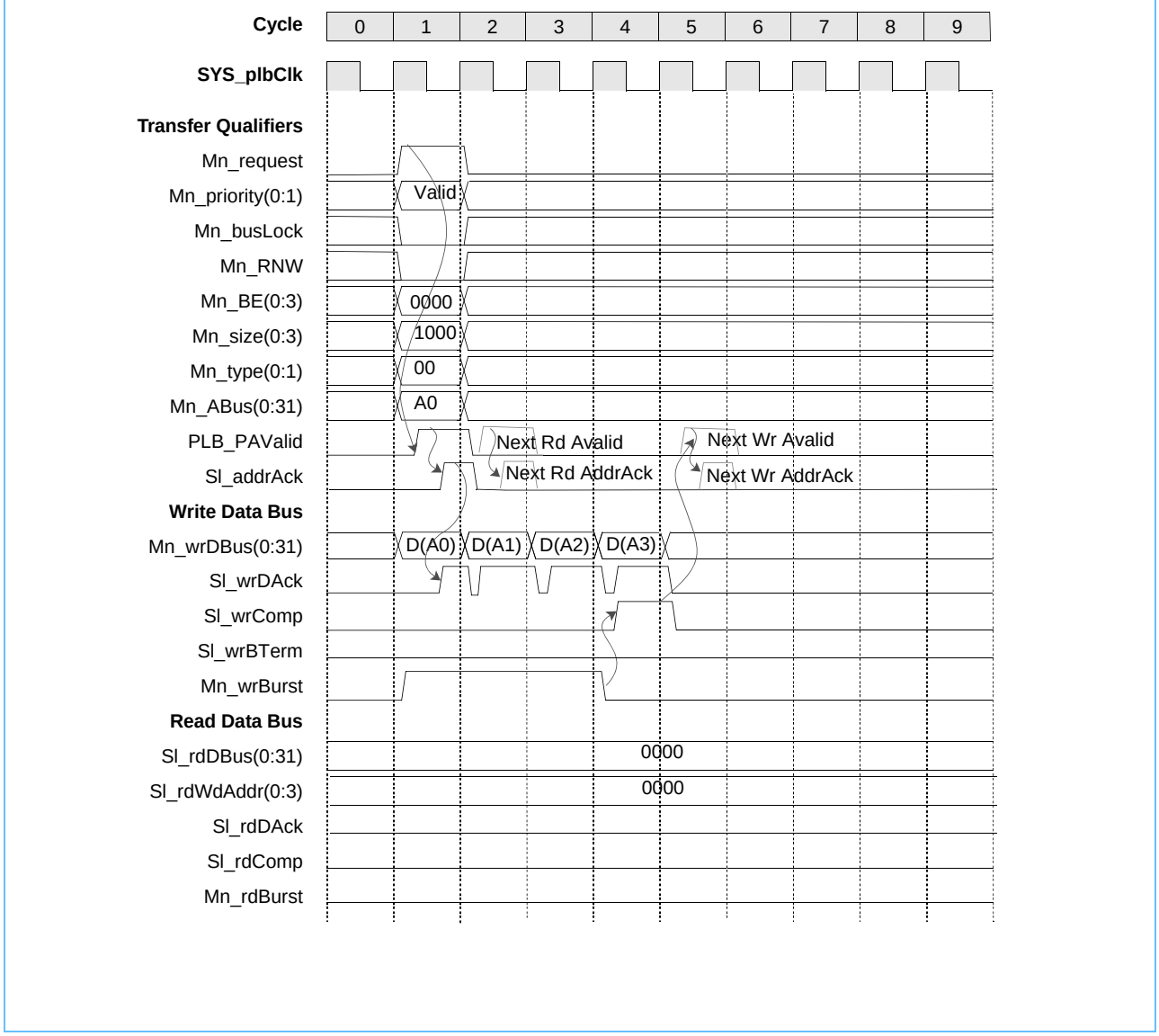
Figure 5-11. Burst Read Transfer Terminated by Slave



5.1.12 Sequential Burst Write Transfer Terminated by Master

Figure 5-12 shows the operation of a burst write to a slave device on the PLB. A master can 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 internally increments the address sequentially for each data transfer. When the slave detects a low value on the Mn_wrBurst signal, the slave asserts the Sl_wrComp signal during the data acknowledgment phase for the next (and last) data transfer cycle to indicate to the PLB arbiter that the burst transfer is complete.

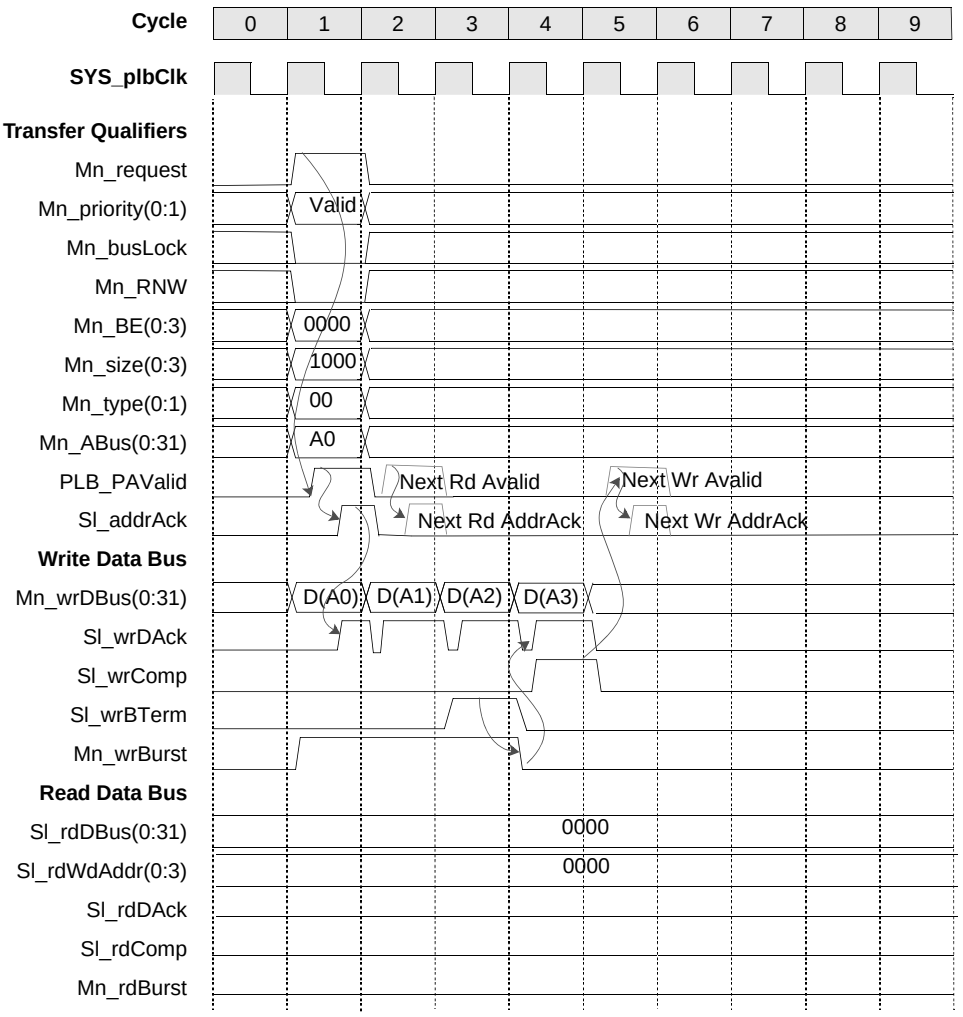
Figure 5-12. Burst Write Transfer Terminated by Master



5.1.13 Sequential Burst Write Transfer Terminated by Slave

Figure 5-13 shows the operation of another burst write to a slave device on the PLB. This burst write transfer differs from the burst write transfer that is illustrated in *Figure 5-12 Burst Write Transfer Terminated by Master* on page 95 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 acknowledgment phase for the next (and last) data transfer cycle.

Figure 5-13. Burst Write Transfer Terminated by Slave



5.1.14 Fixed-Length Burst Transfer

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 used. See *Figure 5-10 Sequential Burst Read Transfer Terminated by Master* on page 93 and *Figure 5-11 Burst Read Transfer Terminated by Slave* on page 94 for more information.

In the case of a long burst, two cycles might 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.

To address this performance concern, a fixed-length transfer protocol is provided that can be optionally implemented in both masters and slaves. This transfer is compatible with the variable-length burst protocol such that the burst transfers occur using the normal transfer protocol for those masters and slaves that do not implement the fixed-length transfer protocol.

During the request phase of a burst transfer, a master can indicate the number of byte, halfword, word, doubleword, quadword, or octalword transfers by providing the length of the burst on the Mn_BE signals. For 32-bit masters bursts of 2 - 16 transfers are possible. These encodings are shown in *Table 5-1*. For 64-bit and larger masters bursts of 2 - 256 transfers are possible. These encodings are shown in *Table 5-2*. For compatibility with 32-bit masters, the burst count is formed by concatenating Mn_BE(4:7) and Mn_BE(0:3) for 64-bit and larger masters.

Table 5-1. Fixed-Length Burst Transfer for 32-Bit Masters

Mn_BE(0:3)	Burst Length
0000	Burst length determined by PLB_rdBurst or PLB_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

The burst length refers to the number of transfers of the data type selected by the Mn_size signals. Mn_size ‘1000’ and Mn_BE ‘1111’ transfer 16 bytes; Mn_size ‘1001’ and Mn_BE ‘1111’ transfer 16 halfwords; Mn_BE ‘1111’ and the Mn_size ‘1010’ transfer 16 words.

Table 5-2. Byte Enable Signals during Burst Transfers for 64-Bit and Larger Masters

Mn_BE(4:7)_Mn_BE(0:3)	Burst Length
0000_0000	Burst length determined by PLB_rdBurst or PLB_wrBurst signal.
0000_0001	Burst of 2
0000_0010	Burst of 3
.	.
.	.
.	.
0000_1111	Burst of 16
0001_0000	Burst of 17
0001_0001	Burst of 18
0001_0010	Burst of 19
.	.
.	.
.	.
1111_1110	Burst of 255
1111_1111	Burst of 256

For 64-bit and larger masters, the burst length is formed by concatenating Mn_BE(4:7) with MN_BE(0:3). This 8-bit value provides for transfer lengths of 2 256 <<bits?>>. The burst length refers to the number of transfers of the data type that is selected by the Mn_size signals. Mn_size ‘1011’ and Mn_BE(0:7) ‘1111 0001’ transfer 32 doublewords; Mn_size ‘1100’ and Mn_BE(0:7) ‘1111 0001’ transfer 32 quadwords; and Mn_BE(0:7) ‘1111 0001’ and Mn_size ‘1101’ transfer 32 octalwords. Slaves must use the Mn_BE and Mn_size signals to calculate the total number of words to be transferred. If a slave a smaller width than the master, it must use this result to determine when to complete the transfer. For example, in the case of a 128-bit master requesting a burst of four quadwords to a 64-bit slave, the slave must expect to perform eight doubleword transfers.

Note: Slaves that do not implement the fixed-length transfer ignore the PLB_BE signals during a burst transfer and continue bursting until the PLB_rdBurst signal or the PLB_wrBurst signal are negated by the master. 32-bit slaves that are connected to 64-bit, and above, implementations do not sample the PLB_BE(4:7) signals and do not know the full transfer length that is requested by 64-bit, and above, masters. They terminate the requested transfer prematurely when PLB_BE(4:7) is not ‘0000’. Also, fixed-length burst transfers to these 32-bit slaves with the PLB_BE(0:3) ‘0000’ and PLB_BE(4:7) not ‘0000’ cause the slave to continue bursting until the PLB_rdBurst signal or the PLB_wrBurst signal is negated by the master.

Masters that do not implement the fixed-length transfer must drive all 0’s on the byte enabled (BE) signals to be compatible with slaves that have implemented the fixed-length burst protocol. Slaves that do not implement the fixed-length transfer ignore the PLB_BE signals during a burst transfer and continue bursting until the PLB_rdBurst signal or the PLB_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_rdBurst signal or the PLB_wrBurst signals negated, 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 early, the slave must simultaneously assert the Sl_rdBTerm signal and the Sl_rdComp signal in the cycle before the last Sl_rdDAck signal. This allows a subsequent read or write transfer to be acknowledged in the Sl_rdComp cycle and thus fully use the read data bus. However, note that if the Sl_rdComp and Sl_rdBterm signals are asserted in the cycle before the last assertion of Sl_rdDAck, the slave must ignore the PLB_rdBurst signal in the following cycle if has not acknowledged or is not currently acknowledging a secondary read burst request. If the slave has acknowledged or is currently acknowledging a secondary read burst request, it can sample the PLB_rdBurst signal for the next transfer. PLB_rdBurst can be asserted because of the arbiter switching to a new read burst transfer in the cycle following the assertion of the Sl_rdComp signal or to the next burst transfer value being driven by the same master. See *Section 5.2.7 Pipelined Back-to-Back Fixed-Length Read Burst Transfers* on page 113.

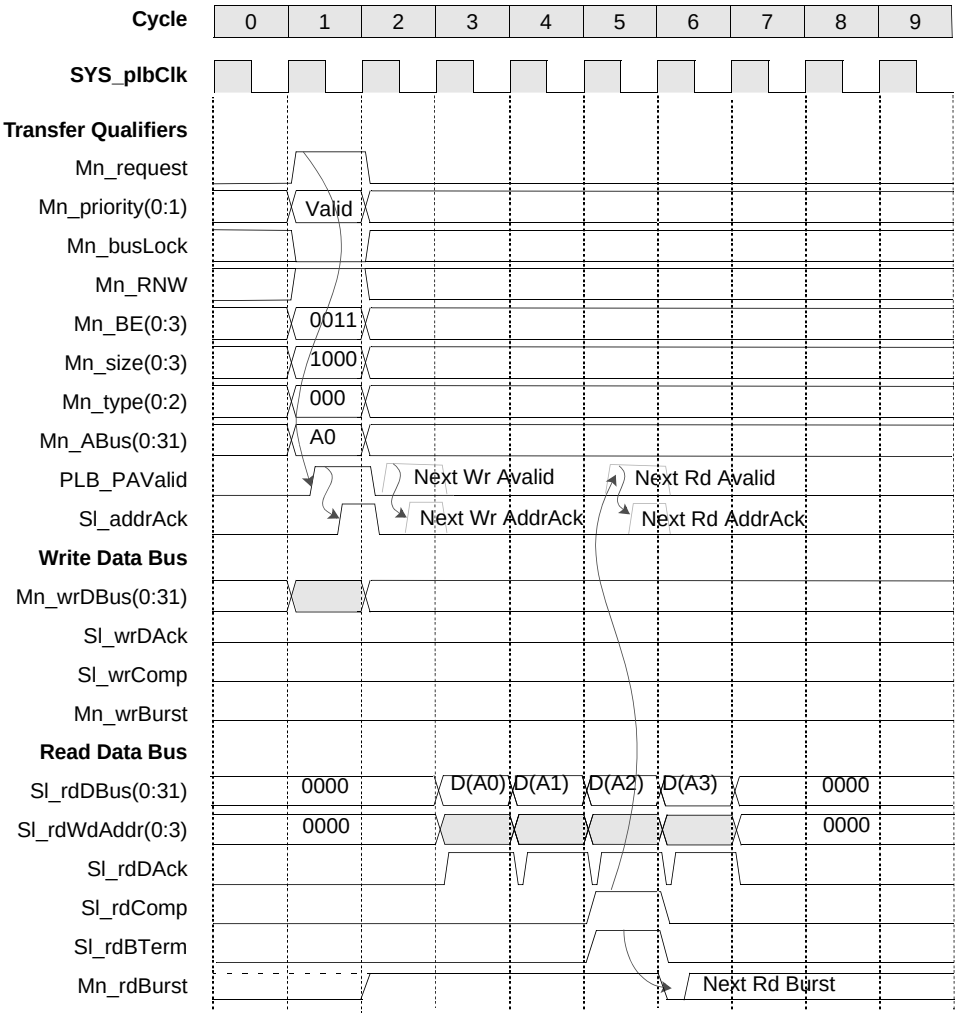
For write burst transfers, if the PLB_wrBurst signal is not negated early, the slave must assert the Sl_wrBTerm signal in the clock cycle before the last Sl_wrDAck signal and assert the Sl_wrComp signal in the same clock cycle as the assertion of the last Sl_wrDAck signal. This allows a subsequent read or write transfer to be acknowledged in the cycle following the assertion of the Sl_wrComp signal and thus fully use the write data bus.

Although the length of the transfer is provided to the slave during the request phase, the master is still required to assert the Mn_rdBurst signal or the Mn_wrBurst signal. Additionally, the master must negate the Mn_rdBurst or the Mn_wrBurst signal either in the clock cycle following the assertion of the Sl_rdBTerm or the Sl_wrBTerm signal or in the clock cycle following the assertion of the second to last Sl_rdDAck or Sl_wrDAck signal. This allows the transfer to complete when bursting to a slave that has not implemented the fixed-length burst protocol.

5.1.15 Fixed-Length Burst Read Transfer

Figure 5-14 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 before the last assertion of the Sl_rdDAck signal. This allows a subsequent read transfer to be acknowledged.

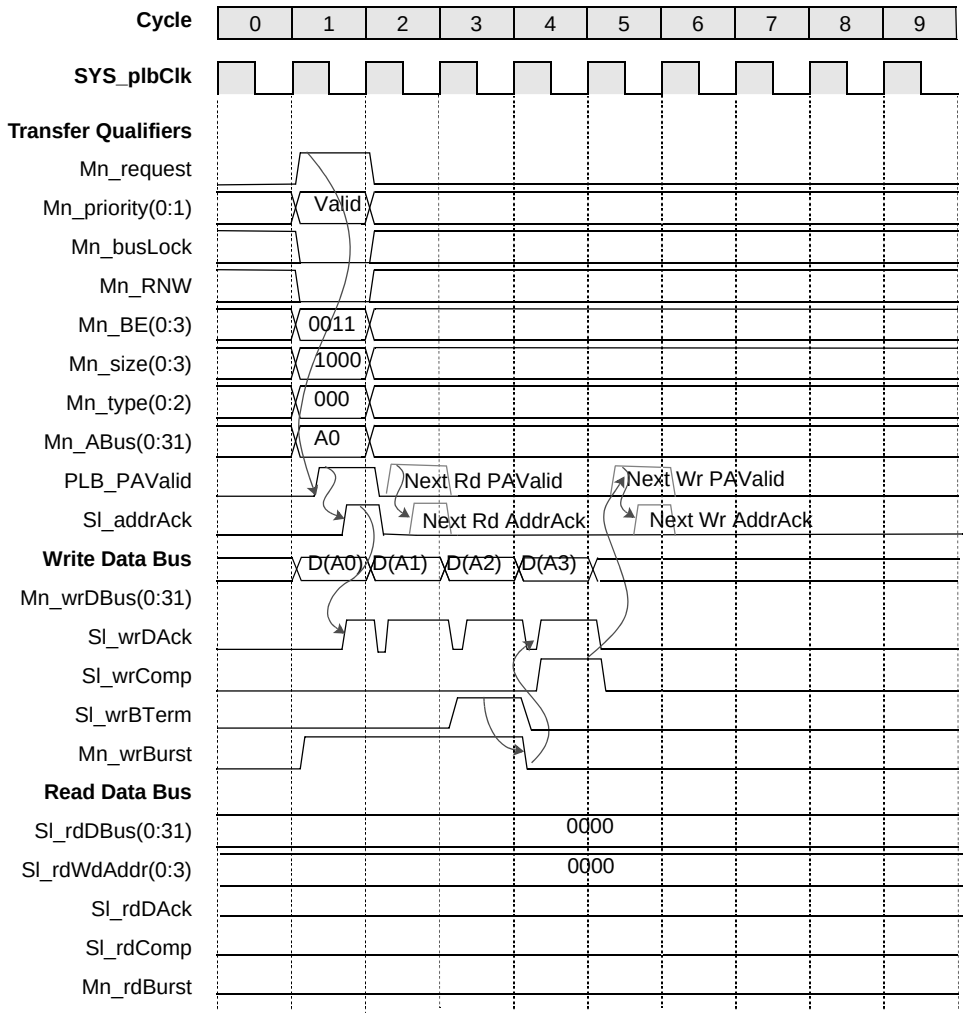
Figure 5-14. Fixed-Length Burst Read Transfer



5.1.16 Fixed-Length Burst Write Transfer

Figure 5-15 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 4 words. The slave uses this length value to count the number of transfers and assert the Sl_wrBTerm signal in the cycle before the last assertion of the Sl_wrDAck signal.

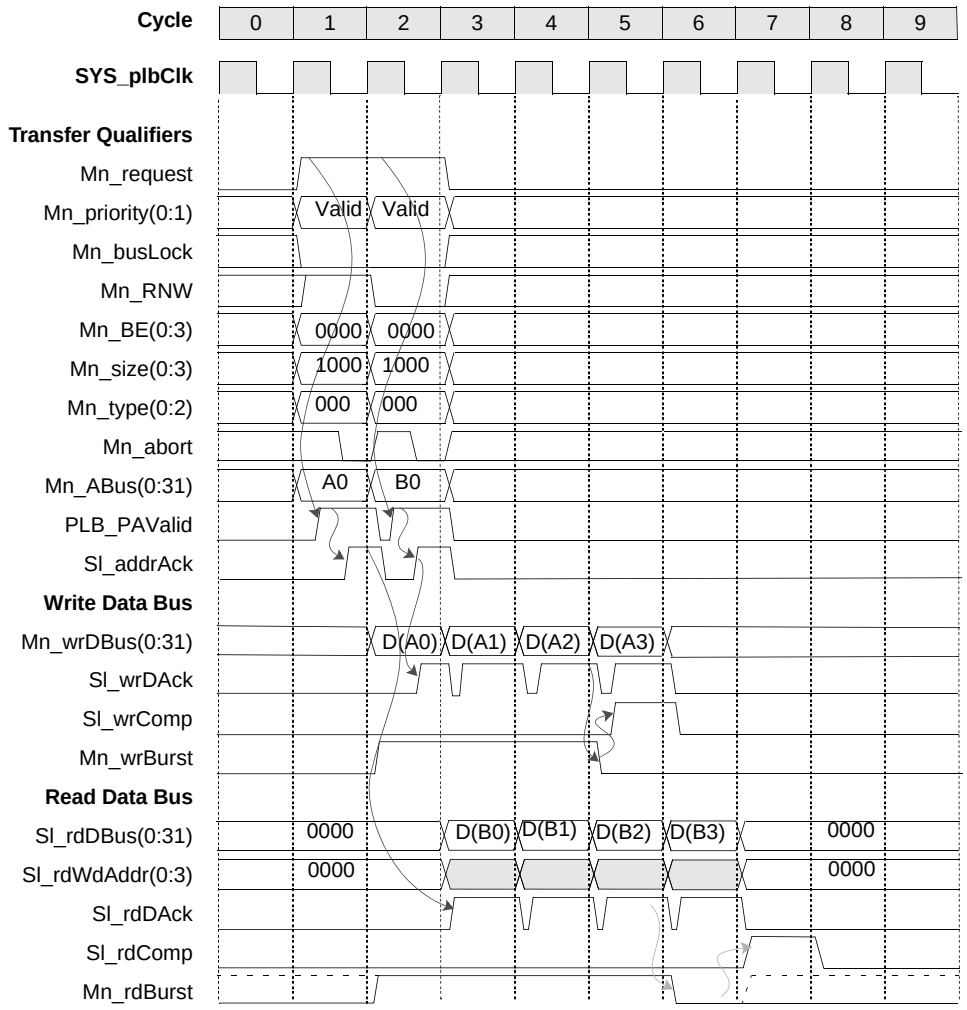
Figure 5-15. Fixed-Length Burst Write Transfer



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

Figure 5-16 shows the operation of a burst read followed immediately by a request for a burst write transfer on the PLB. The master is only required to drive the address bus and transfer qualifiers until the slave acknowledges the address. 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 can continue up to the maximum burst length that the slave device supports.

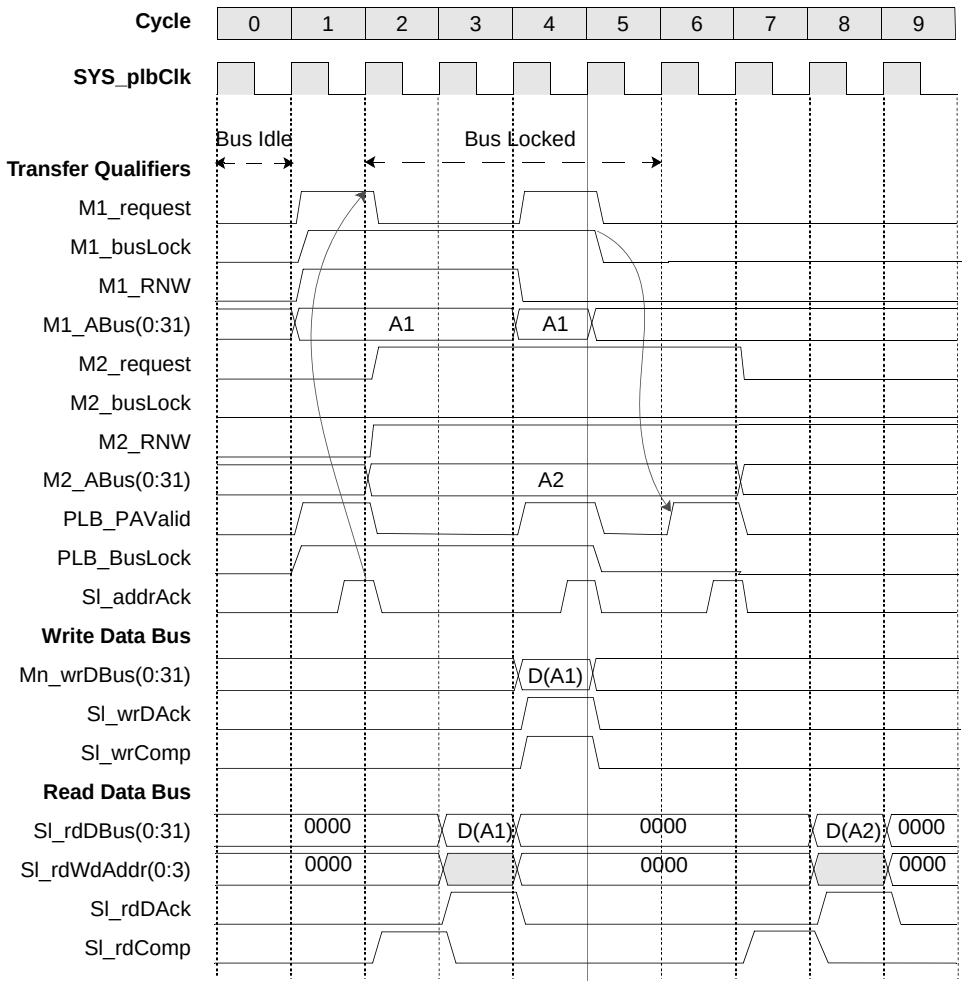
Figure 5-16. Back-to-Back Burst Read /Burst Write Transfers



5.1.18 Locked Transfer

Figure 5-17 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 wants to lock the bus during the current data transfer. Although not illustrated in *Figure 5-17*, the arbiter asserts the PLB_PAValid signal only after detecting that both data buses are idle. The slave then asserts the Sl_addrAck signal, causing the arbiter to lock the bus. The arbiter ignores a second master request 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 rearbitrated and granted to the second master.

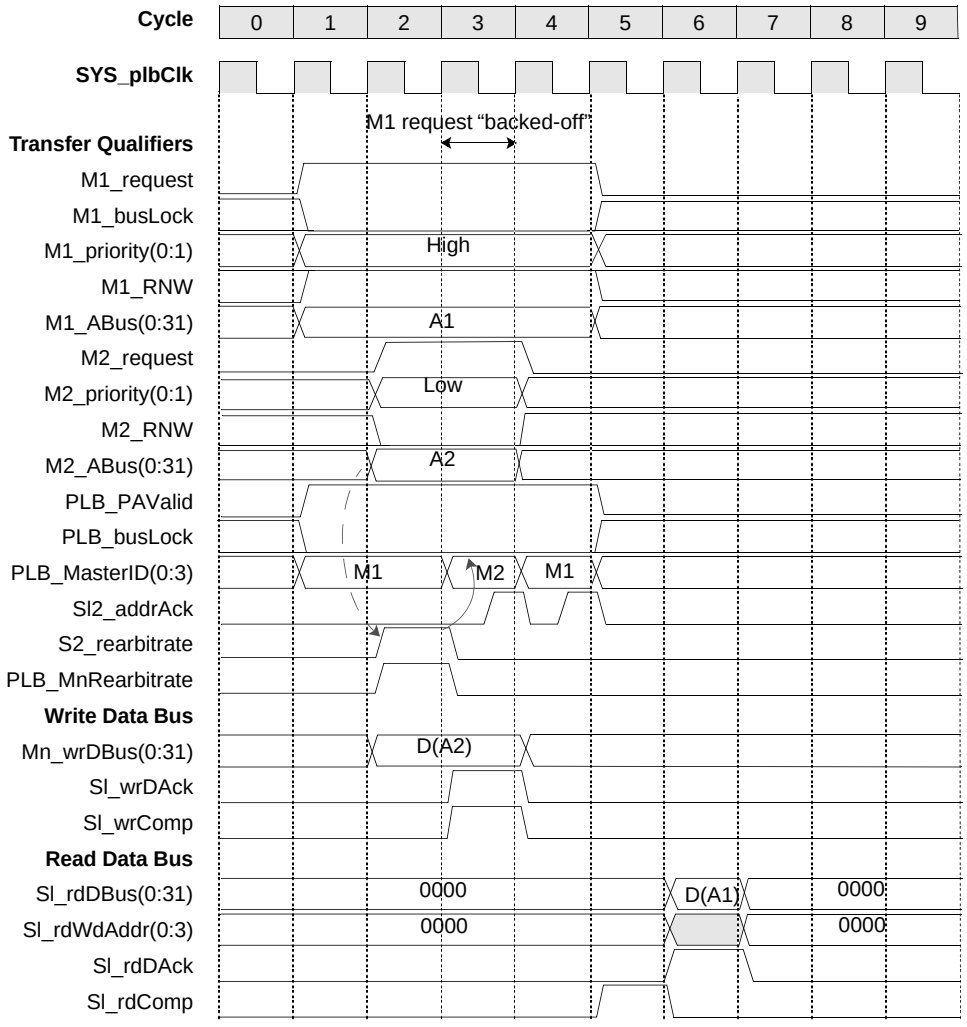
Figure 5-17. Locked Transfer



5.1.19 Slave Requested Rearbitration with Bus Unlocked

Figure 5-18 illustrates a scenario in which a master device or slave device is unable to respond to a PLB data transfer that is initiated by another master until it has first executed a PLB transfer of its own. As a result, the master device or slave device asserts its Sl_rearbitrate signal to request rearbitration of the PLB bus. In response to the assertion of the Sl_rearbitrate signal, the PLB arbiter backs-off the initial request and rearbitrates the bus in the following clock cycle, allowing the master device or slave device to have its request serviced ahead of the initial request. The PLB_PAValid signal is never dropped; instead, the arbiter 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.

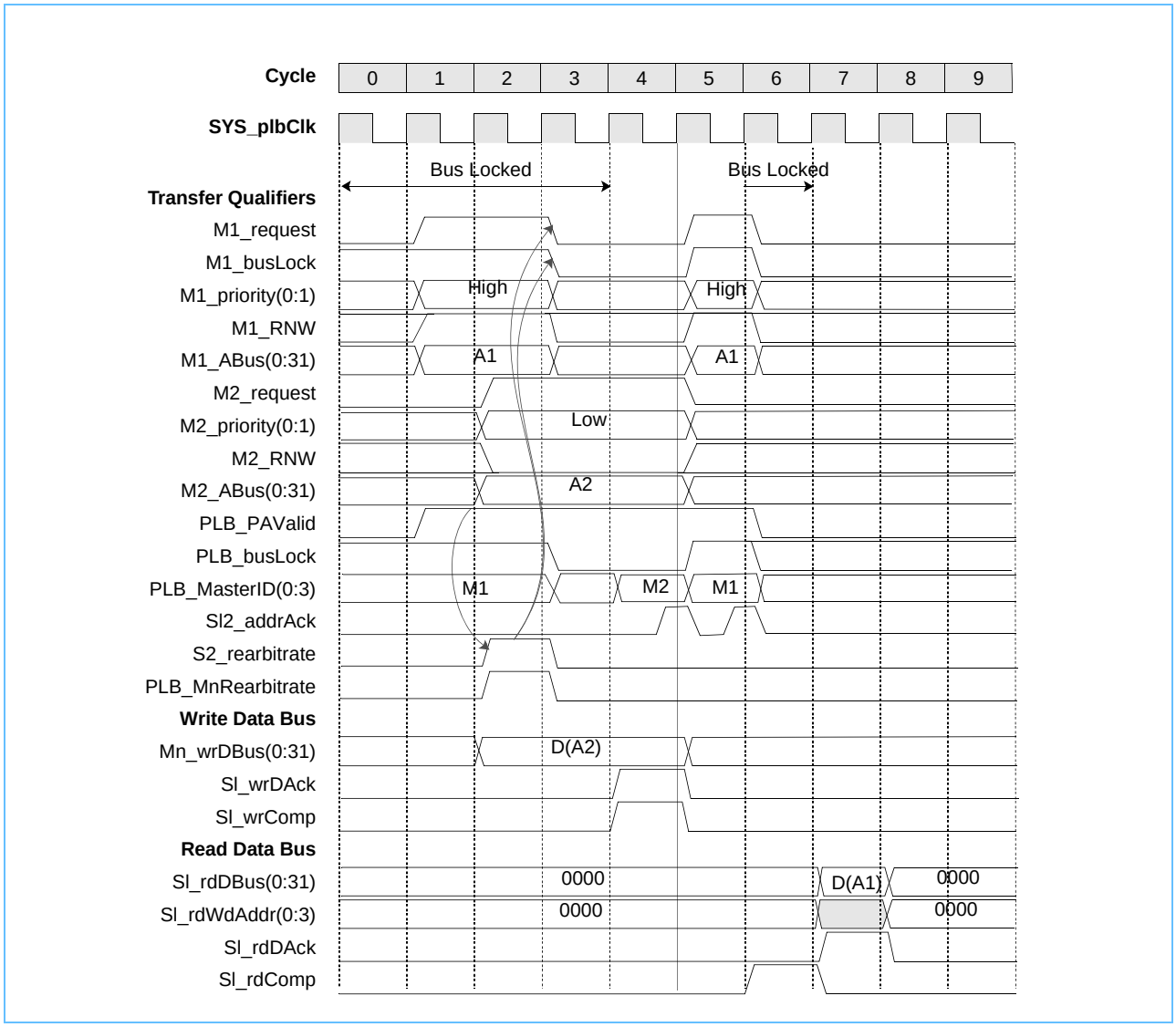
Figure 5-18. Slave Requested Rearbitration with Bus Unlocked



5.1.20 Slave Requested Rearbitration With Bus Locked

Figure 5-19 illustrates a scenario in which a master device or slave device is unable to respond to a PLB data transfer that is initiated by another master until it has first executed a PLB transfer of its own. As a result, the master device or slave device asserts its Sl_rearbitrate signal to request rearbitration 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 clock cycles, allowing the PLB arbiter to rearbitrate the bus in the following clock cycle. Therefore, the master device or slave device request can be serviced ahead of its initial request, averting a possible dead-lock scenario.

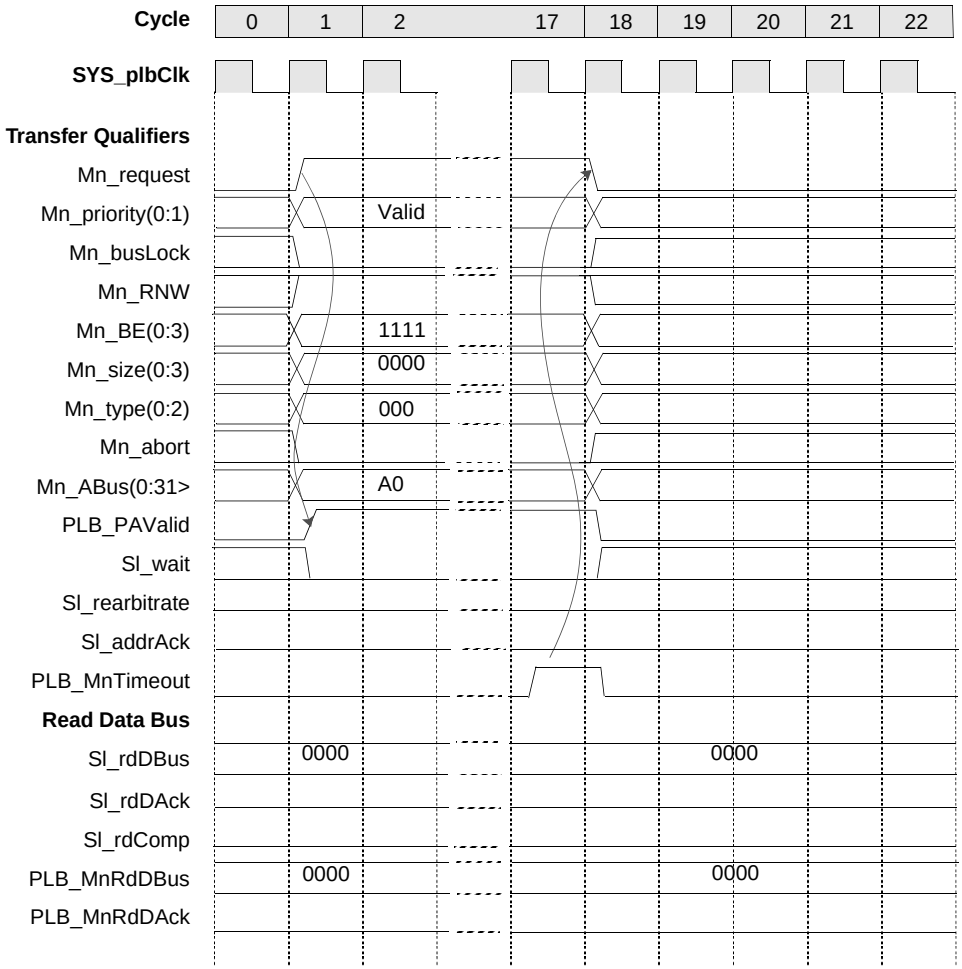
Figure 5-19. Slave Requested Rearbitration with Bus Locked



5.1.21 Bus Timeout Transfer

Figure 5-20 shows a bus timeout for a read transfer on the PLB. The PLB arbiter asserts the PLB_MnTimeout signal seventeen cycles after the initial assertion of the PLB_PAValid signal. The master deasserts its request in the next cycle.

Figure 5-20. Bus Timeout Transfer



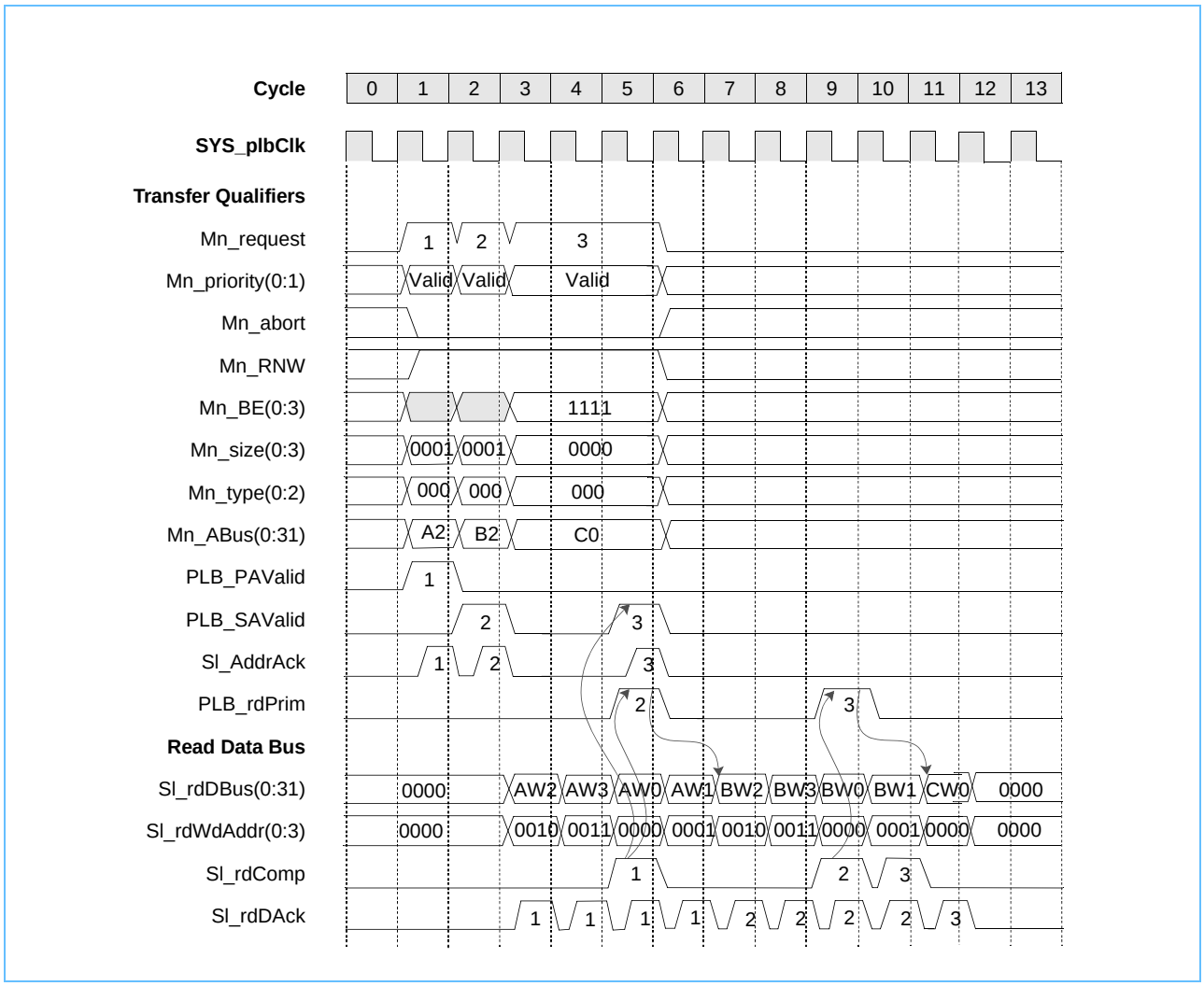
5.2 2 Deep PLB Address Pipelining

The timing diagrams included in this section are examples of 2-deep address-pipelined read and write transfers on the PLB. In this case only single bit PLB_rdPrim and PLB_wrPrim signals are necessary. Also, the Sl_addrAck signal that the arbiter samples can be the output of the ORing of the slave Sl(n)_addrAck signal. However, the signal assertion and negation times shown in these figures 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 5-21 shows the operation of three back-to-back read transfers involving three masters and a slave device that support address pipelining on the PLB. For all transfers, the slave asserts the Sl_rdComp signal in the clock cycle preceding the Sl_rdDAck signal. 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 cannot assert its Sl_rdDAck signal for the data read until two clock cycles following the assertion of the corresponding Sl_addrAck signal. For the secondary read requests, the slave cannot assert its Sl_rdDAck signal or drive the Sl_rdDBus until two clock cycles following the assertion of the PLB_rdPrim signal. 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 can read data every clock cycle from a slave that is capable of providing data in a single clock cycle.

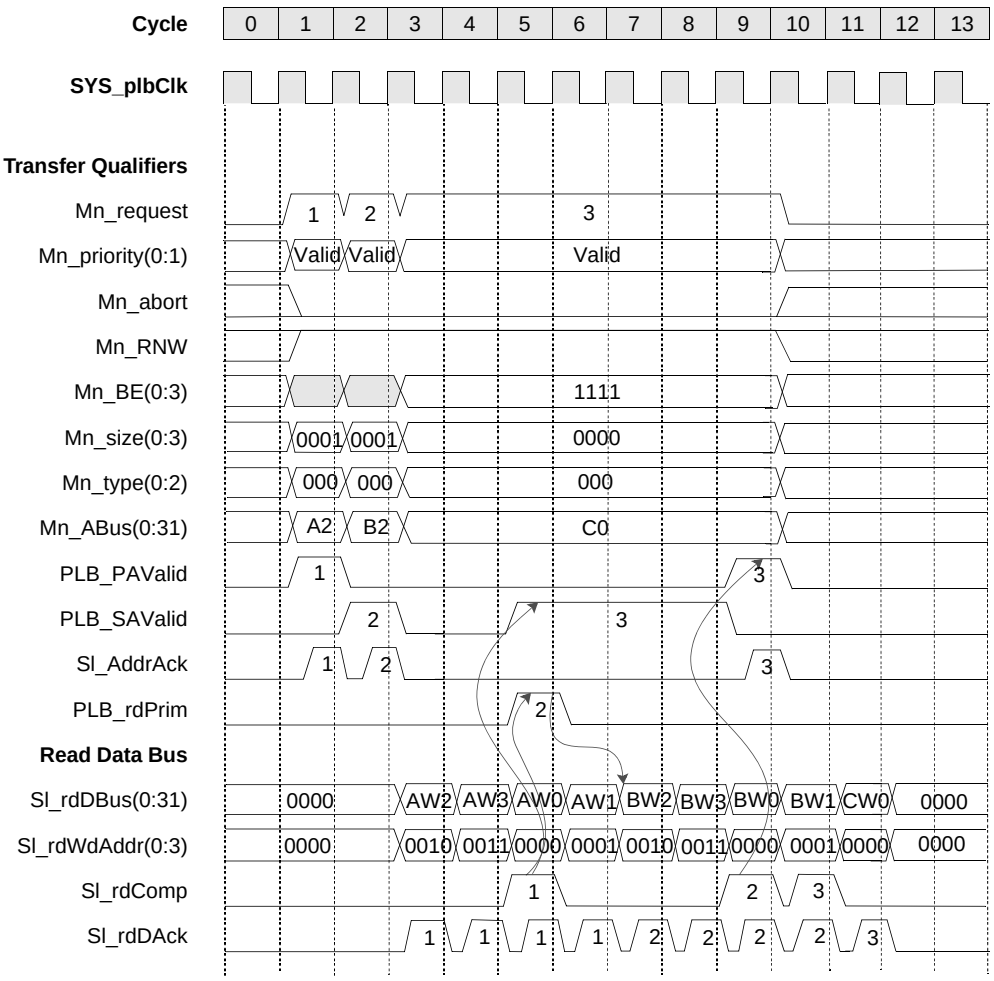
Figure 5-21. Pipelined Back-to-Back Read Transfers



5.2.2 Pipelined Back-to-Back Read Transfers - Delayed AAck

Figure 5-22 is similar to *Figure 5-21 Pipelined Back-to-Back Read Transfers* on page 107, with one exception. For the request that master 3 made, the PLB_SAValid signal is negated and the PLB_PAValid signal is asserted before the slave asserts the Sl_addrAck signal. The assertion of PLB_PAValid for the last read request is made possible by the assertion of Sl_rdComp for the previous secondary request. Also, the PLB_rdPrim signal is not asserted for this request.

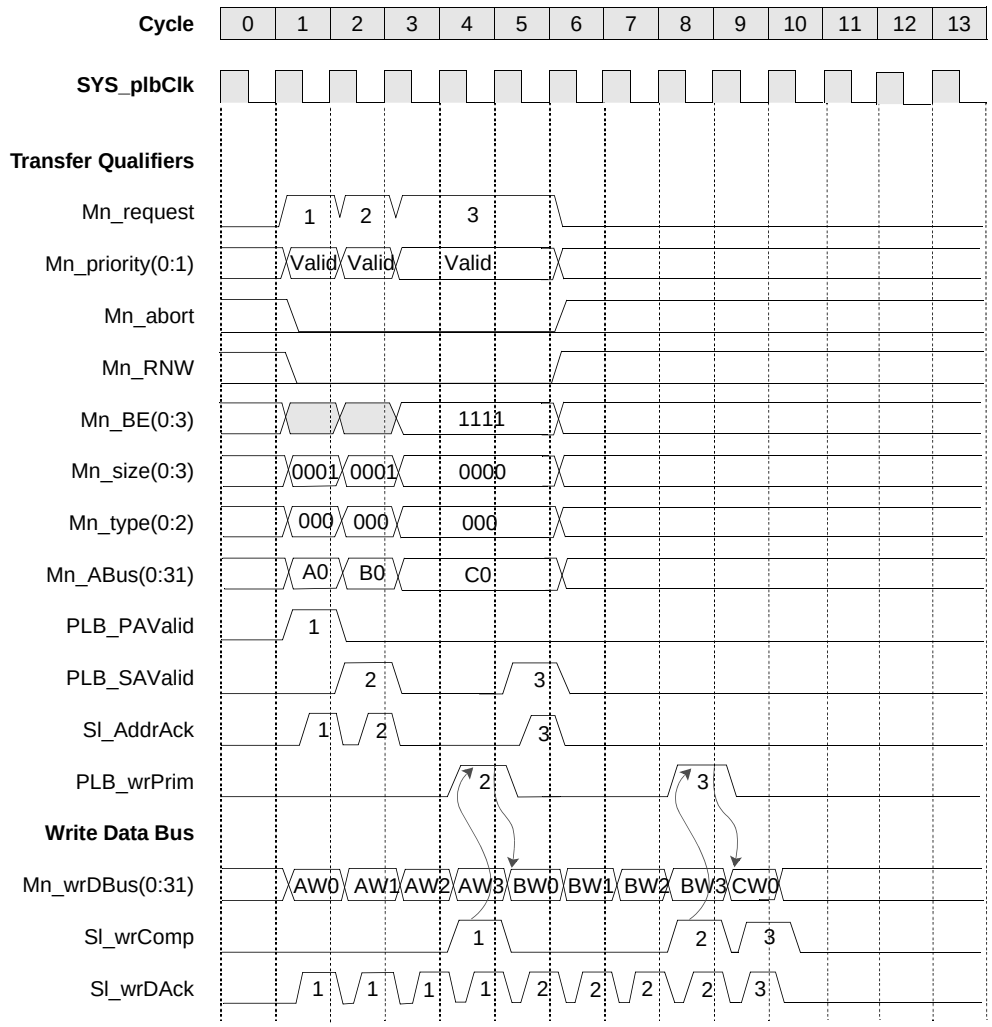
Figure 5-22. Pipelined Back-to-Back Read Transfers - Delayed AAck



5.2.3 Pipelined Back-to-Back Write Transfers

Figure 5-23 shows the operation of three back-to-back write transfers involving three masters and a slave device that support address pipelining on the PLB. For the primary write request, the slave can assert the Sl_wrComp and Sl_wrDAck signals in the same clock cycle that the Sl_addrAck signal is asserted. For the secondary write requests, the slave cannot assert its Sl_wrDAck signal for the written data until the clock cycle following the assertion of the PLB_wrPrim signal. 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.

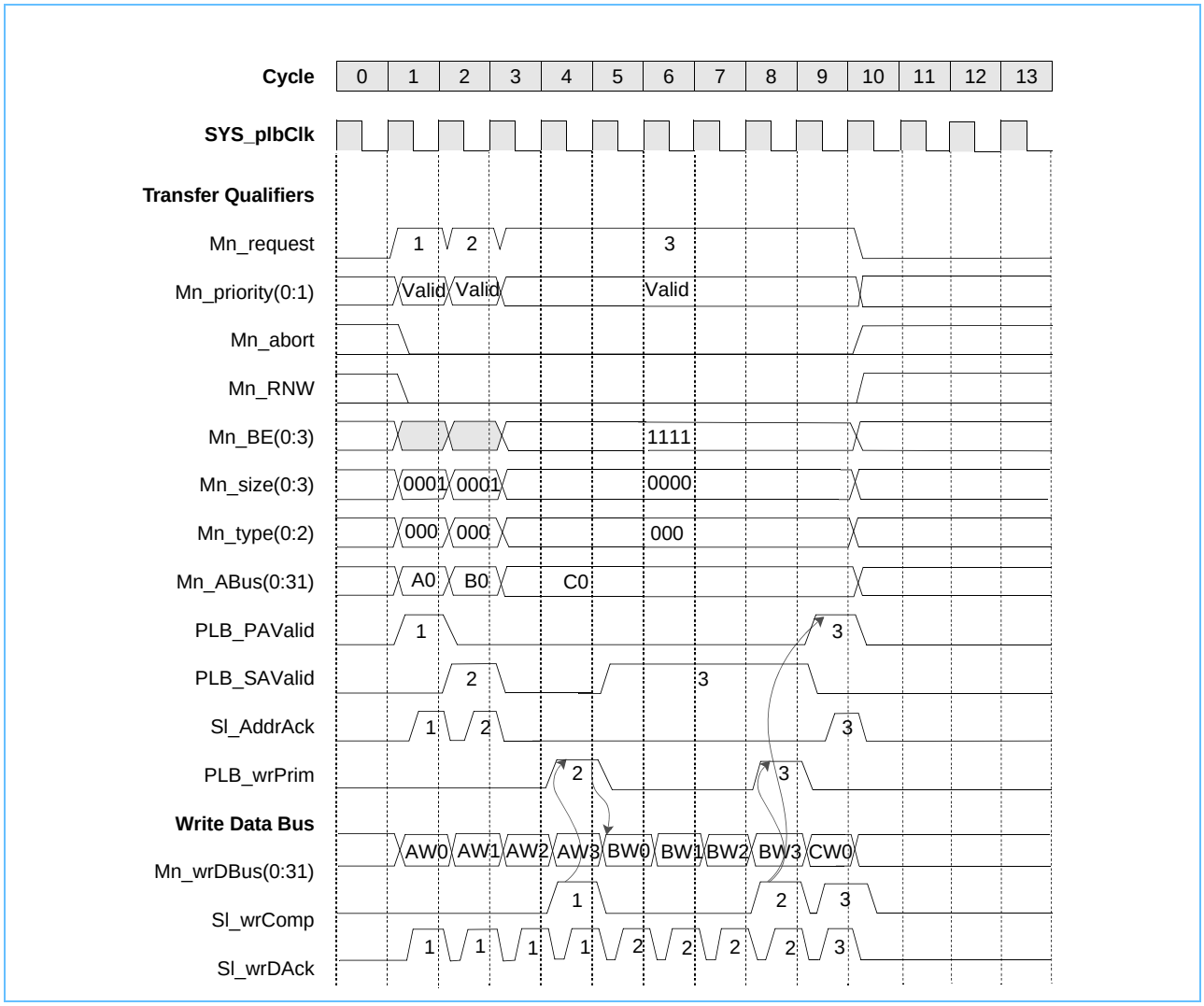
Figure 5-23. Pipelined Back-to-Back Write Transfers



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

Figure 5-24 is similar to *Figure 5-23 Pipelined Back-to-Back Write Transfers* on page 109, with one exception. For the write request in the series from master 3, the PLB_SAValid signal is negated and the PLB_PAValid signal is asserted before the slave's assertion of the Sl_addrAck signal. The assertion of the PLB_PAValid signal for the last write request is made possible by the assertion of the Sl_wrComp signal for the previous secondary request. Also, the PLB_wrPrim signal is not asserted for this request.

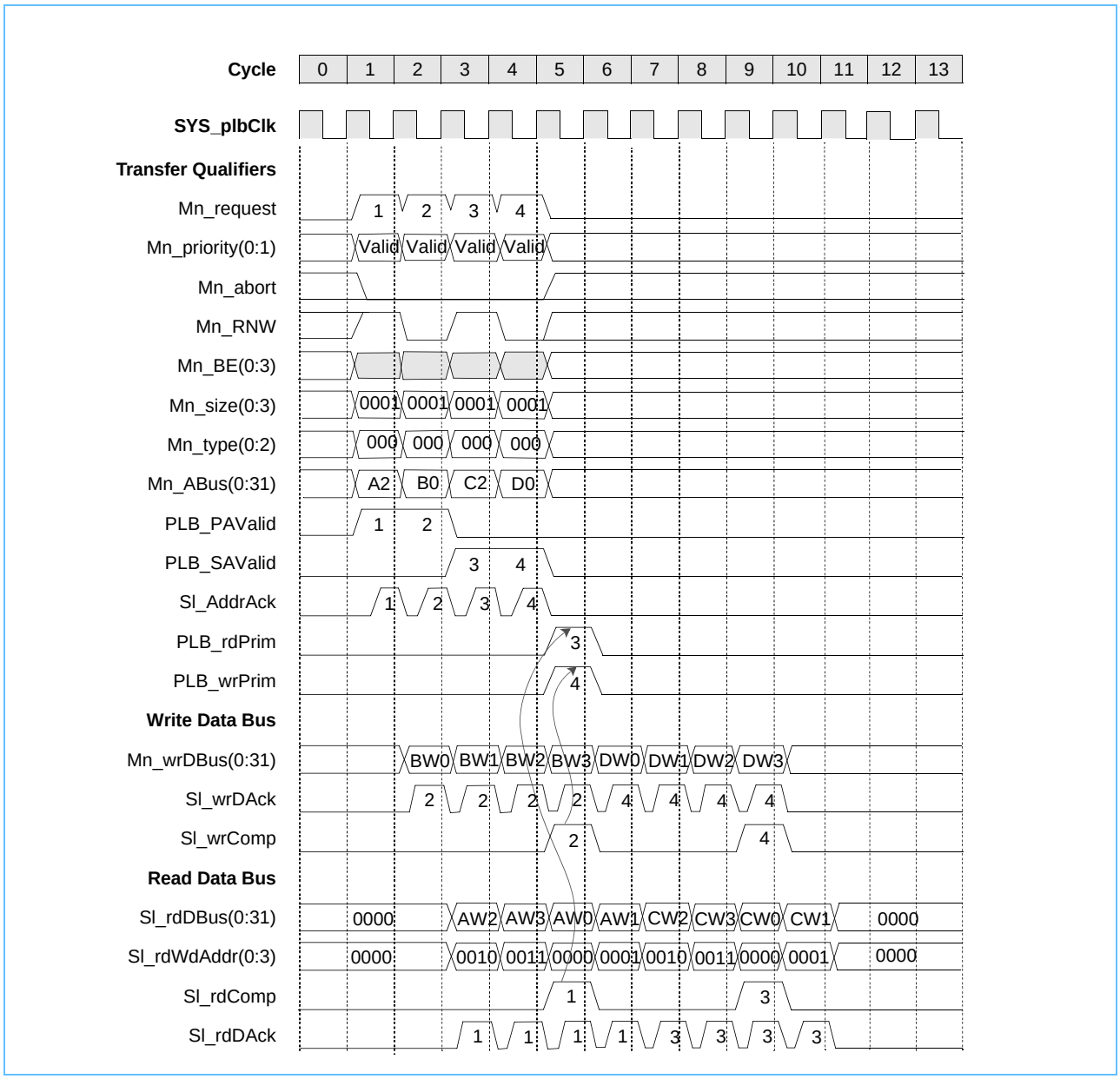
Figure 5-24. Pipelined Back-to-Back Write Transfers - Delayed AAck



5.2.5 Pipelined Back-to-Back Read and Write Transfers

Figure 5-25 shows the operation of four back-to-back read and write transfers involving four masters and a slave device that support address pipelining on the PLB.

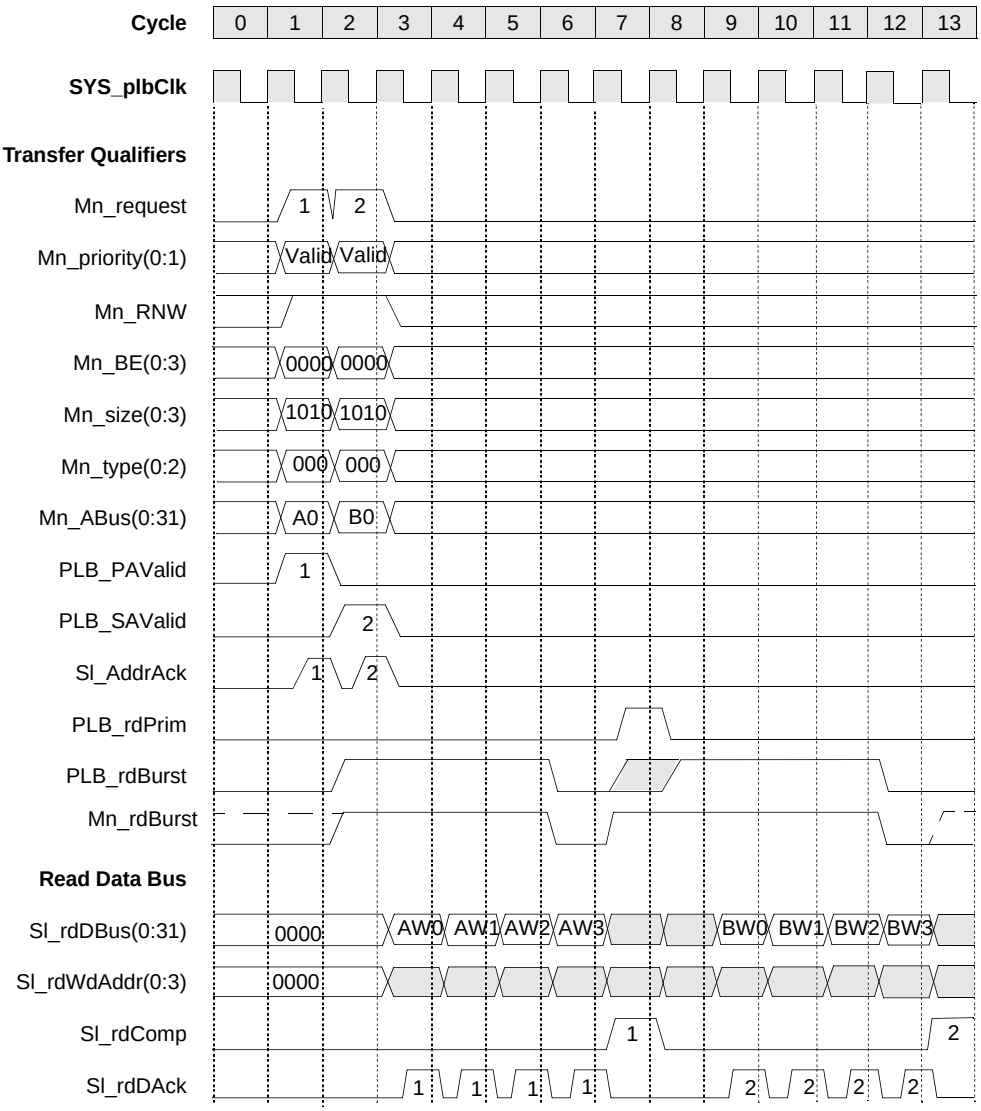
Figure 5-25. Pipelined Back-to-Back Read and Write Transfers



5.2.6 Pipelined Back-to-Back Read Burst Transfers

Figure 5-26 shows the operation of two back-to-back read burst transfers involving a master and a slave device that support address pipelining on the PLB. The Mn_rdBurst signal must be negated during the last data transfer for the first request before it is reasserted for the second request.

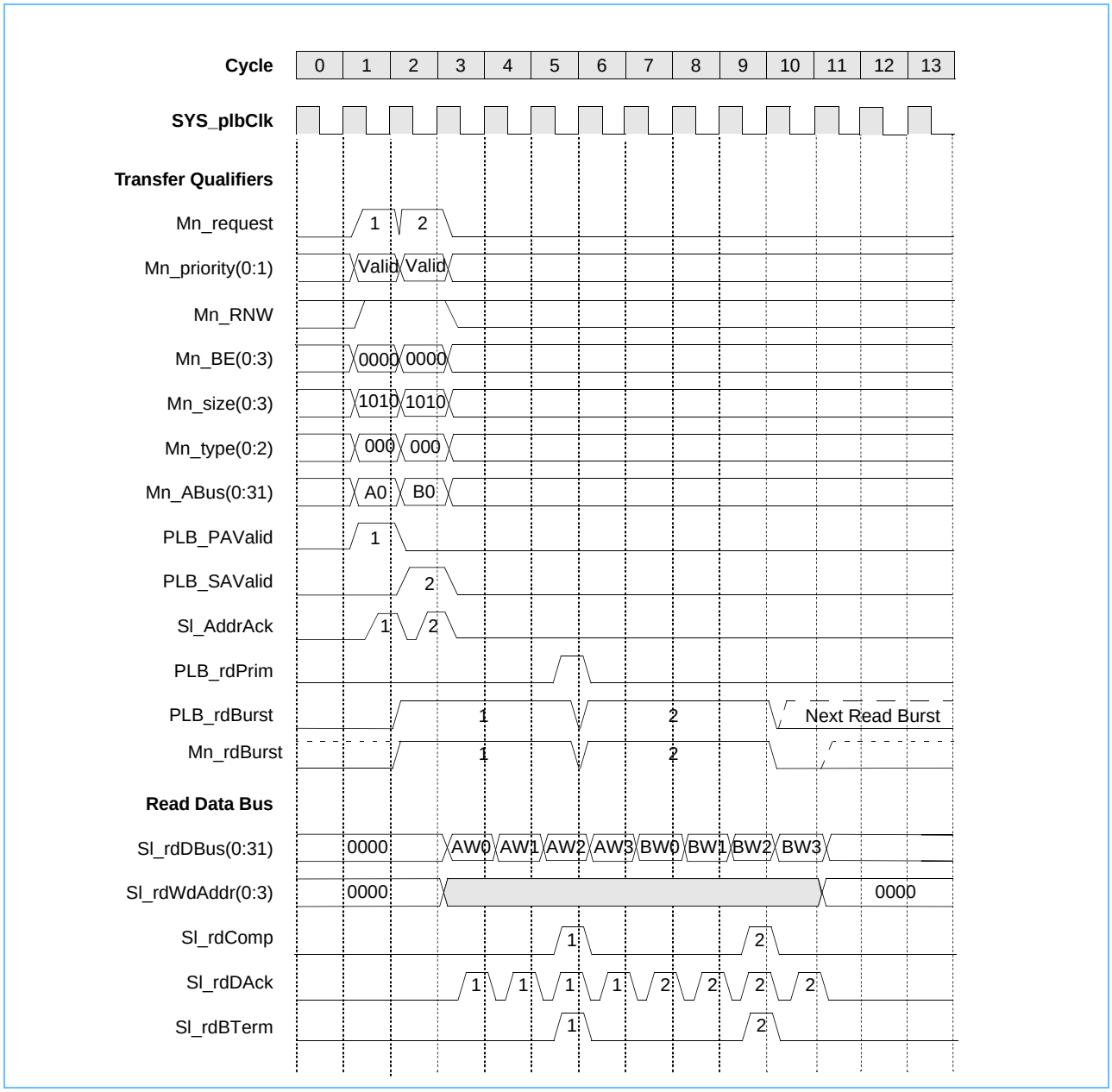
Figure 5-26. Pipelined Back-to-Back Read Burst Transfers



5.2.7 Pipelined Back-to-Back Fixed-Length Read Burst Transfers

Figure 5-27 shows the operation of two back-to-back fixed-length read burst transfers involving a master and a slave device that support address pipelining on the PLB. The master must advance the Mn_rdBurst signal to the secondary pipelined transfer value in clock 6, the cycle following the assertion of Sl_rdBTerm.

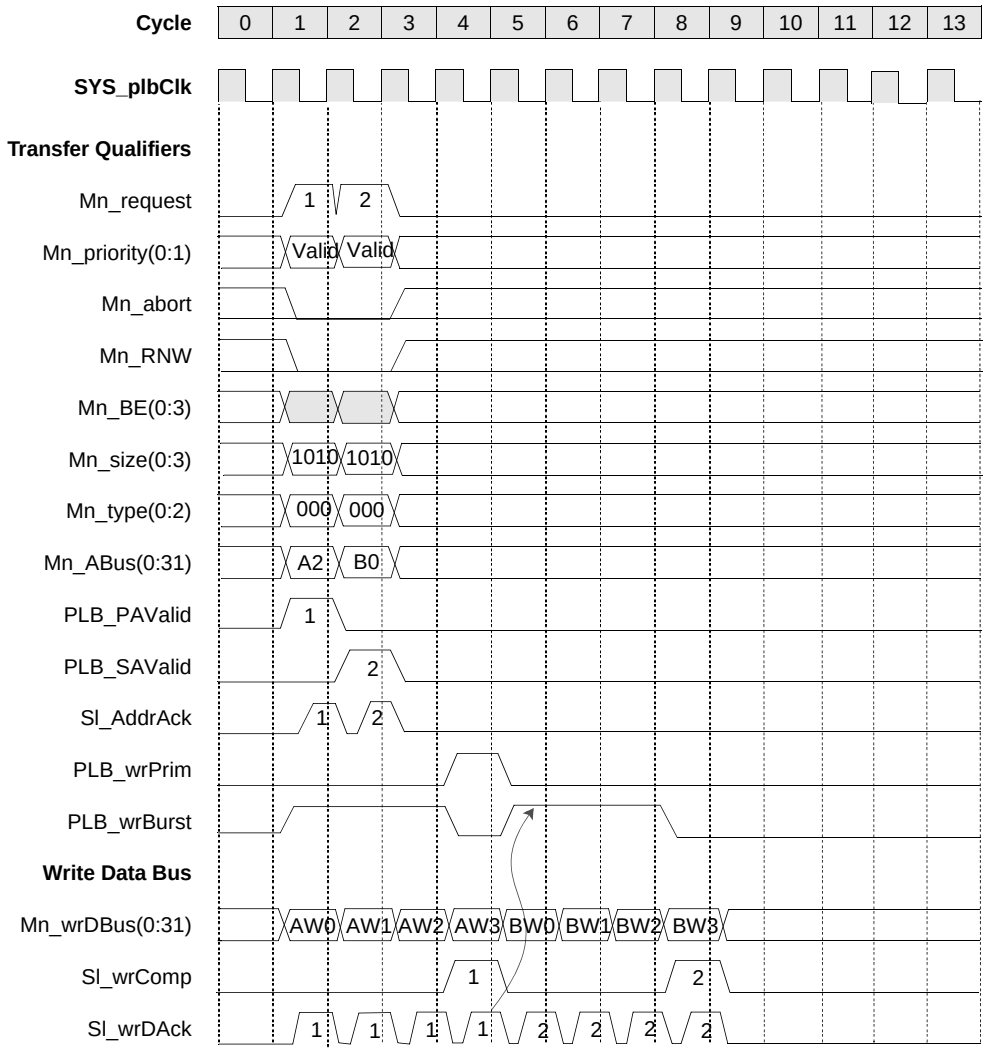
Figure 5-27. Pipelined Back-to-Back Fixed-Length Read Burst Transfers



5.2.8 Pipelined Back-to-Back Write Burst Transfers

Figure 5-28 shows the operation of two back-to-back write burst transfers involving a master and a slave device that support address pipelining on the PLB. The Mn_wrBurst signal must be reasserted for the second request in the cycle immediately following the last data transfer for the first request.

Figure 5-28. Pipelined Back-to-Back Write Burst Transfers



5.3 N Deep PLB Address Pipelining

The PLB architecture supports broadcasting pending master requests to the slaves during the busy state of the data buses. This capability provides for a pipelining of transfers onto the bus. This allows slaves to allocate resources or prefetch data before their respective tenure on the requested data bus. The advantage of this operation is that general latency is reduced and overall bus throughput can be significantly increased.

In the simplest case, the address pipelining of the PLB is implemented as two deep. This means that a primary and secondary, or pipelined, transfer can both be broadcast and acknowledged on the bus at one time. In a more general case, the PLB supports unlimited independent pipelining of each data bus. In order to support this, the PLB arbiter implementation is required to track the master and slave that are involved in a particular pipelined transfer and assure the proper slave is notified when a previously acknowledged pipelined transfer is now considered primary. The arbiter is also responsible for steering the slave responses to the correct master. Arbiter implementations are also required to provide the highest priority pending read and write request in the acknowledged pipelines.

To achieve this operation, the arbiter must sample each Sl_addrAck signal independently, before the bus OR logic, up to the number of slaves that are supported: Sl_addrAck(0:7) for an eight master arbiter for example. The arbiter must also provide a separate PLB_rdPrim and the PLB_wrPrim signal to each slave to notify one and only one slave that its position in the pipeline has been promoted to the primary transfer: PLB_rdPrim(0:7) andPLB_wrPrim(0:7) for an eight master arbiter for example. The mechanism for broadcasting all pipelined transfers is through the PLB_SAValid signal. For a particular type of transfer, read or write, each subsequent assertion of the PLB_SAValid signal by the arbiter without an intervening primary PLB_PAValid signal assertion is considered an increase the depth of pipelining.

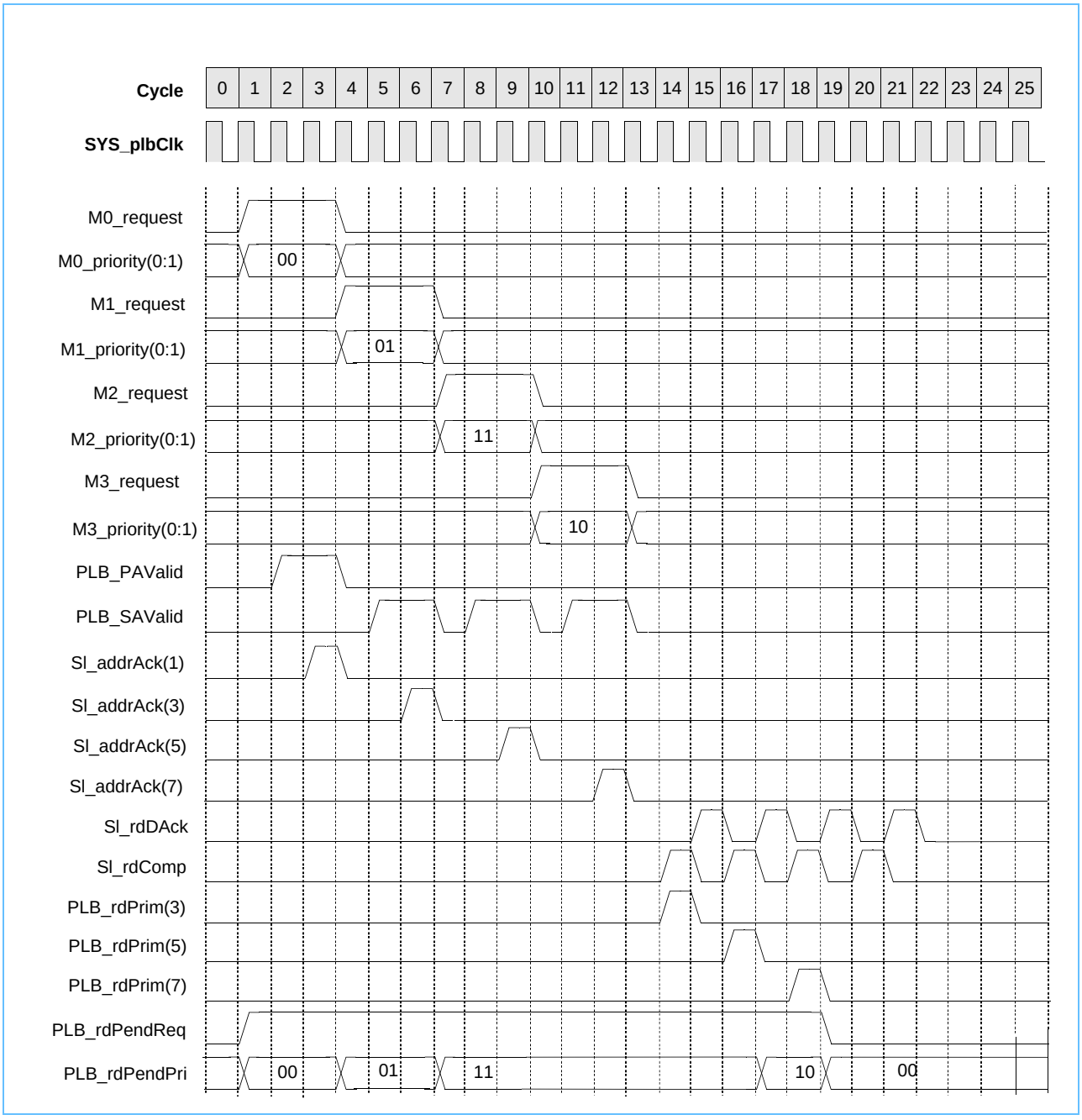
From a master viewpoint, it is unaware that any pipelining is occurring at all. The master receives its PLB_MnaddrAck signal and awaits the appropriate data acknowledgment. From a slave viewpoint, it acknowledges a pipelined transfer without concern for its position in the pipeline. Furthermore, the master can request additional pipelined requests for the same bus on the condition that it continues to receive PLB_MnaddrAck assertions. The slave might or might not acknowledge subsequent assertions of the PLB_SAValid signal to claim as many pipelined transfers as possible. Because there is no bus timeout for pipelined transfers, slaves that cannot acknowledge a secondary request in a reasonable amount of time must assert the Sl_rearbitrate signal to allow the arbiter to advance to the next pending bus request. This allows for optimal use of the address and transfer qualifier buses.

The timing diagrams included in this section are examples of a 4-deep address-pipelined read and a 4-deep address pipelined write on the PLB.

5.3.1 4-Deep Read Pipelining

Figure 5-29 4-Deep Read Pipelining on page 116 shows 4-deep read pipelining, one primary and three pipelined, by four different master read requests. Different slaves acknowledge each transfer request. The arbiter generates the pending priority of the highest request priority of the current or pipelined master. All slave Sl_addrAck(n) signals are sampled by the arbiter. Independent PLB_rdPrim(n) signals are generated to the appropriate slave to notify them that they are now the primary transfers and can drive the data bus and the rdDAck signal.

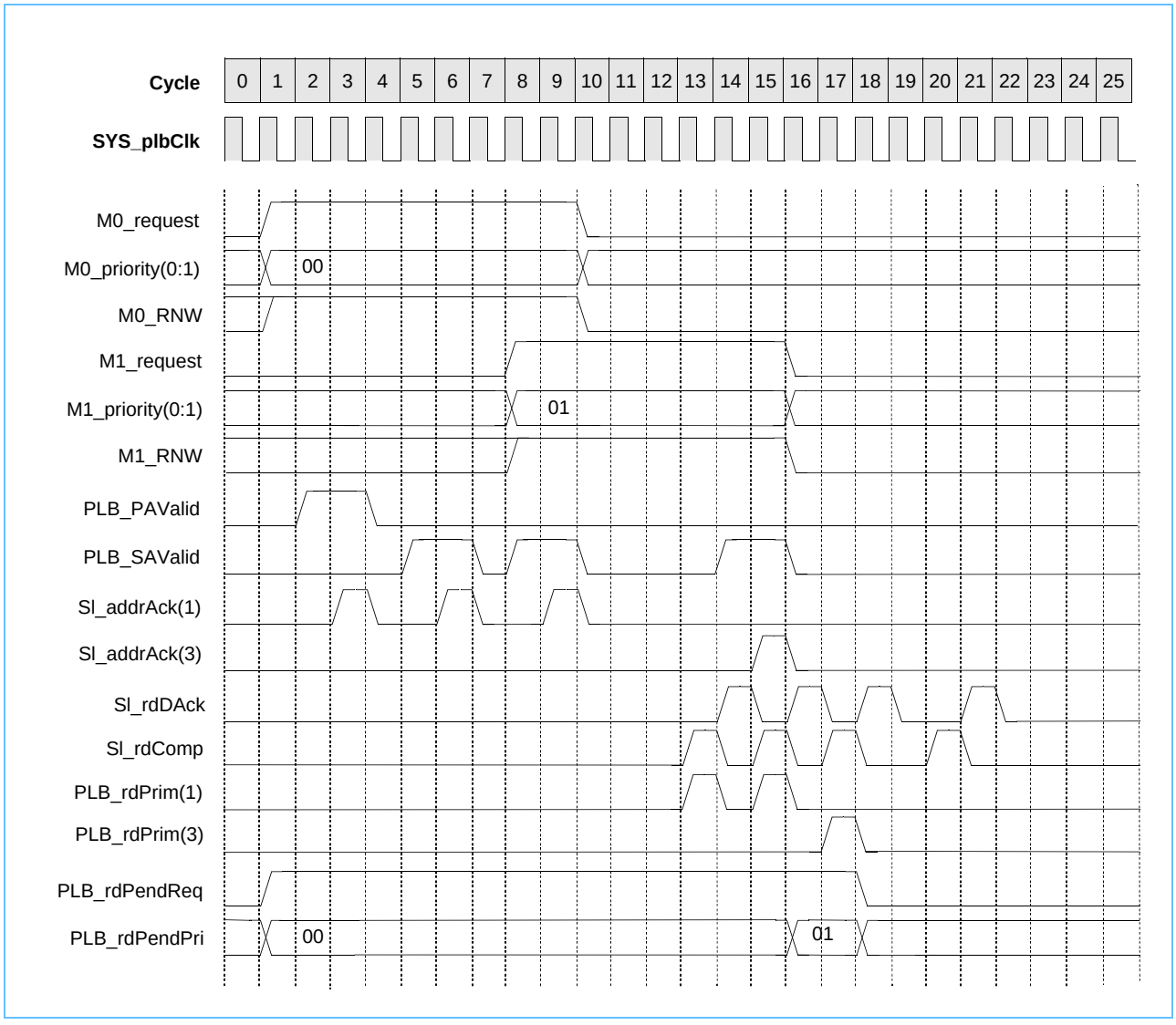
Figure 5-29. 4-Deep Read Pipelining



5.3.2 3-Deep Read Pipelining

Figure 5-30 shows an arbiter that only supports a 3-deep read pipeline. The read pipeline is filled by three read requests from master 0. Slave 1 acknowledges each transfer request. Master 1 requests a read after the PLB_SAValid signal is asserted for the third read. The read pipeline fills in the next clock. PLB_SAValid is asserted when the primary request completes and a location in the read pipeline becomes available. Slave 3 acknowledges the read. The arbiter generates the pending priority of the highest request priority of the current or pipelined master. Independent PLB_rdPrim(n) signals are generated to the appropriate slaves to notify them that they are now the primary transfer and can drive the data bus.

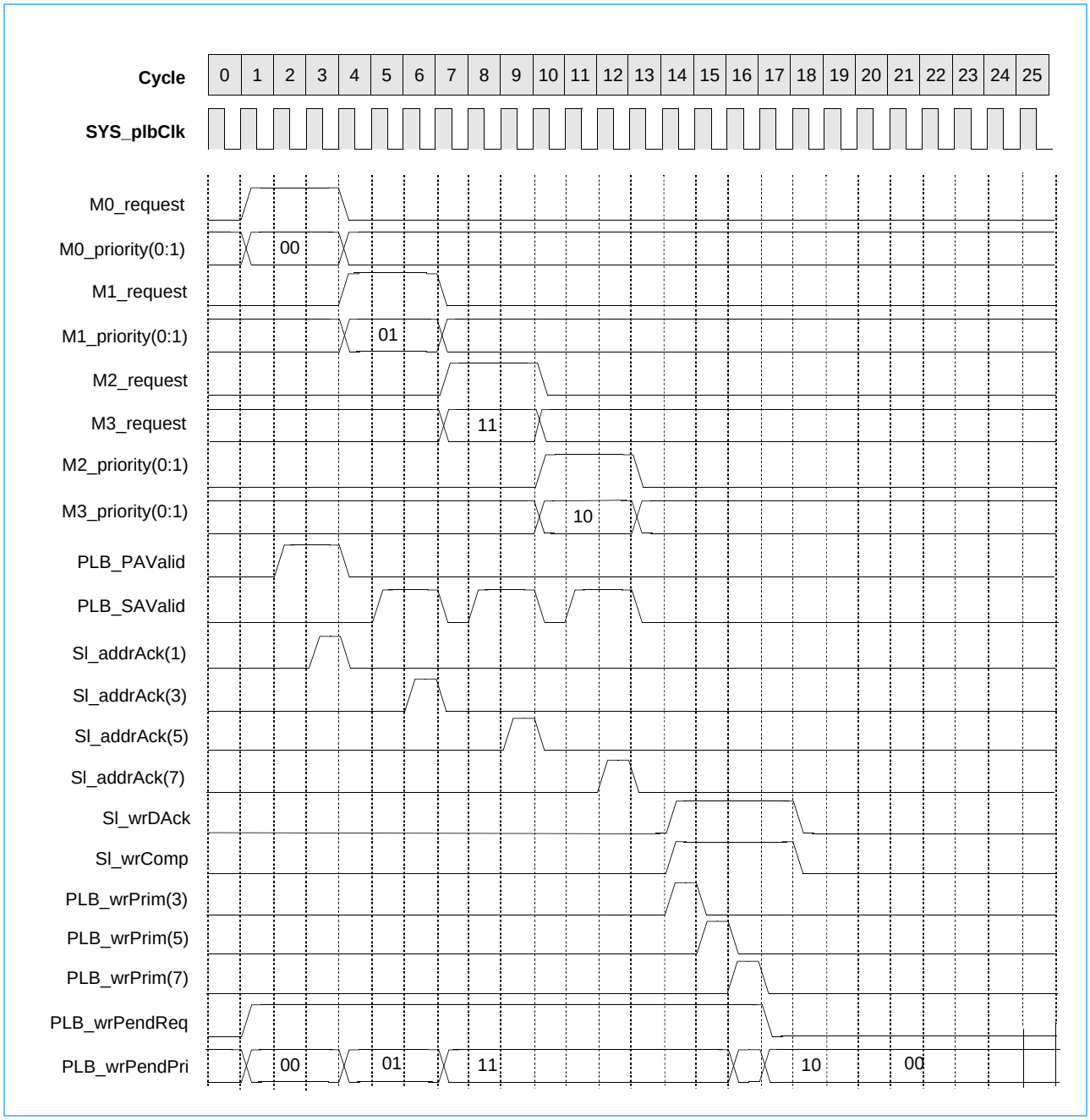
Figure 5-30. 3-Deep Read Pipelining



5.3.3 4-Deep Write Pipelining

Figure 5-31 shows 4-deep write pipelining, one primary and three pipelined, by four different master write requests. Different slaves acknowledge each transfer request. The arbiter generates the pending priority of the highest request priority of the current or pipelined master. All slave Sl_addrAck(n) signals are sampled by the arbiter. Independent PLB_wrPrim(n) signals are generated to the appropriate slave to notify them they are now the primary transfers and can latch the data bus and assert the Sl_wrDAck signal.

Figure 5-31. 4-Deep Write Pipelining



5.4 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 latency in a particular application, a master latency timer is provided in each master capable of long burst transfers. This timer assures latency for a particular master and also guarantees a certain amount of bandwidth for a bursting master.

5.4.1 PLB Master Latency Timer

The master latency timer is a programmable timer that limits a master's latency on the PLB bus when long burst transfers are performed. Each master that is capable of performing a long burst, greater than four data transfers, is required to have a latency timer. Two registers are required to implement the master latency timer:

- The Latency Count Register (10 bits; the 4 low-order bits can be hardwired)
- The Latency Timer (10 bits)

The Latency Count Register is programmable, can be read or written by software, and can be either memory-mapped or device-control-register-bus mapped. The 4 low-order bits of the Latency Count register can be hardwired such that the minimum latency value is sixteen clock cycles and the granularity of the count is sixteen clock cycles (that is, you can program counts of 16, 32, 48, and so on, clock cycles). The Latency Count Register must be designed so that it is cleared by reset.

The Latency Timer is used as clock cycle counter and is not accessible by code. The Latency Timer is cleared and disabled when the master is not performing a burst data transfer on the bus and during reset. During burst data transfers, the Latency Timer is enabled and begins counting the clock cycle after the first data acknowledgment, Sl_rdDAck or Sl_wrDAck, is asserted by the slave device.

5.4.2 PLB Master Latency Timer Expiration

Upon expiration of the Latency Timer for a given burst transfer, 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, any one of the three following options can be implemented in a master:

- The master can monitor the PLB_rdPendReq and PLB_rdPendPri(0:1) signals for a read burst or the PLB_wrPendReq and PLB_wrPendPri(0:1) signals for a write burst continuously. The master can negate the burst signal immediately after the Latency Timer has expired and a pending request of equal or higher priority is detected.
- The master can monitor the PLB_rdPendReq or PLB_wrPendReq signal continuously. The master can also negate the rdBurst or wrBurst signal immediately after the Latency Timer has expired and a pending read or write request is detected.
- The master can negate the burst signal immediately after the Latency Timer has expired.

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

5.4.3 Dual Latency Timer Implementation

Most master devices perform read and write burst operations one after the other. Some masters, however, might find it advantageous to perform long read and write burst operations simultaneously.

In such instances, it is recommended that for optimal bus bandwidth, two separate latency timer and latency count registers are implemented. One timer and one count register must be implemented for each data bus. This allows totally independent operation of each bus for the highest possible throughput.

5.5 PLB Ordering and Coherence Requirements

This section outlines the ordering and coherence requirements for the PLB. Slaves must adhere to the following rules to ensure that their operation is compatible with master expectations for the ordering of read and write cycles.

If a PLB slave address-acknowledges a request, the PLB slave is obligated to perform the data transfers for overlapping addresses that are associated with this request, and any subsequently address-acknowledged request. The PLB slave must perform these data transfers in the order that the Sl_addrAck signal assertion dictates. The subsequent request can either be a burst or nonburst request.

For example, a write request is address acknowledged, then is followed by a read request with addresses that overlap the addresses that are associated with the write request. The read data acknowledgments that are associated with the subsequent read requests must provide the new data that is written by the write request. This requirement applies whether the subsequent read requests are burst or nonburst. Therefore, for this example of a write request followed by a read request, it is required that the write data acknowledgments be issued before, or at the same time as, the overlapping read data acknowledgments. This requirement exists so that the slave can provide the new write data to the read.

Conversely, if a read request is address acknowledged first, any read data acknowledgments that are associated with that request must be satisfied with old data. The read acknowledgments must not be supplied with data that is associated with any overlapping write request that is address acknowledged after the read request. This requirement applies whether the write request is burst or nonburst. However, for read requests that are followed by write requests, the PLB slave can issue the write data acknowledgments for the subsequent write request before it issues the read data acknowledgments for the earlier read requests. The PLB slave can only do this, though, if it buffers the write data and maintains the ability to supply the old data to the read request.

5.6 PLB Data Bus Extension

The base PLB architecture supports 32-bit read and write data buses; however, provisions exist for 64-bit, and 128-bit data buses. This extension allows 32-bit, 64-bit, and 128-bit masters and slaves to be connected to various size PLB implementations. Optimal bandwidth is achieved when both master and slave data buses are equal to the implemented bus width, but a protocol exists for all required conversion cycles between masters and slaves. For each increase in the PLB, byte enables operations are also extended to match the data bus width. To facilitate these transfers, each master port on the PLB has a master size, Mn_MSize(0:1), input that is steered to all PLB slaves during a master request. During the address acknowledgment phase of the transfer, the slave drives its slave size, Sl_SSize(0:1), input to the PLB, which is forwarded to the master. At this time both master and slave know each other’s size and the master determines if its current request can be accepted in one data phase or if it requires a subsequent conversion cycle to complete the requested operation. Conversion cycles are only required in the case of a larger master accessing a smaller slave with requested bytes on the unconnected portion of the data bus. For transfers initiated by smaller masters to larger slaves, no conversion cycles are necessary.

5.6.1 Data Steering

For greater than 32-bit PLB implementations, masters must mirror data for certain write operations. Also, slave steering of data is required for certain read operations. Master mirroring of data and slave data steering are illustrated in the following tables for various size masters.

Note: Master and slave designs do not need to support some or all other bus widths. It is obviously easier to implement a master or slave assuming it supports only one bus width. This extension is presented here as an interchangeability guideline. Because of performance, area, and development effort, some sizes might not be supported by a particular master or slave. It must decided when a master or slave is developed what the scope and future applicability of the device might be. Also, care must be taken when integrating existing cores to understand exactly which size devices it supports.

5.6.1.1 64-Bit Write Data Mirroring

Table 5-3 shows the data mirroring for 64-bit master write cycles. A 64-bit master is responsible for mirroring the data during write cycles when ABus(29) is a one. Because the master does not know what size the slave is until the PLB_addrAck signal and, potentially, the PLB_wrDAck signal, are asserted, the master must always mirror data when ABus(29) is a one. Masters making single write and multiple write, burst, or line requests must adhere to this requirement in order to support 32-bit slaves.

Table 5-3. 64-Bit Write Data Mirroring (Sheet 1 of 2)

ABus (29:31)	Bytes Enabled (0:7)	64-Bit Data Bus							
		Dbus 0:7 byte0	Dbus 8:15 byte1	Dbus 16:23 byte2	Dbus 24:31 byte3	Dbus 32:39 byte4	Dbus 40:47 byte5	Dbus 48:55 byte6	Dbus 56:63 byte7
000	1111_1111	byte0	byte1	byte2	byte3	byte4	byte5	byte6	byte7
000	1111_1110	byte0	byte1	byte2	byte3	byte4	byte5	byte6	
001	0111_1111		byte1	byte2	byte3	byte4	byte5	byte6	byte7
000	1111_1100	byte0	byte1	byte2	byte3	byte4	byte5		
001	0111_1110		byte1	byte2	byte3	byte4	byte5	byte6	
010	0011_1111			byte2	byte3	byte4	byte5	byte6	byte7
000	1111_1000	byte0	byte1	byte2	byte3	byte4			
001	0111_1100		byte1	byte2	byte3	byte4	byte5		
010	0011_1110			byte2	byte3	byte4	byte5	byte6	
011	0001_1111				byte3	byte4	byte5	byte6	byte7
000	1111_0000	byte0	byte1	byte2	byte3				
001	0111_1000		byte1	byte2	byte3	byte4			
010	0011_1100			byte2	byte3	byte4	byte5		
011	0001_1110				byte3	byte4	byte5	byte6	
100	0000_1111	byte4	byte5	byte6	byte7	byte4	byte5	byte6	byte7
000	1110_0000	byte0	byte1	byte2					
001	0111_0000		byte1	byte2	byte3				
010	0011_1000			byte2	byte3	byte4			
011	0001_1100				byte3	byte4	byte5		

Table 5-3. 64-Bit Write Data Mirroring (Sheet 2 of 2)

ABus (29:31)	Bytes Enabled (0:7)	64-Bit Data Bus							
		Dbus 0:7 byte0	Dbus 8:15 byte1	Dbus 16:23 byte2	Dbus 24:31 byte3	Dbus 32:39 byte4	Dbus 40:47 byte5	Dbus 48:55 byte6	Dbus 56:63 byte7
100	0000_1110	byte4	byte5	byte6		byte4	byte5	byte6	
101	0000_0111		byte5	byte6	byte7		byte5	byte6	byte7
000	1100_0000	byte0	byte1						
001	0110_0000		byte1	byte2					
010	0011_0000			byte2	byte3				
011	0001_1000				byte3	byte4			
100	0000_1100	byte4	byte5			byte4	byte5		
101	0000_0110		byte5	byte6			byte5	byte6	
110	0000_0011			byte6	byte7			byte6	byte7
000	1000_0000	byte0							
001	0100_0000		byte1						
010	0010_0000			byte2					
011	0001_0000				byte3				
100	0000_1000	byte4				byte4			
101	0000_0100		byte5				byte5		
110	0000_0010			byte6				byte6	
111	0000_0001				byte7				byte7

5.6.1.2 128-Bit Write Data Mirroring

Table 5-4 shows the data mirroring for 128-bit master write cycles. A 128-bit master that supports 32-bit and 64-bit slaves is responsible for mirroring the data during write cycles when ABus(28) or ABus(29) is a one. Because the master does not know what size the slave is until the PLB_addrAck signal and, potentially, the PLB_wrDAck signal are asserted, the master must always mirror data when ABus(28) or ABus(29) is a one. In rows 1 4 of the table, “O” means data is optionally driven, depending on the number of bytes in the write transfer. Masters making single write and multiple write, burst, or line requests must adhere to this requirement to support 64-bit and 32-bit slaves.

Table 5-4. 128-Bit Write Data Mirroring (Sheet 1 of 3)

ABus (28:31)	128-Bit Data Bus															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0100	4	5	6	7	4	5	6	7	O	O	O	O	O	O	O	O
0101		5	6	7		5	6	7	O	O	O	O	O	O	O	O
0110			6	7			6	7	O	O	O	O	O	O	O	O
0111				7				7	O	O	O	O	O	O	O	O
1000	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
1000	8	9	10	11	12	13	14		8	9	10	11	12	13	14	

Table 5-4. 128-Bit Write Data Mirroring (Sheet 2 of 3)

ABus (28:31)	128-Bit Data Bus															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1001		9	10	11	12	13	14	15		9	10	11	12	13	14	15
1000	8	9	10	11	12	13			8	9	10	11	12	13		
1001		9	10	11	12	13	14			9	10	11	12	13	14	
1010			10	11	12	13	14	15			10	11	12	13	14	15
1000	8	9	10	11	12				8	9	10	11	12			
1001		9	10	11	12	13				9	10	11	12	13		
1010			10	11	12	13	14				10	11	12	13	14	
1011				11	12	13	14	15				11	12	13	14	15
1000	8	9	10	11					8	9	10	11				
1001		9	10	11	12					9	10	11	12			
1010			10	11	12	13					10	11	12	13		
1011				11	12	13	14					11	12	13	14	
1100	12	13	14	15	12	13	14	15					12	13	14	15
0100	4	5	6		4	5	6									
1000	8	9	10						8	9	10					
1001		9	10	11						9	10	11				
1010			10	11	12						10	11	12			
1011				11	12	13						11	12	13		
1100	12	13	14		12	13	14						12	13	14	
1101		13	14	15		13	14	15						13	14	15
0100	4	5			4	5										
0101		5	6			5	6									
1000	8	9							8	9						
1001		9	10							9	10					
1010			10	11							10	11				
1011				11	12							11	12			
1100	12	13			12	13							12	13		
1101		13	14			13	14							13	14	
1110			14	15			14	15							14	15
0100	4				4											
0101		5				5										
0110			6				6									
1000	8								8							
1001		9								9						
1010			10								10					
1011				11								11				

Table 5-4. 128-Bit Write Data Mirroring (Sheet 3 of 3)

ABus (28:31)	128-Bit Data Bus															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1100	12				12								12			
1101		13				13								13		
1110			14				14								14	
1111				15				15								15

5.6.1.3 64-Bit Read Data Steering

Table 5-5 shows the data steering for 64-bit slave read cycles by a 32-bit master. For read cycles by a 64-bit master, no data steering is required. A 64-bit slave is responsible for steering the data during partial read cycles when ABus (29) is a one and PLB_MSize(0:1) is sampled indicating a 32-bit master. During 32-bit master accesses on a 64-bit PLB implementation, Mn_BE(0:7) might appear noncontiguous to a 64-bit slave. To conserve power consumption, it is recommended that only the appropriate bytes, bytes 0 3, must be switched during these types of cycles.

Table 5-5. 64-Bit Slave Read Steering to a 32-Bit Master

ABus (29:31)	Mn_BE (0:7)	64-Bit Data Bus							
		Dbus 0:7 byte0	Dbus 8:15 byte1	Dbus 16:23 byte2	Dbus 24:31 byte3	Dbus 32:39 byte4	Dbus 40:47 byte5	Dbus 48:55 byte6	Dbus 56:63 byte7
000	1111_1111	byte0	byte1	byte2	byte3				
100	1111_1111	byte4	byte5	byte6	byte7				
000	1110_1110	byte0	byte1	byte2					
001	0111_0111		byte1	byte2	byte3				
100	1110_1110	byte4	byte5	byte6					
101	0111_0111		byte5	byte6	byte7				
000	1100_1100	byte0	byte1						
001	0110_0110		byte1	byte2					
010	0011_0011			byte2	byte3				
100	1100_1100	byte4	byte5						
101	0110_0110		byte5	byte6					
110	0011_0011			byte6	byte7				
000	1000_1000	byte0							
001	0100_0100		byte1						
010	0010_0010			byte2					
011	0001_0001				byte3				
100	1000_1000	byte4							
101	0100_0100		byte5						
110	0010_0010			byte6					
111	0001_0001				byte7				

5.6.1.4 128-Bit Read Data Steering to a 32-Bit Master

Table 5-6 describes 128-bit slave steering to a 32-bit master.

Table 5-6. 128-Bit Slave Steering to a 32-Bit Master

ABus (28:31)	128-Bit Data Bus															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0100	4	5	6	7												
1000	8	9	10	11												
1100	12	13	14	15												
0100	4	5	6													
0101		5	6	7												
1000	8	9	10													
1001		9	10	11												
1100	12	13	14													
1101		13	14	15												
0100	4	5														
0101		5	6													
0110			6	7												
1000	8	9														
1001		9	10													
1010			10	11												
1100	12	13														
1101		13	14													
1110			14	15												
0100	4															
0101		5														
0110			6													
0111				7												
1000	8															
1001		9														
1010			10													
1011				11												
1100	12															
1101		13														
1110			14													
1111				15												

5.6.1.5 128-Bit Slave Steering to a 64-Bit Master

Table 5-7 describes 128-bit slave steering to a 64-bit master.

Table 5-7. 128-Bit Slave Steering to a 64-Bit Master (Sheet 1 of 2)

ABus (28:31)	128-Bit Data Bus															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1000	8	9	10	11	12	13	14	15								
1000	8	9	10	11	12	13	14									
1001		9	10	11	12	13	14	15								
1000	8	9	10	11	12	13										
1001		9	10	11	12	13	14									
1010			10	11	12	13	14	15								
1000	8	9	10	11	12											
1001		9	10	11	12	13										
1010			10	11	12	13	14									
1011				11	12	13	14	15								
1000	8	9	10	11												
1001		9	10	11	12											
1010			10	11	12	13										
1011				11	12	13	14									
1100					12	13	14	15								
1000	8	9	10													
1001		9	10	11												
1010			10	11	12											
1011				11	12	13										
1100					12	13	14									
1101						13	14	15								
1000	8	9														
1001		9	10													
1010			10	11												
1011				11	12											
1100					12	13										
1101						13	14									
1110							14	15								
1000	8															
1001		9														
1010			10													
1011				11												
1100					12											

Table 5-7. 128-Bit Slave Steering to a 64-Bit Master (Sheet 2 of 2)

ABus (28:31)	128-Bit Data Bus															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1101						13										
1110							14									
1111								15								

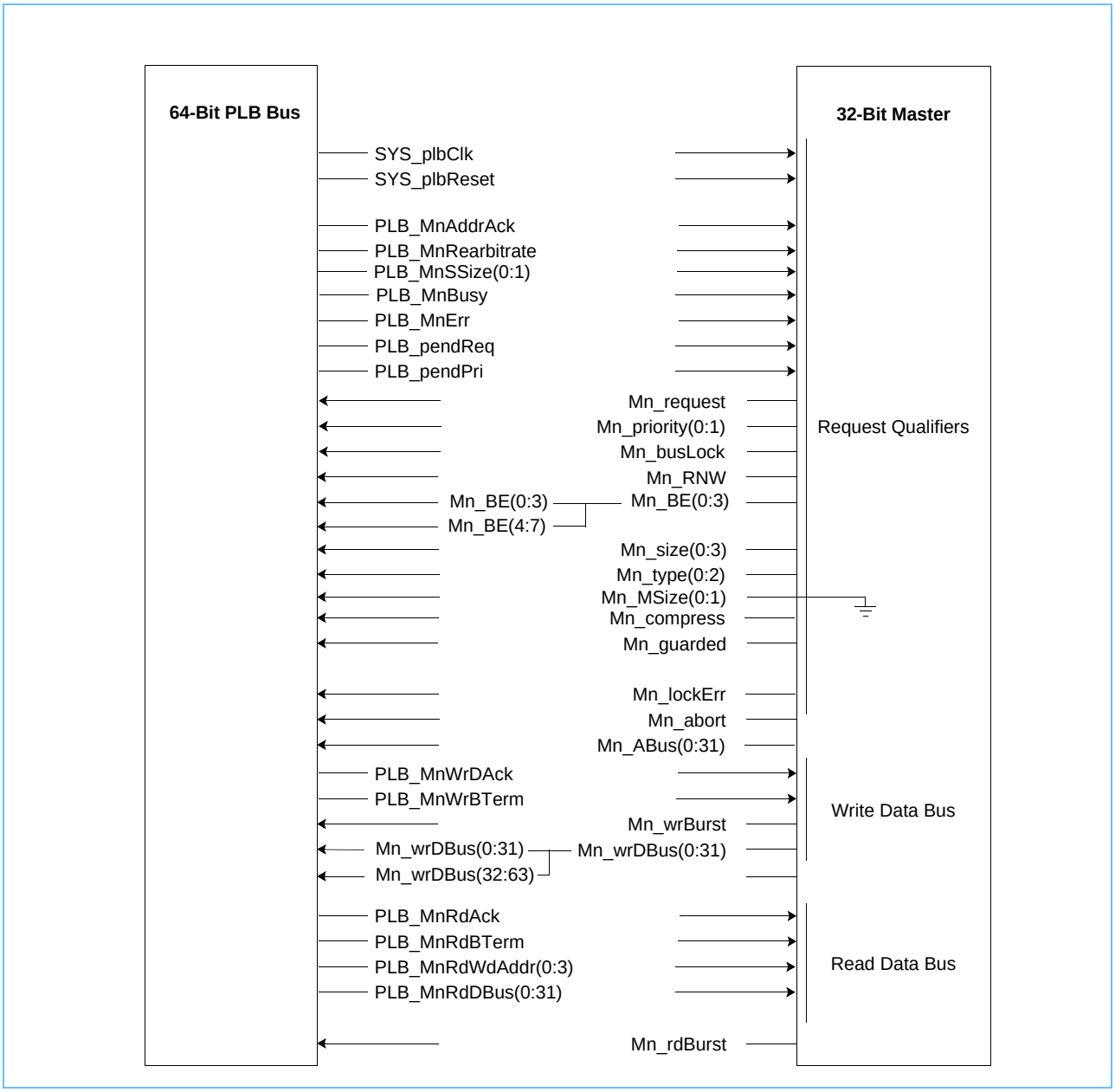
5.6.2 Connecting 32-Bit Devices to a 64-Bit PLB

The connection of 32-bit devices to a 64-bit PLB implementation is relatively straightforward. The following diagrams show the interconnection required.

5.6.2.1 32-Bit Master Interface to 64-Bit PLB

Figure 5-32 32-Bit Master Interface to 64-Bit PLB on page 128 demonstrates the connection of a 32-bit master to a 64-bit PLB implementation. The byte enables, Mn_BE(0:3), are mirrored to both the lower and upper 4-bits of the of the 8-bit PLB master byte enable port. The master write-data bus, Mn_wrDBus(0:31), is mirrored to both the lower and upper 32-bits of the of the 64-bit PLB master write-data bus-port. The read data bus is connected to the lower 32-bits of the 64-bit data bus, PLB_MnRdDBus(0:31), and the higher 32-bits of the read data bus are not connected. Both Mn_MSize(0:1) PLB inputs are tied inactive.

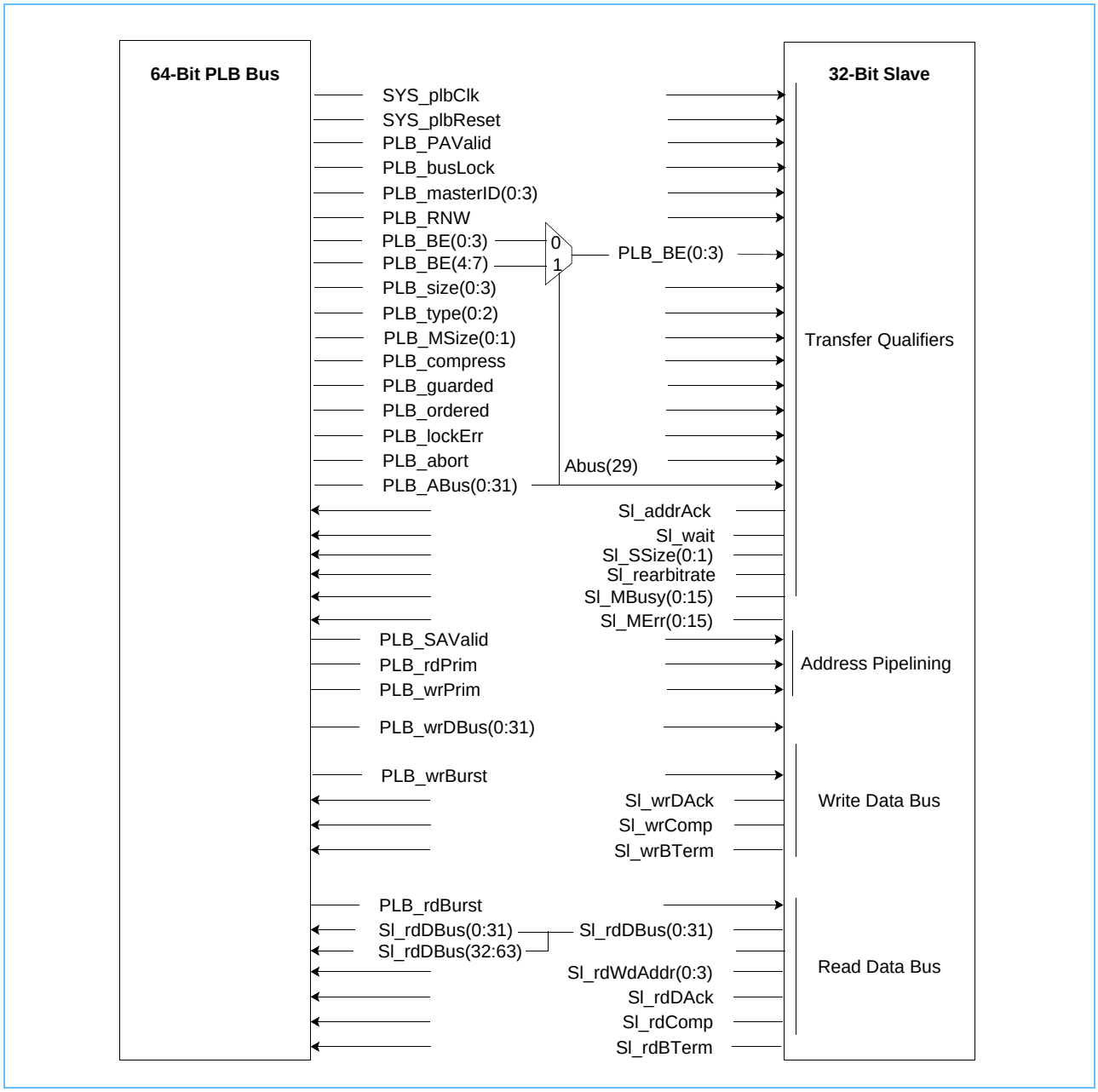
Figure 5-32. 32-Bit Master Interface to 64-Bit PLB



5.6.2.2 32-Bit Slave Interface to 64-Bit PLB

Figure 5-33 demonstrates the connection of a 32-bit slave to a 64-bit PLB implementation. The lower and upper 4 bits of the 8-bit byte enables, PLB_BE(0:7), are fed into a 4-bit two-to-one multiplexer with PLB_ABus(29) used as the select signal. This arrangement causes the slave to sample the lower four byte enables during accesses to data in the lower portion of the data bus and the upper four byte enables to accesses in the upper half of the bus. The slave write data bus connects to the lower 32 bits of the write data bus, PLB_wrDBus(0:31). The read data bus is mirrored to both the lower and upper 32 bits of the 64-bit PLB slave data bus port, Sl_rdDBus(0:63).

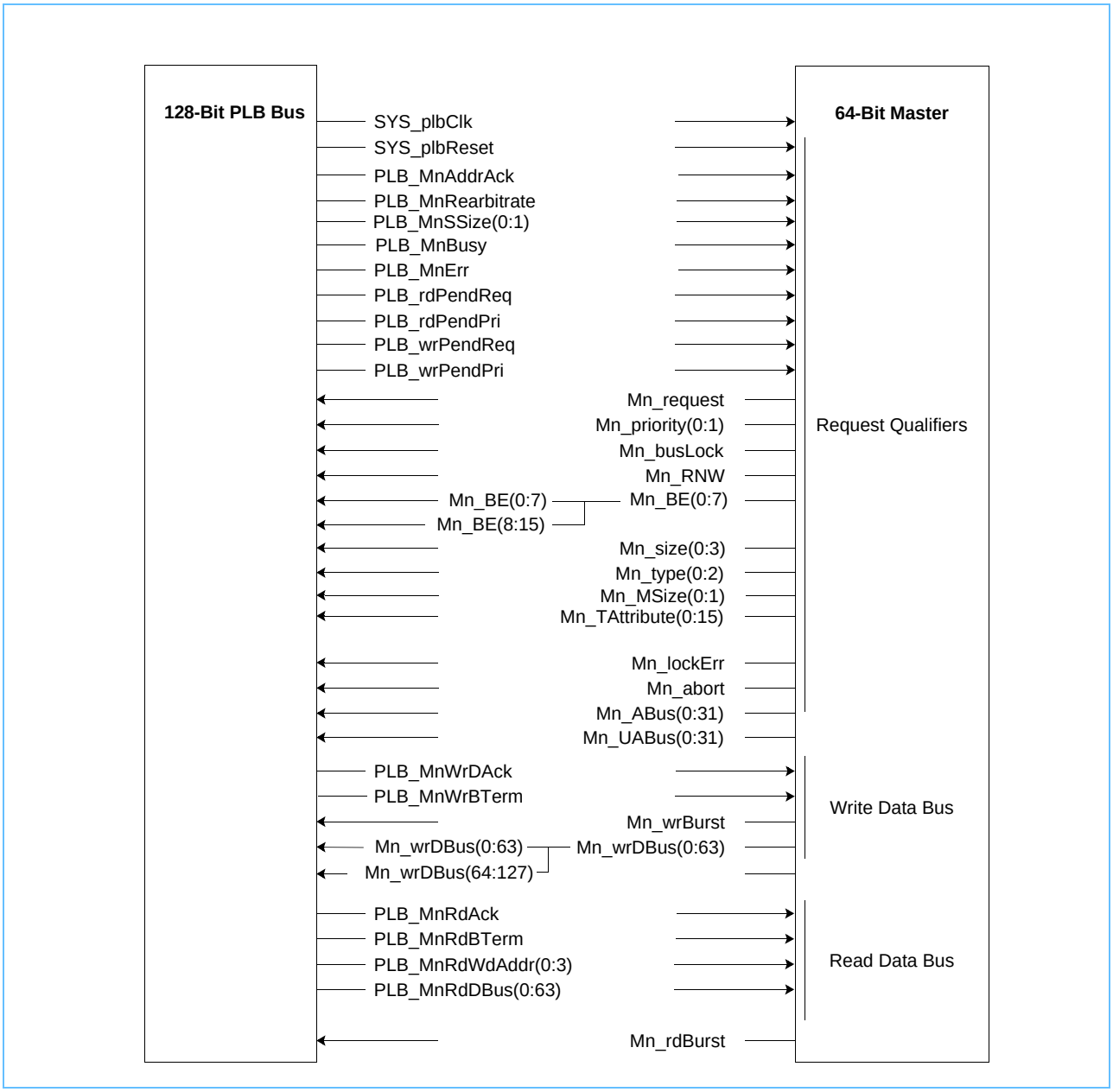
Figure 5-33. 32-Bit Slave Interface to 64-Bit PLB



5.6.2.3 64-Bit Master Interface to 128-Bit PLB

Figure 5-34 demonstrates the connection of a 64-bit master to a 128-bit PLB implementation. The byte enables, Mn_BE(0:7), are mirrored to both the lower and upper 8-bits of the 16-bit PLB master byte enable port. The master write data bus, Mn_wrDBus(0:63), is mirrored to both the lower and upper 64 bits of the of the 128-bit PLB master write data bus port. The read data bus is connected to the lower order 64 bits of the 128-bit data bus, PLB_MnRdDBus(0:63). The higher 64 bits of the read data bus are not connected. Mn_MSize(0) PLB input is tied inactive, and Mn_MSize(1) PLB input is tied active.

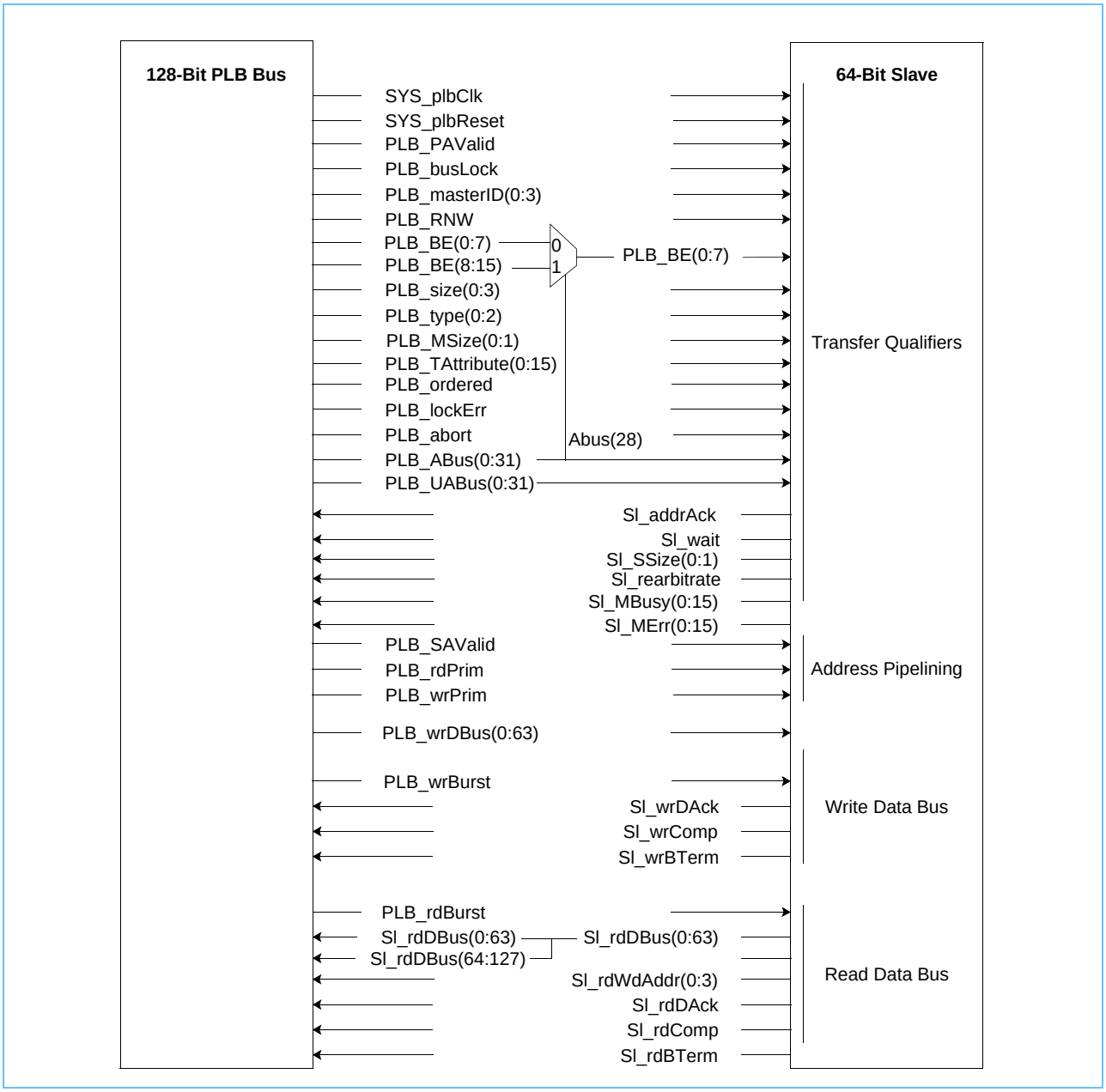
Figure 5-34. 64-Bit Master Interface to 128-Bit PLB



5.6.2.4 64-Bit Slave Interface to 128-Bit PLB

Figure 5-35 demonstrates the connection of a 64-bit slave to a 128-bit PLB implementation. The lower and upper 8 bits of the 16-bit byte enables, PLB_BE(0:15), are fed into an 8-bit two-to-one multiplexer with PLB_ABus(28) used as the select signal. This arrangement causes the slave to sample the lower eight byte enables during accesses to data in the lower portion of the data bus and the upper four byte enables to accesses in the upper half of the bus. The slave write data bus connects to the lower 64 bits of the write data bus, PLB_wrDBus(0:63). The read data bus is mirrored to both the lower and upper 64 bits of the 128-bit PLB slave data bus port, Sl_rdDBus(0:127).

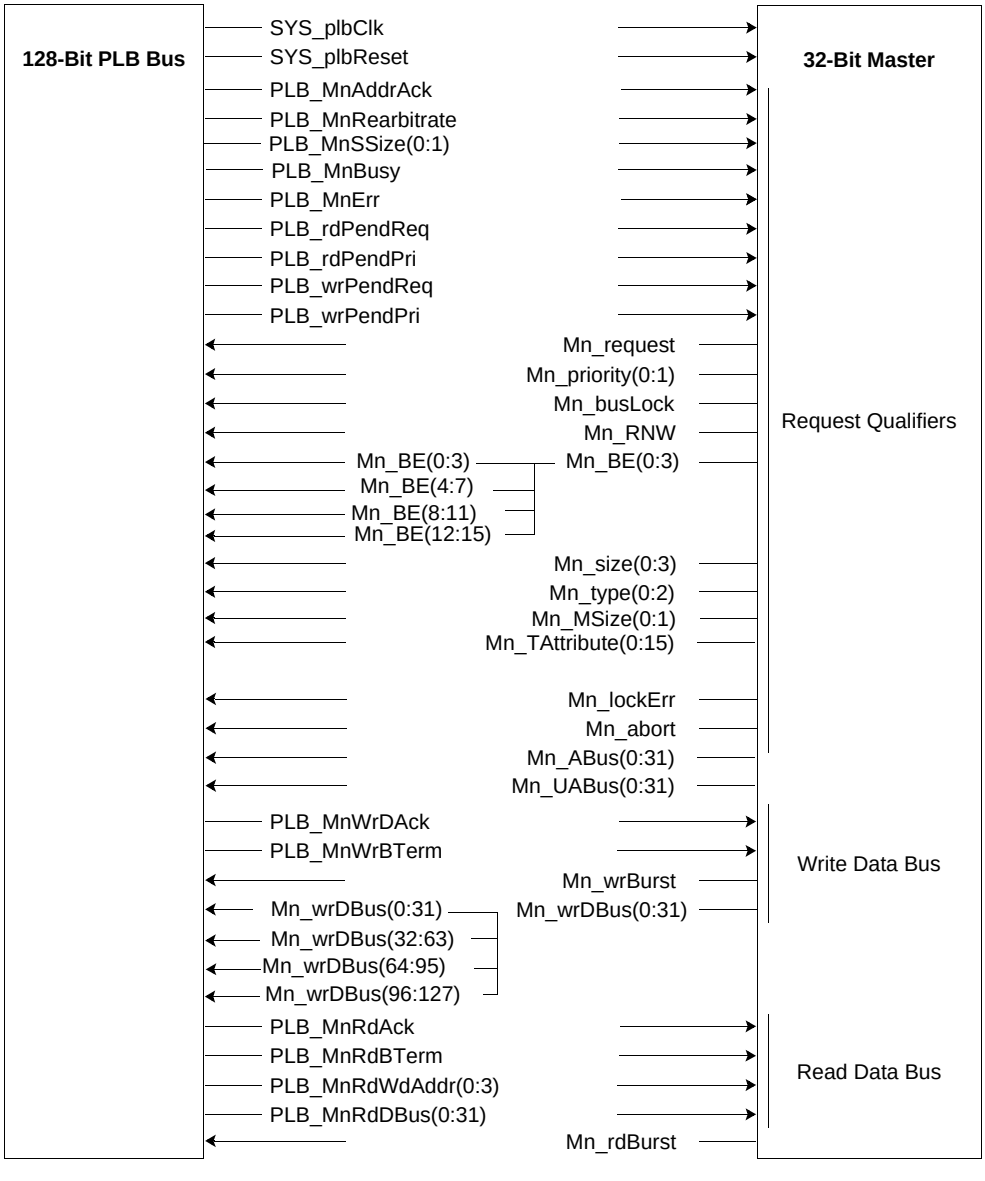
Figure 5-35. 64-Bit Slave Interface to 128-Bit PLB



5.6.2.5 32-Bit Master Interface to 128-Bit PLB

Figure 5-36 demonstrates the connection of a 32-bit master to a 128-bit PLB implementation. The byte enables, Mn_BE(0:3), are mirrored to Mn_BE(4:7), Mn_BE(8:11), and Mn_BE(12:15) of the 16-bit PLB master byte enable port. The master write data bus, Mn_wrDBus(0:31), is mirrored to Mn_wrDBus(32:63), Mn_wrDBus(64:95), Mn_wrDBus(96:127) of the 128-bit PLB master write data bus port. The read data bus is connected to the lower order 32 bits of the 128-bit data bus, PLB_MnRdDBus(0:31), and the higher 96 bits of the read data bus are not connected. Mn_MSize(0:1) is tied inactive.

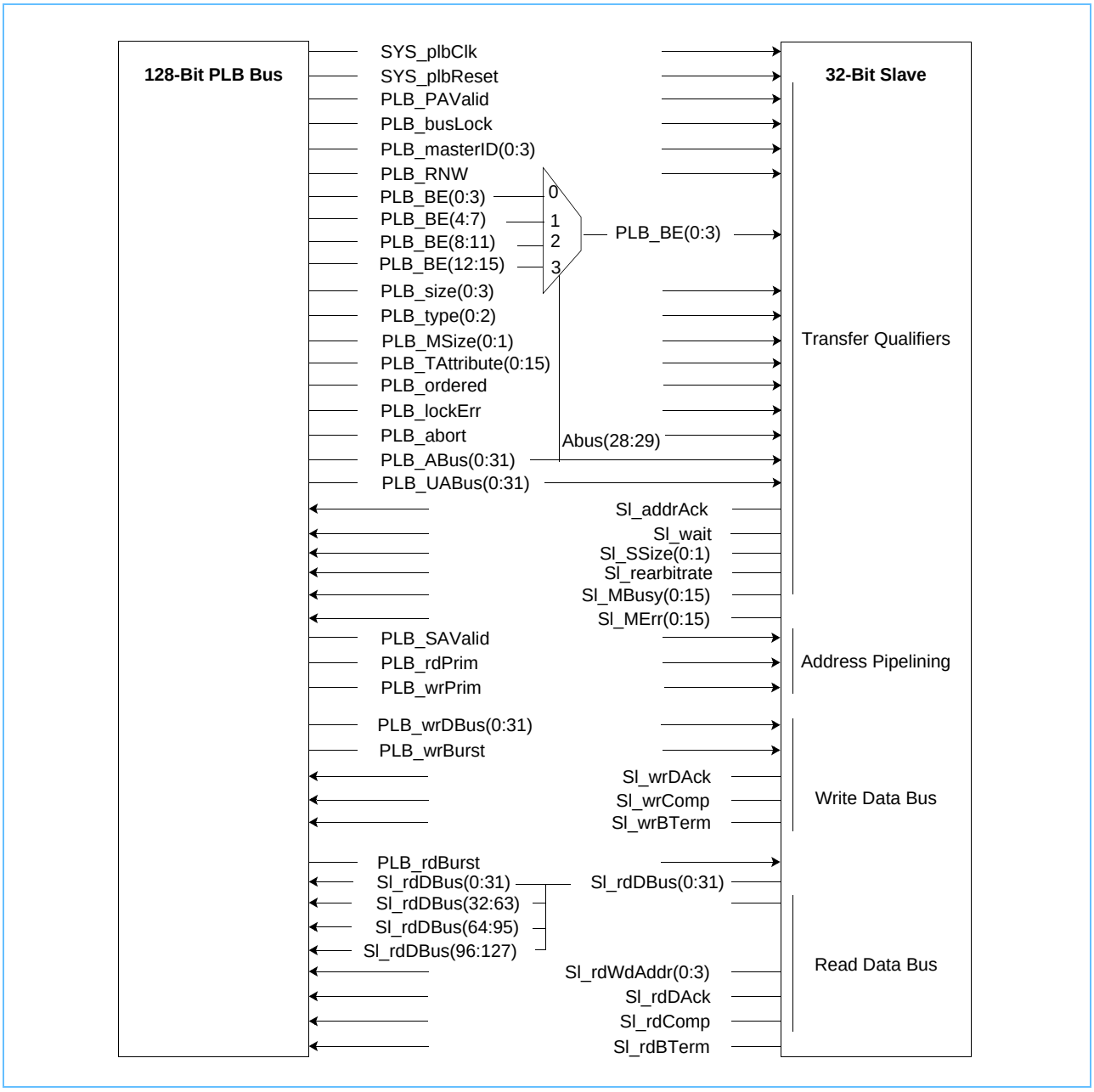
Figure 5-36. 32-Bit Master Interface to 128-Bit PLB



5.6.2.6 32-Bit Slave Interface to 128-Bit PLB

Figure 5-37 demonstrates the connection of a 32-bit slave to a 128-bit PLB implementation. PLB_BE(0:3), PLB_BE(4:7), PLB_BE(8:11), PLB_BE(12:15) are fed into a 4-bit four-to-one multiplexer with PLB_ABus(28:29) used as the select signal. This arrangement causes the slave to sample the appropriate four byte enables during accesses. The slave write data bus connects to the lower 32 bits of the write data bus, PLB_wrDBus(0:31). The read data bus is mirrored to Sl_rdDBus(0:31), Sl_rdDBus(32:63), Sl_rdDBus(64:95), and Sl_rdDBus(96:127).

Figure 5-37. 32-Bit Slave Interface to 128-Bit PLB



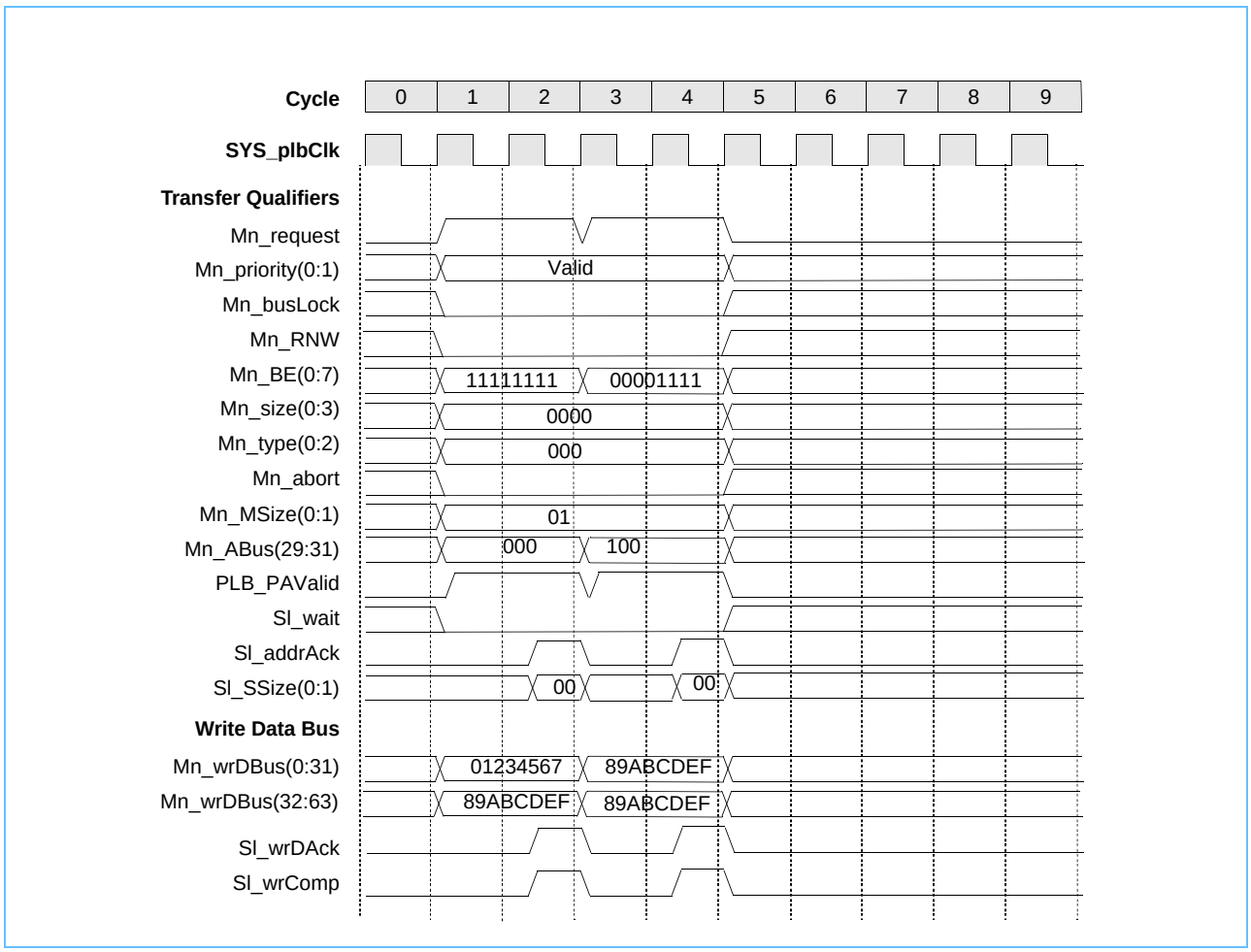
5.6.3 64-Bit Master to 32-Bit Conversion Cycles

During the address acknowledgment phase of a transfer, the slave drives its slave size, Sl_SSize(0:1), input to the PLB, which is forwarded to the master. At this time, both master and slave know each other's size and the master determines if its current request will be accepted in one data phase or require a subsequent conversion cycle to complete the requested operation. Conversion cycles are only required in the case of a 64-bit master accessing a 32-bit slave with requested bytes on both the lower and upper 32 bits of the 64-bit data bus. For 32-bit masters or 64-bit slaves, no conversion cycles are necessary.

5.6.3.1 64-Write Conversion Cycle

Figure 5-38 shows the operation of a 64-bit write transfer to a 32-bit slave on the PLB. The master must perform a conversion cycle because the slave size that is sampled with the Sl_addrAck signal indicates that the slave can only accept 32 bits of data. The master makes a second request in the following cycle and updates the Mn_BE, Mn_ABus, and Mn_wrDBus signals for the conversion cycle. The master must mirror the write data during the conversion cycle.

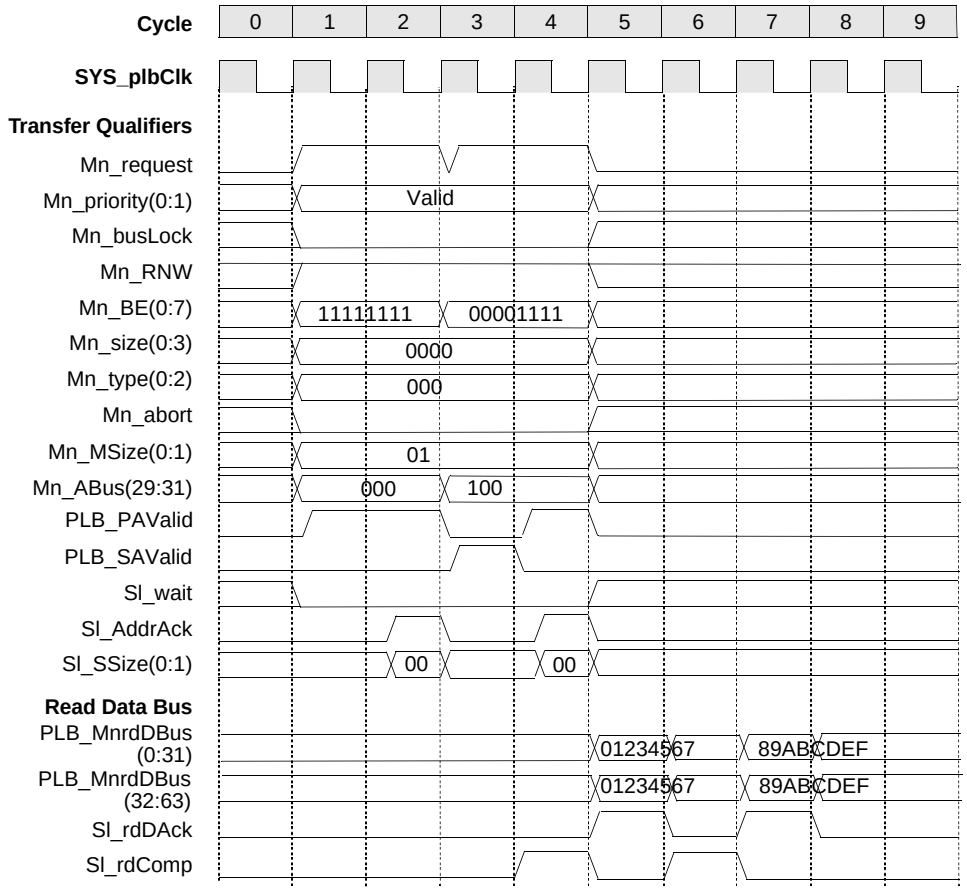
Figure 5-38. 64-Bit Write Conversion Cycle



5.6.3.2 64-Bit Read Conversion Cycle

Figure 5-39 shows the operation of a 64-bit read transfer from a 32-bit slave on the PLB. The master must perform a conversion cycle because the slave size that is sampled with the Sl_addrAck signal indicates that the slave can only provide 32 bits of data. The master makes a second request in the following cycle and updates the Mn_BE and Mn_ABus signals for the conversion cycle. The conversion cycle in this case causes a secondary pipelined read. The 32-bit slave drives data to the lower and upper words of the data bus.

Figure 5-39. 64-Bit Read Conversion Cycle



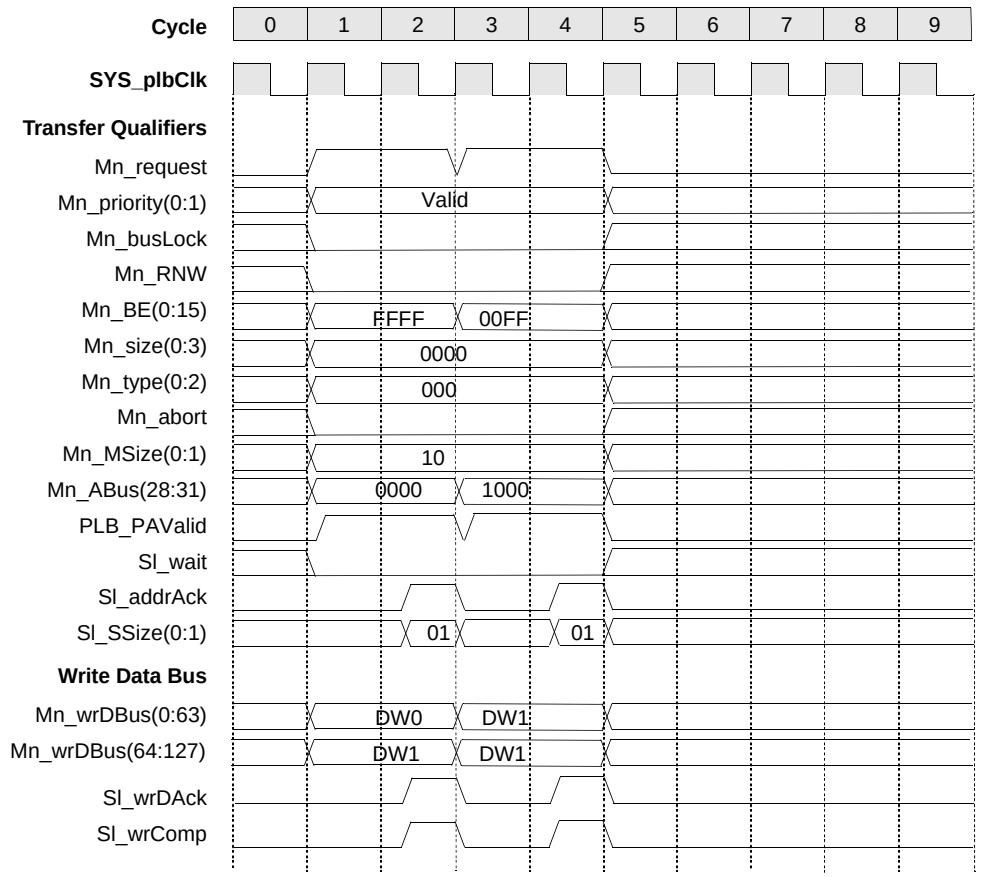
5.6.4 128-Bit Master to 64-Bit Slave Conversion Cycles

During the address acknowledgment phase of a transfer, the slave drives its slave size, Sl_SSize(0:1), input to the PLB, which is forwarded to the master. At this time, both master and slave know each other's size. The master determines if its current request will be accepted in one data phase or require a subsequent conversion cycle to complete the requested operation. Conversion cycles are required in the case of a 128-bit master accessing a 64-bit slave with requested bytes on both the lower and upper 64-bits of the 128-bit data bus. For 32-bit masters, 64-bit masters, or 128-bit slaves, no conversion cycles are necessary.

5.6.4.1 64-Bit Write Conversion Cycle

Figure 5-40 shows the operation of a 128-bit write transfer to a 64-bit slave on the PLB. The master must perform a conversion cycle because the slave size that is sampled with the Sl_addrAck signal indicates that the slave can only accept 64 bits of data. The master makes a second request in the following cycle and updates the Mn_BE, Mn_ABus, and Mn_wrDBus signals for the conversion cycle. The master must mirror the write data during the conversion cycle.

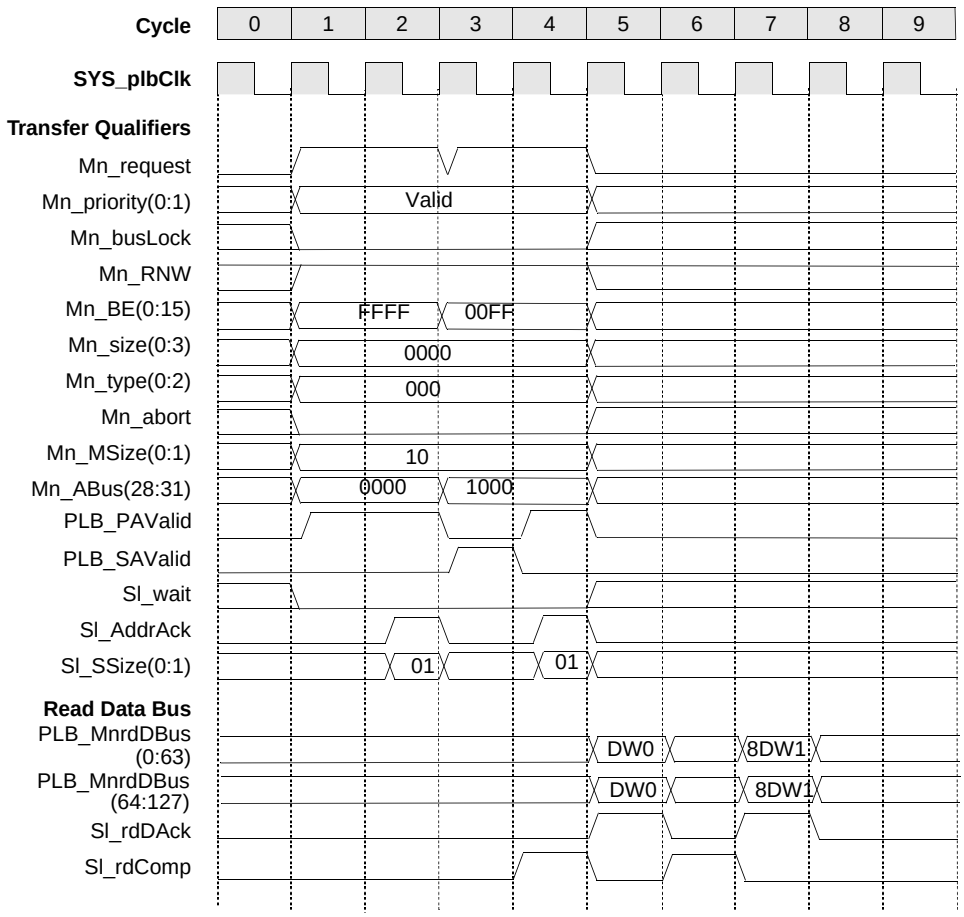
Figure 5-40. 128-Bit Write Conversion Cycle



5.6.4.2 12-Bit Read Conversion Cycle

Figure 5-41 shows the operation of a 128-bit read transfer from a 64-bit slave on the PLB. The master must perform a conversion cycle because the slave size that is sampled with the Sl_addrAck signal indicates that the slave can only provide 64 bits of data. The master makes a second request in the following cycle and updates the Mn_BE and Mn_ABus signals for the conversion cycle. The conversion cycle in this case causes a secondary pipelined read. The 64-bit slave drives data to the lower and upper doublewords of the data bus.

Figure 5-41. 128-Bit Read Conversion Cycle



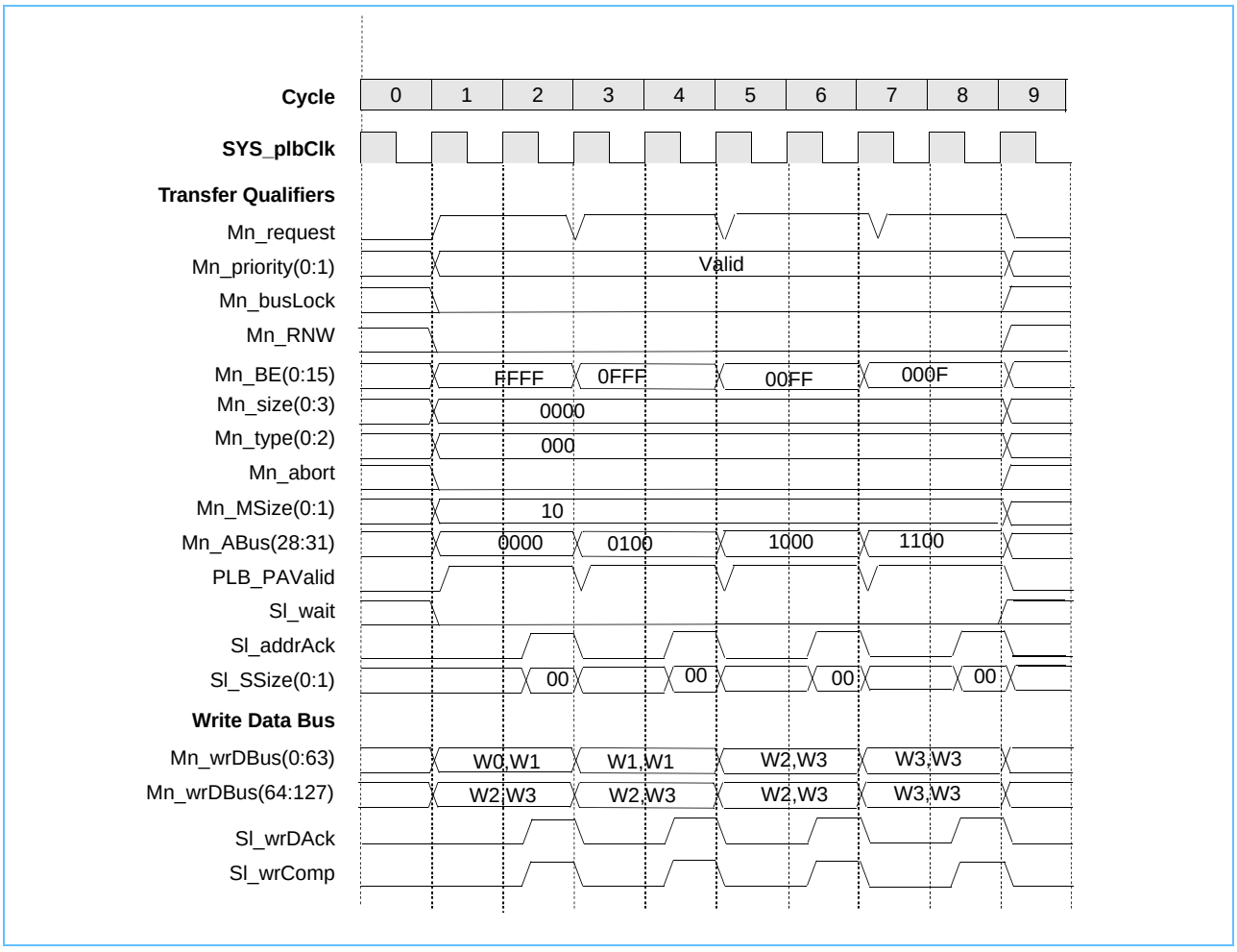
5.6.5 128-Bit Master to 32-Bit Slave Multiple Conversion Cycles

During the address acknowledgment phase of a transfer, the slave drives its slave size, Sl_SSize(0:1), input to the PLB, which is forwarded to the master. At this time both master and slave know each others size. The master determines if its current request will be accepted in one data phase or require a subsequent conversion cycle to complete the requested operation. Conversion cycles are required in the case of a 128-bit master accessing a 32-bit slave with requested bytes crossing any 4-byte boundary of the 128-bit data bus. For 32-bit masters or 128-bit slaves, no conversion cycles are necessary.

5.6.5.1 128-Bit Multiple Write Conversion Cycle

Figure 5-42 shows the operation of a 128-bit write transfer to a 32-bit slave on the PLB. The master must perform a multiple conversion cycle because the slave size that is sampled with the Sl_addrAck signal indicates that the slave can only accept 32 bits of data. The master makes a second request in the following cycle and updates the Mn_BE, Mn_ABus, and Mn_wrDBus signals for the conversion cycle. This sequence continues for the third and fourth transfers. The master must mirror the write data to Mn_wrDBus(0:31) during the conversion cycle.

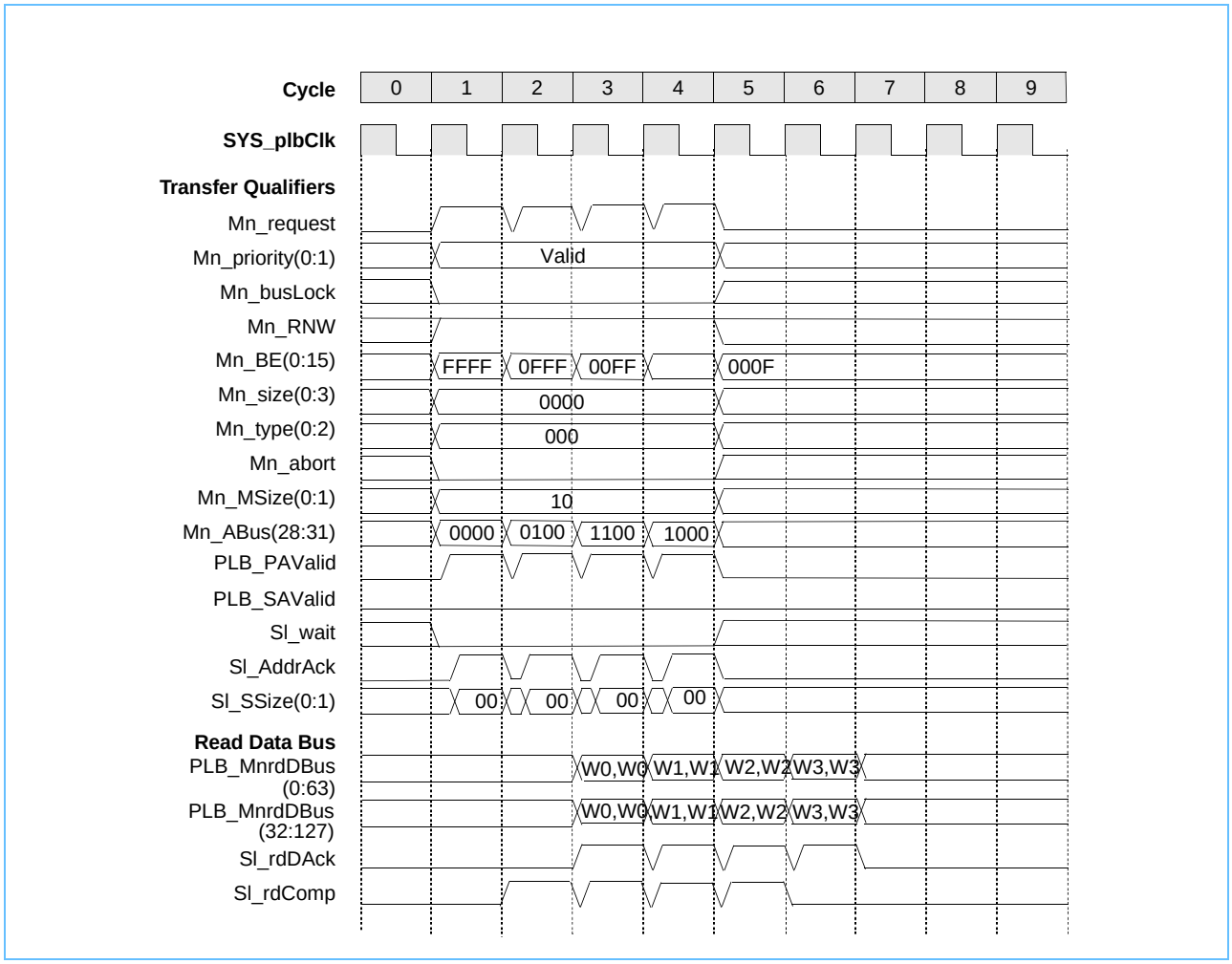
Figure 5-42. 128-Bit Multiple Write Conversion Cycle



5.6.5.2 128-Bit Multiple Read Conversion Cycle

Figure 5-43 shows the operation of a 128-bit read transfer from a 32-bit slave on the PLB. The master must perform a multiple conversion cycle because the slave size that is sampled with the Sl_addrAck signal indicates that the slave can only provide 32 bits of data. The master makes a second request in the following cycle and updates the Mn_BE and Mn_ABus signals for the conversion cycle. This sequence continues for the third and fourth transfers. The 32-bit slave mirrors data to the all words of the data bus.

Figure 5-43. 128-Bit Multiple Read Conversion Cycle



5.6.6 64-Bit Conversion Cycle Byte Enables

Conversion cycles are only required in the case of a 64-bit master accessing a 32-bit slave with requested bytes on both the lower and upper 32 bits of the 64-bit data bus. *Table 5-8* shows the byte enables driven by the master during a conversion cycle. Note that Mn_ABus(29:31) for a conversion cycle is always '100'.

Table 5-8. Byte Enables for Conversion Cycles (Sheet 1 of 2)

Current Cycle Mn_ABus(29:31)	Current Cycle Mn_BE(0:7)	Conversion Cycle Mn_BE(0:7)	Transfer Size (Bytes)
000	1111_1111	0000_1111	8
000	1111_1110	0000_1110	7
001	0111_1111	0000_1111	7
000	1111_1100	0000_1100	6
001	0111_1110	0000_1110	6
010	0011_1111	0000_1111	6
000	1111_1000	0000_1000	5
001	0111_1100	0000_1100	5
010	0011_1110	0000_1110	5
011	0001_1111	0000_1111	5
000	1111_0000	No Conversion	4
001	0111_1000	0000_1000	4
010	0011_1100	0000_1100	4
011	0001_1110	0000_1110	4
100	0000_1111	No Conversion	4
000	1110_0000	No Conversion	3
001	0111_0000	No Conversion	3
010	0011_1000	0000_1000	3
011	0001_1100	0000_1100	3
100	0000_1110	No Conversion	3
101	0000_0111	No Conversion	3
000	1100_0000	No Conversion	2
001	0110_0000	No Conversion	2
010	0011_0000	No Conversion	2
011	0001_1000	0000_1000	2
100	0000_1100	No Conversion	2
101	0000_0110	No Conversion	2
110	0000_0011	No Conversion	2
000	1000_0000	No Conversion	1
001	0100_0000	No Conversion	1
010	0010_0000	No Conversion	1
011	0001_0000	No Conversion	1

Table 5-8. Byte Enables for Conversion Cycles (Sheet 2 of 2)

Current Cycle Mn_ABus(29:31)	Current Cycle Mn_BE(0:7)	Conversion Cycle Mn_BE(0:7)	Transfer Size (Bytes)
100	0000_1000	No Conversion	1
101	0000_0100	No Conversion	1
110	0000_0010	No Conversion	1
111	0000_0001	No Conversion	1

5.6.7 128-Bit Conversion Cycle Byte Enables

Conversion cycles are only required in the case of a 128-bit master accessing a 64-bit slave with requested bytes on both the lower and upper 64 bits of the 128-bit data bus. *Table 5-9* shows the byte enables driven by the master during a conversion cycle. Note that Mn_ABus(29:31) for a conversion cycle is always ‘100’.

Table 5-9. Byte Enables for 128-Bit Conversion Cycles (Sheet 1 of 4)

Current Cycle Mn_ABus(28:31)	Current Cycle Mn_BE(0:15)	Conversion Cycle for 64-Bit Slave Mn_BE(0:15)	Conversion Cycle for 32-Bit Slave Mn_BE(0:15)	Transfer Size (Bytes)
0000	1111_1111_1111_1111	0000_0000_1111_1111	0000_1111_1111_1111	16
0000	1111_1111_1111_1110	0000_0000_1111_1110	0000_1111_1111_1110	15
0001	0111_1111_1111_1111	0000_0000_1111_1111	0000_1111_1111_1111	15
0000	1111_1111_1111_1100	0000_0000_1111_1100	0000_1111_1111_1100	14
0001	0111_1111_1111_1110	0000_0000_1111_1110	0000_1111_1111_1110	14
0010	0011_1111_1111_1111	0000_0000_1111_1111	0000_1111_1111_1111	14
0000	1111_1111_1111_1000	0000_0000_1111_1000	0000_1111_1111_1000	13
0001	0111_1111_1111_1100	0000_0000_1111_1100	0000_1111_1111_1100	13
0010	0011_1111_1111_1110	0000_0000_1111_1110	0000_1111_1111_1110	13
0011	0001_1111_1111_1111	0000_0000_1111_1111	0000_1111_1111_1111	13
0000	1111_1111_1111_0000	0000_0000_1111_0000	0000_1111_1111_0000	12
0001	0111_1111_1111_1000	0000_0000_1111_1000	0000_1111_1111_1000	12
0010	0011_1111_1111_1100	0000_0000_1111_1100	0000_1111_1111_1100	12
0011	0001_1111_1111_1110	0000_0000_1111_1110	0000_1111_1111_1110	12
0100	0000_1111_1111_1111	0000_0000_1111_1111	0000_0000_1111_1111	12
0000	1111_1111_1110_0000	0000_0000_1110_0000	0000_1111_1110_0000	11
0001	0111_1111_1111_0000	0000_0000_1111_0000	0000_1111_1111_0000	11
0010	0011_1111_1111_1000	0000_0000_1111_1000	0000_1111_1111_1000	11
0011	0001_1111_1111_1100	0000_0000_1111_1100	0000_1111_1111_1100	11
0100	0000_1111_1111_1110	0000_0000_1111_1110	0000_0000_1111_1110	11
0101	0000_0111_1111_1111	0000_0000_1111_1111	0000_0000_1111_1111	11
0000	1111_1111_1100_0000	0000_0000_1100_0000	0000_1111_1100_0000	10
0001	0111_1111_1110_0000	0000_0000_1110_0000	0000_1111_1110_0000	10

Table 5-9. Byte Enables for 128-Bit Conversion Cycles (Sheet 2 of 4)

Current Cycle Mn_ABus(28:31)	Current Cycle Mn_BE(0:15)	Conversion Cycle for 64-Bit Slave Mn_BE(0:15)	Conversion Cycle for 32-Bit Slave Mn_BE(0:15)	Transfer Size (Bytes)
0010	0011_1111_1111_0000	0000_0000_1111_0000	0000_1111_1111_0000	10
0011	0001_1111_1111_1000	0000_0000_1111_1000	0000_1111_1111_1000	10
0100	0000_1111_1111_1100	0000_0000_1111_1100	0000_0000_1111_1100	10
0101	0000_0111_1111_1110	0000_0000_1111_1110	0000_0000_1111_1110	10
0110	0000_0011_1111_1111	0000_0000_1111_1111	0000_0000_1111_1111	10
0000	1111_1111_1000_0000	0000_0000_1000_0000	0000_1111_1000_0000	9
0001	0111_1111_1100_0000	0000_0000_1100_0000	0000_1111_1100_0000	9
0010	0011_1111_1110_0000	0000_0000_1110_0000	0000_1111_1110_0000	9
0011	0001_1111_1111_0000	0000_0000_1111_0000	0000_1111_1111_0000	9
0100	0000_1111_1111_1000	0000_0000_1111_1000	0000_0000_1111_1000	9
0101	0000_0111_1111_1100	0000_0000_1111_1100	0000_0000_1111_1100	9
0110	0000_0011_1111_1110	0000_0000_1111_1110	0000_0000_1111_1110	9
0111	0000_0001_1111_1111	0000_0000_1111_1111	0000_0000_1111_1111	9
0000	1111_1111_0000_0000	No Conversion	0000_1111_0000_0000	8
0001	0111_1111_1000_0000	0000_0000_1000_0000	0000_1111_1000_0000	8
0010	0011_1111_1100_0000	0000_0000_1100_0000	0000_1111_1100_0000	8
0011	0001_1111_1110_0000	0000_0000_1110_0000	0000_1111_1110_0000	8
0100	0000_1111_1111_0000	0000_0000_1111_0000	0000_0000_1111_0000	8
0101	0000_0111_1111_1000	0000_0000_1111_1000	0000_0000_1111_1000	8
0110	0000_0011_1111_1100	0000_0000_1111_1100	0000_0000_1111_1100	8
0111	0000_0001_1111_1110	0000_0000_1111_1110	0000_0000_1111_1110	8
1000	0000_0000_1111_1111	No Conversion	0000_0000_0000_1111	8
0000	1111_1110_0000_0000	No Conversion	0000_1110_0000_0000	7
0001	0111_1111_0000_0000	No Conversion	0000_1111_0000_0000	7
0010	0011_1111_1000_0000	0000_0000_1000_0000	0000_1111_1000_0000	7
0011	0001_1111_1100_0000	0000_0000_1100_0000	0000_1111_1100_0000	7
0100	0000_1111_1110_0000	0000_0000_1110_0000	0000_0000_1110_0000	7
0101	0000_0111_1111_0000	0000_0000_1111_0000	0000_0000_1111_0000	7
0110	0000_0011_1111_1000	0000_0000_1111_1000	0000_0000_1111_1000	7
0111	0000_0001_1111_1100	0000_0000_1111_1100	0000_0000_1111_1100	7
1000	0000_0000_1111_1110	No Conversion	0000_0000_0000_1110	7
1001	0000_0000_0111_1111	No Conversion	0000_0000_0000_1111	7
0000	1111_1100_0000_0000	No Conversion	0000_1100_0000_0000	6
0001	0111_1110_0000_0000	No Conversion	0000_1110_0000_0000	6
0010	0011_1111_0000_0000	No Conversion	0000_1111_0000_0000	6

Table 5-9. Byte Enables for 128-Bit Conversion Cycles (Sheet 3 of 4)

Current Cycle Mn_ABus(28:31)	Current Cycle Mn_BE(0:15)	Conversion Cycle for 64-Bit Slave Mn_BE(0:15)	Conversion Cycle for 32-Bit Slave Mn_BE(0:15)	Transfer Size (Bytes)
0011	0001_1111_1000_0000	0000_0000_1000_0000	0000_1111_1000_0000	6
0100	0000_1111_1100_0000	0000_0000_1100_0000	0000_0000_1100_0000	6
0101	0000_0111_1110_0000	0000_0000_1110_0000	0000_0000_1110_0000	6
0110	0000_0011_1111_0000	0000_0000_1111_0000	0000_0000_1111_0000	6
0111	0000_0001_1111_1000	0000_0000_1111_1000	0000_0000_1111_1000	6
1000	0000_0000_1111_1100	No Conversion	0000_0000_0000_1100	6
1001	0000_0000_0111_1110	No Conversion	0000_0000_0000_1110	6
1010	0000_0000_0011_1111	No Conversion	0000_0000_0000_1111	6
0000	1111_1000_0000_0000	No Conversion	0000_1000_0000_0000	5
0001	0111_1100_0000_0000	No Conversion	0000_1100_0000_0000	5
0010	0011_1110_0000_0000	No Conversion	0000_1110_0000_0000	5
0011	0001_1111_0000_0000	No Conversion	0000_1111_0000_0000	5
0100	0000_1111_1000_0000	0000_0000_1000_0000	0000_0000_1000_0000	5
0101	0000_0111_1100_0000	0000_0000_1100_0000	0000_0000_1100_0000	5
0110	0000_0011_1110_0000	0000_0000_1110_0000	0000_0000_1110_0000	5
0111	0000_0001_1111_0000	0000_0000_1111_0000	0000_0000_1111_0000	5
1000	0000_0000_1111_1000	No Conversion	0000_0000_0000_1000	5
1001	0000_0000_0111_1100	No Conversion	0000_0000_0000_1100	5
1010	0000_0000_0011_1110	No Conversion	0000_0000_0000_1110	5
1011	0000_0000_0001_1111	No Conversion	0000_0000_0000_1111	5
0001	0111_1000_0000_0000	No Conversion	0000_1000_0000_0000	4
0010	0011_1100_0000_0000	No Conversion	0000_1100_0000_0000	4
0011	0001_1110_0000_0000	No Conversion	0000_1110_0000_0000	4
0101	0000_0111_1000_0000	0000_0000_1000_0000	0000_0000_1000_0000	4
0110	0000_0011_1100_0000	0000_0000_1100_0000	0000_0000_1100_0000	4
0111	0000_0001_1110_0000	0000_0000_1110_0000	0000_0000_1110_0000	4
1001	0000_0000_0111_1000	No Conversion	0000_0000_0000_1000	4
1010	0000_0000_0011_1100	No Conversion	0000_0000_0000_1100	4
1011	0000_0000_0001_1110	No Conversion	0000_0000_0000_1110	4
0010	0011_1000_0000_0000	No Conversion	0000_1000_0000_0000	3
0011	0001_1100_0000_0000	No Conversion	0000_1100_0000_0000	3
0110	0000_0011_1000_0000	0000_0000_1000_0000	0000_0000_1000_0000	3
0111	0000_0001_1100_0000	0000_0000_1100_0000	0000_0000_1100_0000	3
1010	0000_0000_0011_1000	No Conversion	0000_0000_0000_1000	3
1011	0000_0000_0001_1100	No Conversion	0000_0000_0000_1100	3

Table 5-9. Byte Enables for 128-Bit Conversion Cycles (Sheet 4 of 4)

Current Cycle Mn_ABus(28:31)	Current Cycle Mn_BE(0:15)	Conversion Cycle for 64-Bit Slave Mn_BE(0:15)	Conversion Cycle for 32-Bit Slave Mn_BE(0:15)	Transfer Size (Bytes)
0011	0001_1000_0000_0000	No Conversion	0000_1000_0000_0000	2
0111	0000_0001_1000_0000	0000_0000_1000_0000	0000_0000_1000_0000	2
1011	0000_0000_0001_1000	No Conversion	0000_0000_0001_1000	2

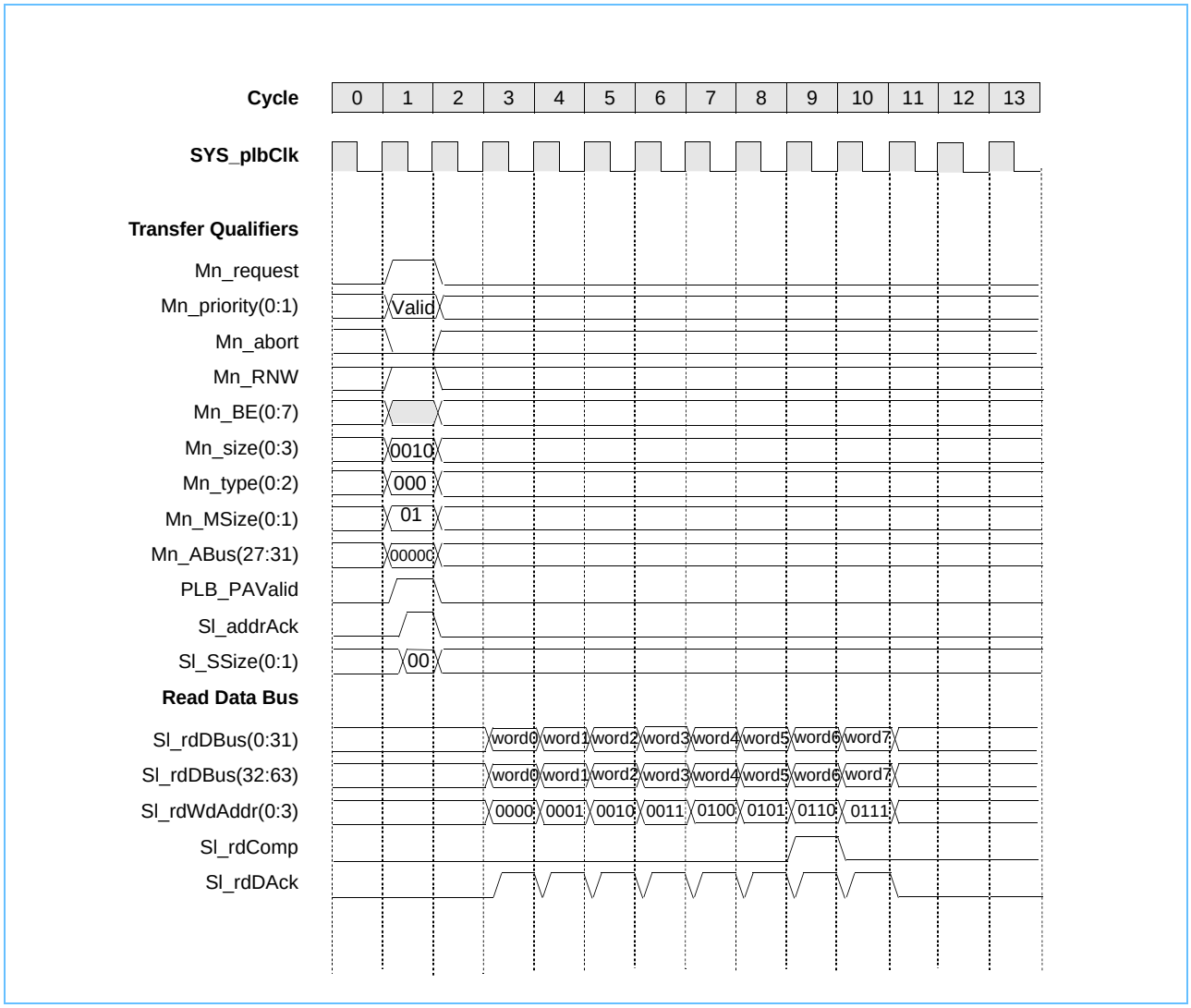
5.6.8 Line Transfers

Line transfers that are initiated by a master might take on a different appearance based on the size of the slave claiming the transfer. The master must account for this difference by detecting the slave size and possibly modifying the number of expected data transfers.

5.6.8.1 64-Bit Master 8-Word Line Read from a 32-Bit Slave

Figure 5-44 64-Bit Master 8-Word Line Read from a 32-Bit Slave on page 145 illustrates a 64-bit master 8-word line read from a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master it is a 32-bit slave with Sl_SSize(0:1) equal to ‘00’. The master must then expect eight assertions of Sl_rdDAck and only 32 bits of data per data phase. The master must monitor the Sl_rdWdAddr(0:3) signals to determine which word in the line is being transferred. Because the data is mirrored by the slave, the master can expect the word data in the correct position on the data bus.

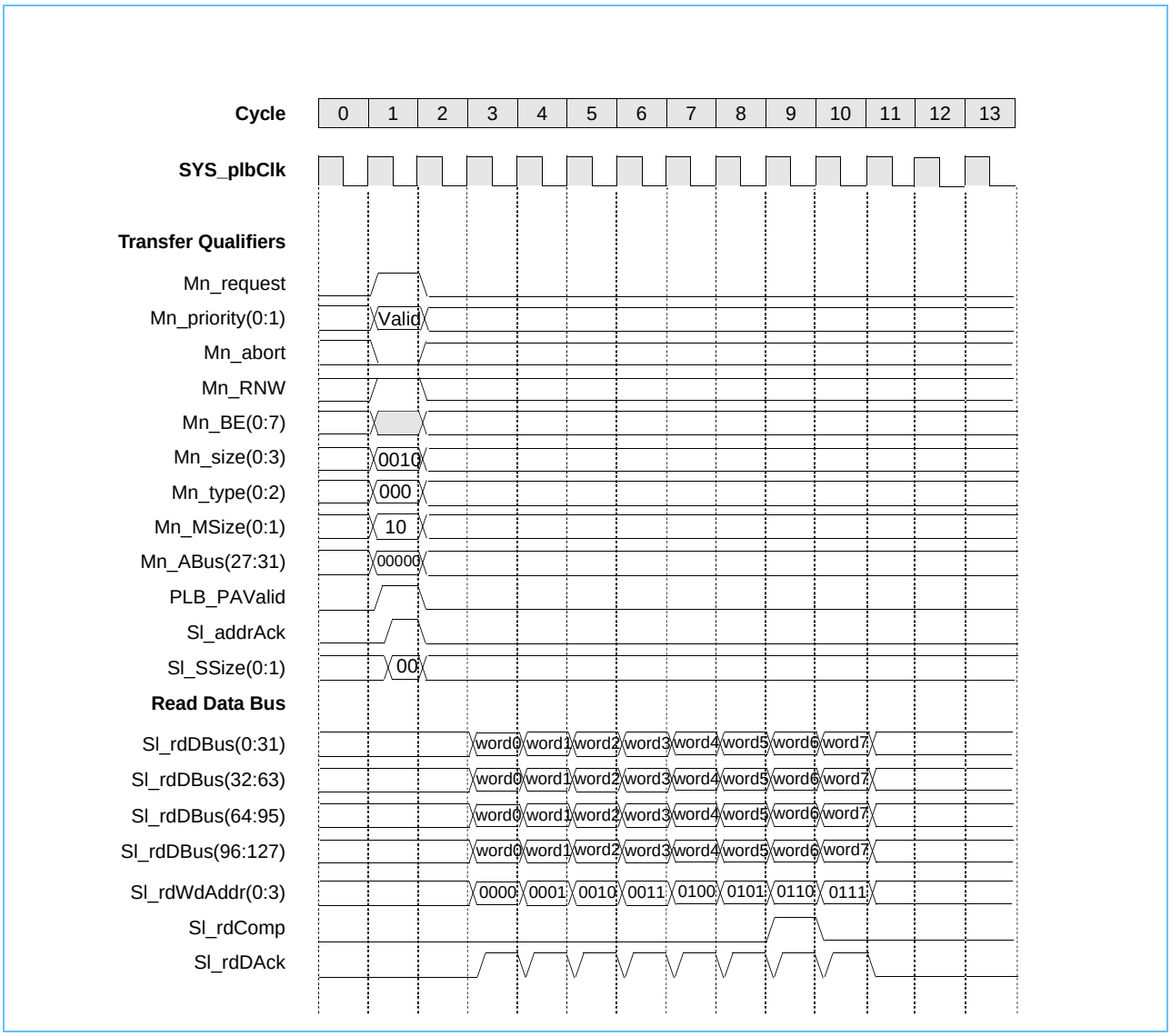
Figure 5-44. 64-Bit Master 8-Word Line Read from a 32-Bit Slave



5.6.8.2 128-Bit Master 8-Word Line Read from a 32-Bit Slave

Figure 5-45 illustrates a 128-bit master 8-word line read from a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must then expect eight assertions of the Sl_rdDAck signal and only 32 bits of data per data phase. The master must monitor the Sl_rdWdAddr(0:3) signals to determine which word in the line is being transferred. Because the data is mirrored by the slave, the master can expect the word data in the correct position on the data bus.

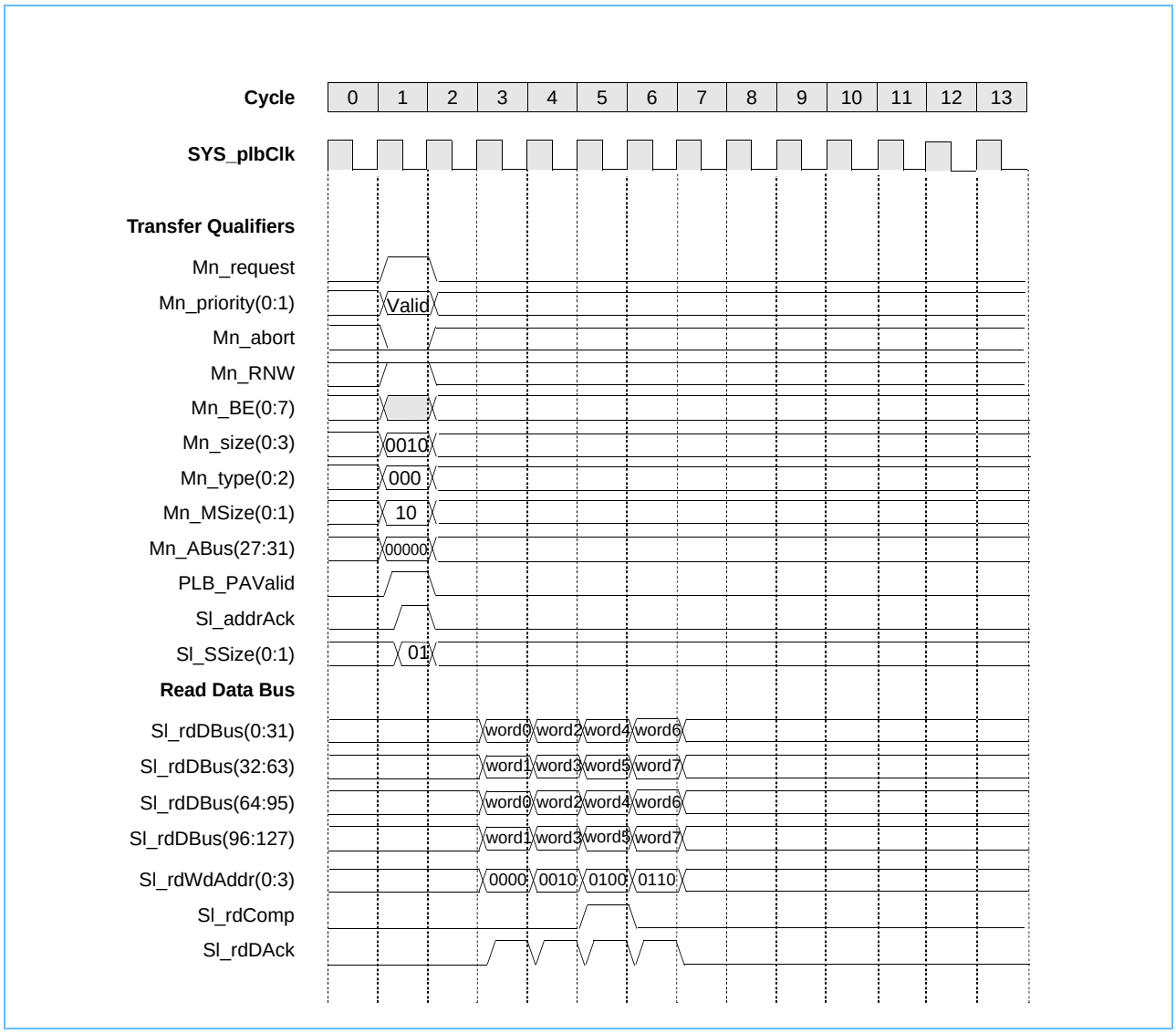
Figure 5-45. 128-Bit Master 8-Word Line Read from a 32-Bit Slave



5.6.8.3 128-Bit Master 8-Word Line Read from a 64-Bit Slave

Figure 5-46 illustrates a 128-bit master 8-word line read from a 64-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 64-bit slave with Sl_SSize(0:1) equal to '01'. The master must then expect four assertions of the Sl_rdDAck signal and only 64 bits of data per data phase. The master must monitor the Sl_rdWdAddr(0:3) signals to determine which words in the line are being transferred. Because the data is mirrored by the slave, the master can expect the word data in the correct position on the data bus.

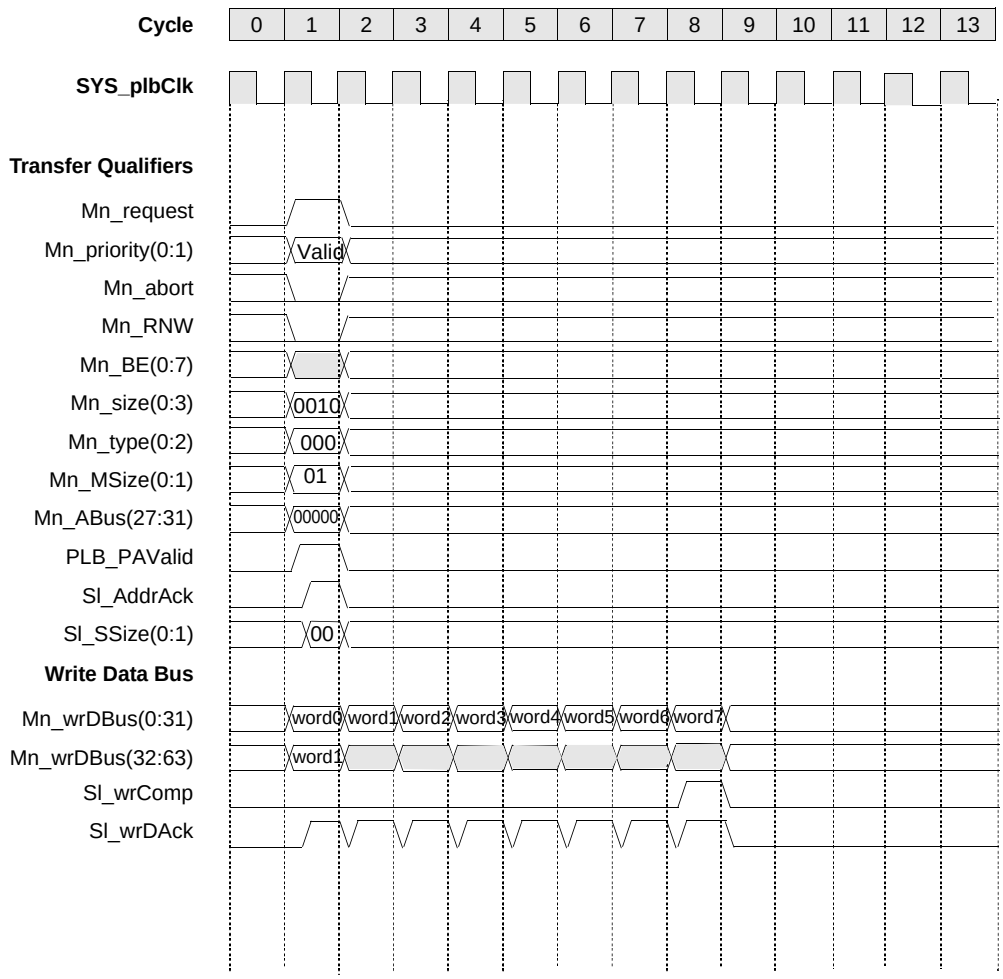
Figure 5-46. 128-Bit Master 8-Word Line Read from a 64-Bit Slave



5.6.8.4 64-Bit Master 8-Word Line Write to a 32-Bit Slave

Figure 5-47 illustrates a 64-bit master 8-word line write to a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must expect eight assertions of Sl_wrDAck writing only 32 bits of data per data phase. The master must provide word0 and word1 on the data bus during the request phase, and then steer data bus bits 32:63 down to bits 0:31 appropriately during the rest of the transfer.

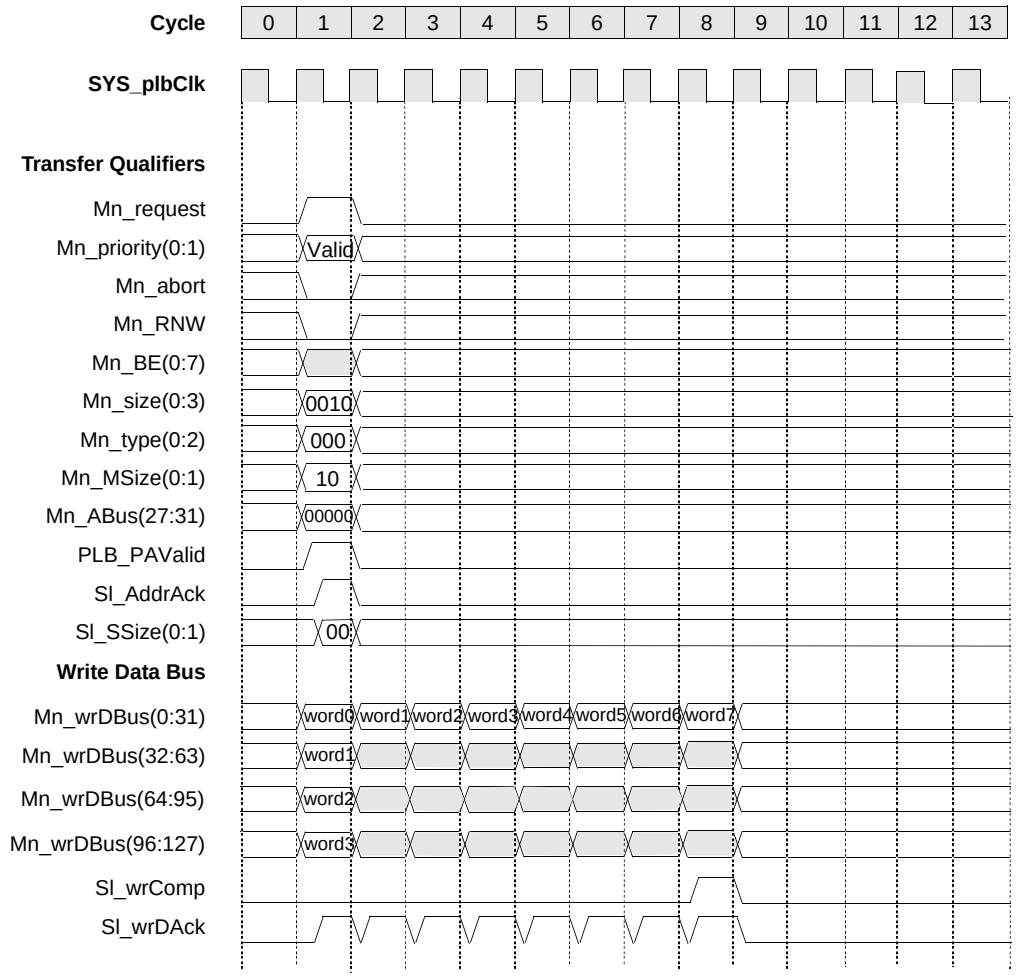
Figure 5-47. 64-Bit Master 8-Word Line Write to a 32-Bit Slave



5.6.8.5 128-Bit Master 8-Word Line Write to a 32-Bit Slave

Figure 5-48 illustrates a 128-bit master 8-word line write to a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must expect eight assertions of Sl_wrDAck writing only 32 bits of data per data phase. The master must provide word0 through word3 on the data bus during the request phase and then steer data bus bits 32:127 down to bits 0:31 appropriately during the rest of the transfer.

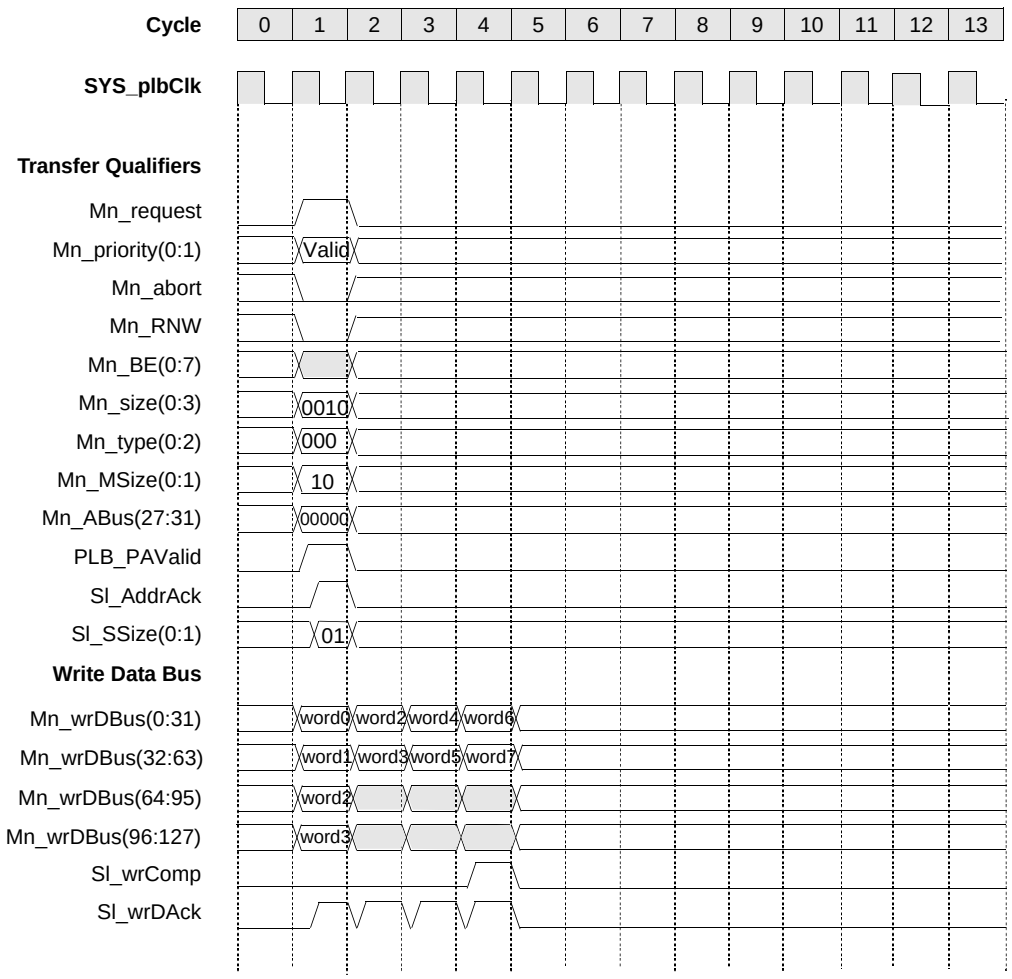
Figure 5-48. 128-Bit Master 8-Word Line Write to a 32-Bit Slave



5.6.8.6 128-Bit Master 8-word Line Write to a 64-Bit Slave

Figure 5-49 illustrates a 128-bit master 8-word line write to a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must expect four assertions of Sl_wrDAck writing only 32 bits of data per data phase. The master must provide word0 and word1 on the data bus during the request phase and then steer data bus bits 32:63 down to bits 0:31 appropriately during the rest of the transfer.

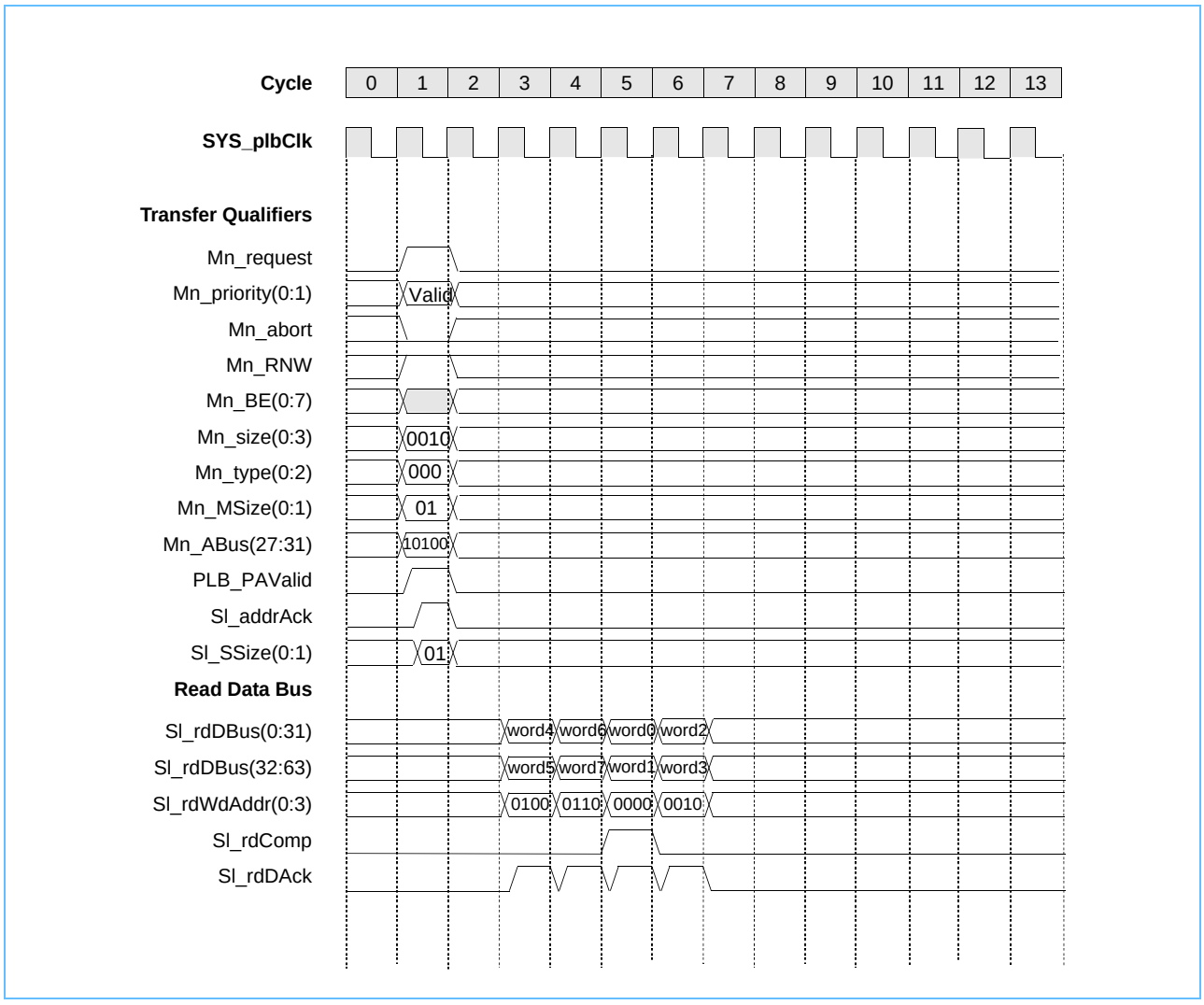
Figure 5-49. 128-Bit Master 8-Word Line Write to a 64-Bit Slave



5.6.8.7 64-Bit Master 8-Word Line Read from a 64-Bit Slave (Target Word First)

Figure 5-50 illustrates a 64-bit master 8-word line read by a 64-bit master from a 64-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 64-bit slave with Sl_SSize(0:1) equal to '01'. The master is requesting word5 first. The master must monitor the Sl_rdWdAddr(0:3) signals to determine which words in the line are being transferred. Because the slave implements the target-word-first protocol, it returns word4 and word5 first. The transfer continues until all eight words are read.

Figure 5-50. 64-Bit Master 8-Word Line Read from a 64-Bit Slave



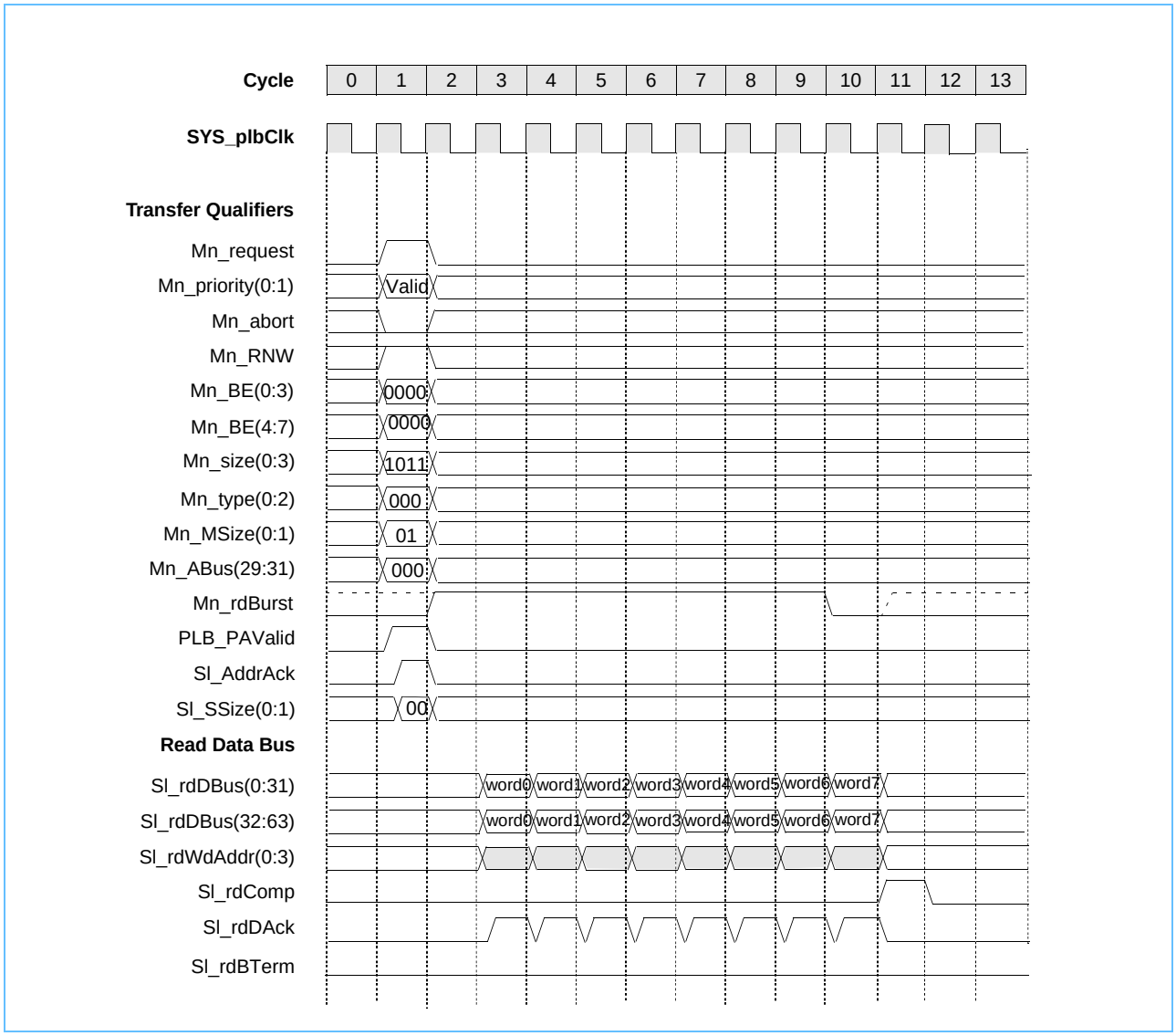
5.6.9 Burst Transfers

Burst transfers that are initiated by a master might take on a different appearance based on the size of the slave that is claiming the transfer. The master must account for this difference by detecting the slave size and handling the transfer appropriately.

5.6.9.1 64-Bit Master 4-Doubleword Burst Read from a 32-Bit Slave

Figure 5-51 illustrates a 64-bit master 4-doubleword burst read from a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must then expect only 32 bits of data per data phase. Because the data is mirrored by the slave the master can expect the word data in the correct position on the data bus. The master must deassert its Mn_rdBurst signal after sampling seven assertions of the Sl_rdDAck signal.

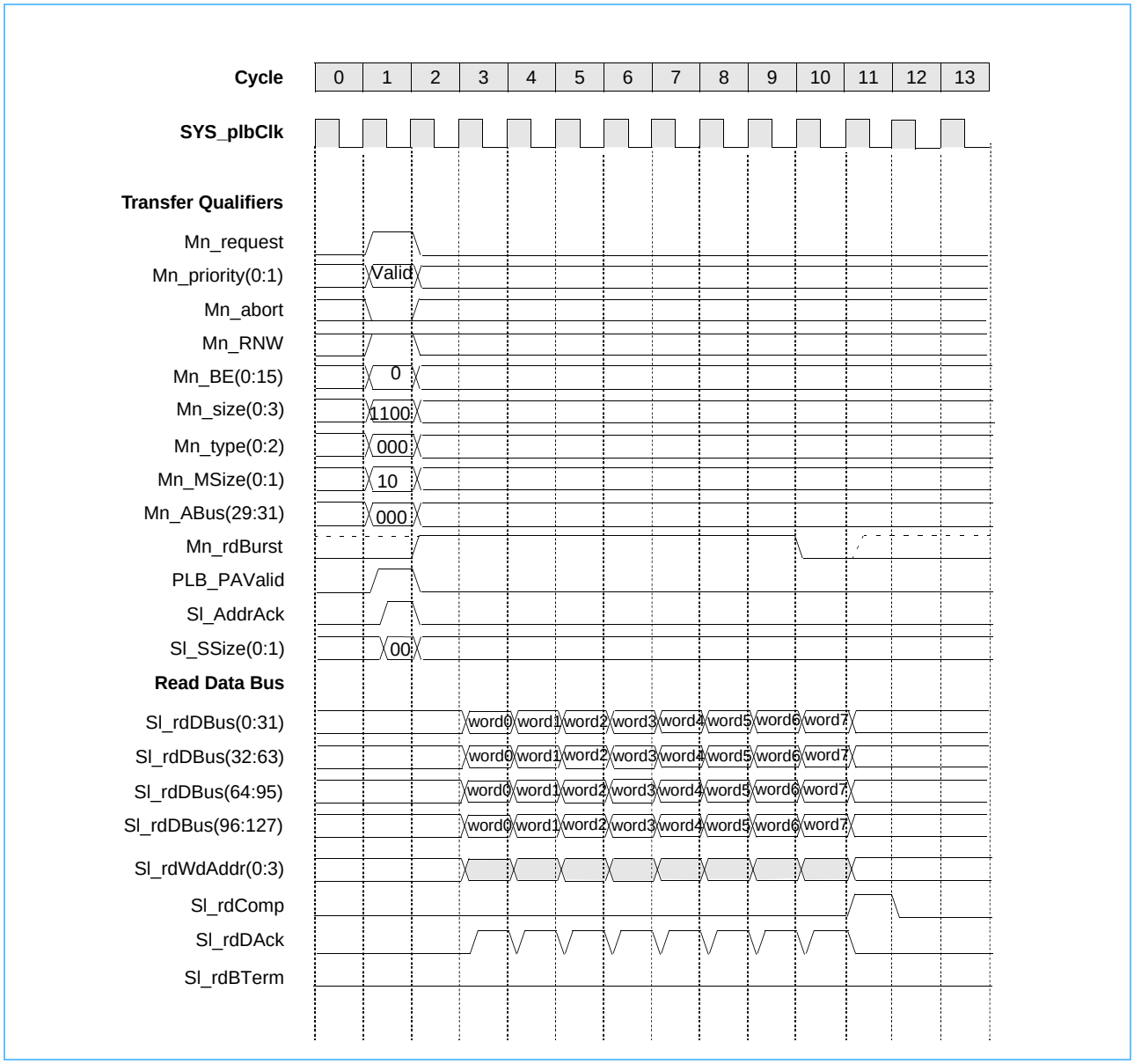
Figure 5-51. 64-Bit Master 4-Doubleword Burst Read from a 32-Bit Slave



5.6.9.2 128-Bit Master 2-Quadword Burst Read from a 32-Bit Slave

Figure 5-52 illustrates a 128-bit master 2-quadword burst read from a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must then expect only 32 bits of data per data phase. Because the data is mirrored by the slave, the master can expect the word data in the correct position on the data bus. The master must deassert its Mn_rdBurst signal after sampling seven assertions of the Sl_rdDAck signal.

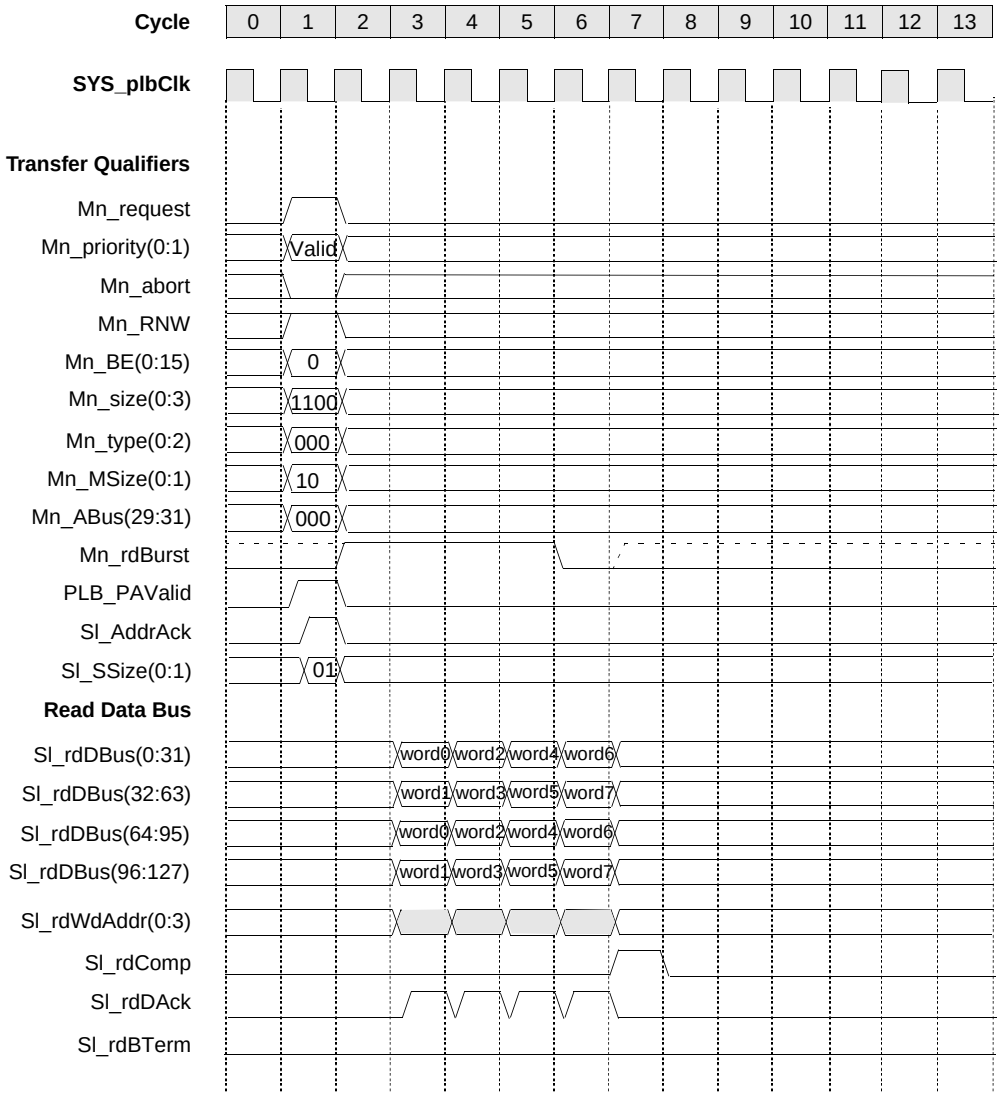
Figure 5-52. 128-Bit Master 2-Quadword Burst Read from a 32-Bit Slave



5.6.9.3 128-Bit Master 2-Quadword Burst Read from a 64-Bit Slave

Figure 5-53 illustrates a 128-bit master 2-quadword burst read from a 64-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 64-bit slave with Sl_SSize(0:1) equal to '01'. The master must then expect only 64 bits of data per data phase. Because the data is mirrored by the slave, the master can expect the word data in the correct position on the data bus. The master must deassert its Mn_rdBurst signal after sampling four assertions of the Sl_rdDAck signal.

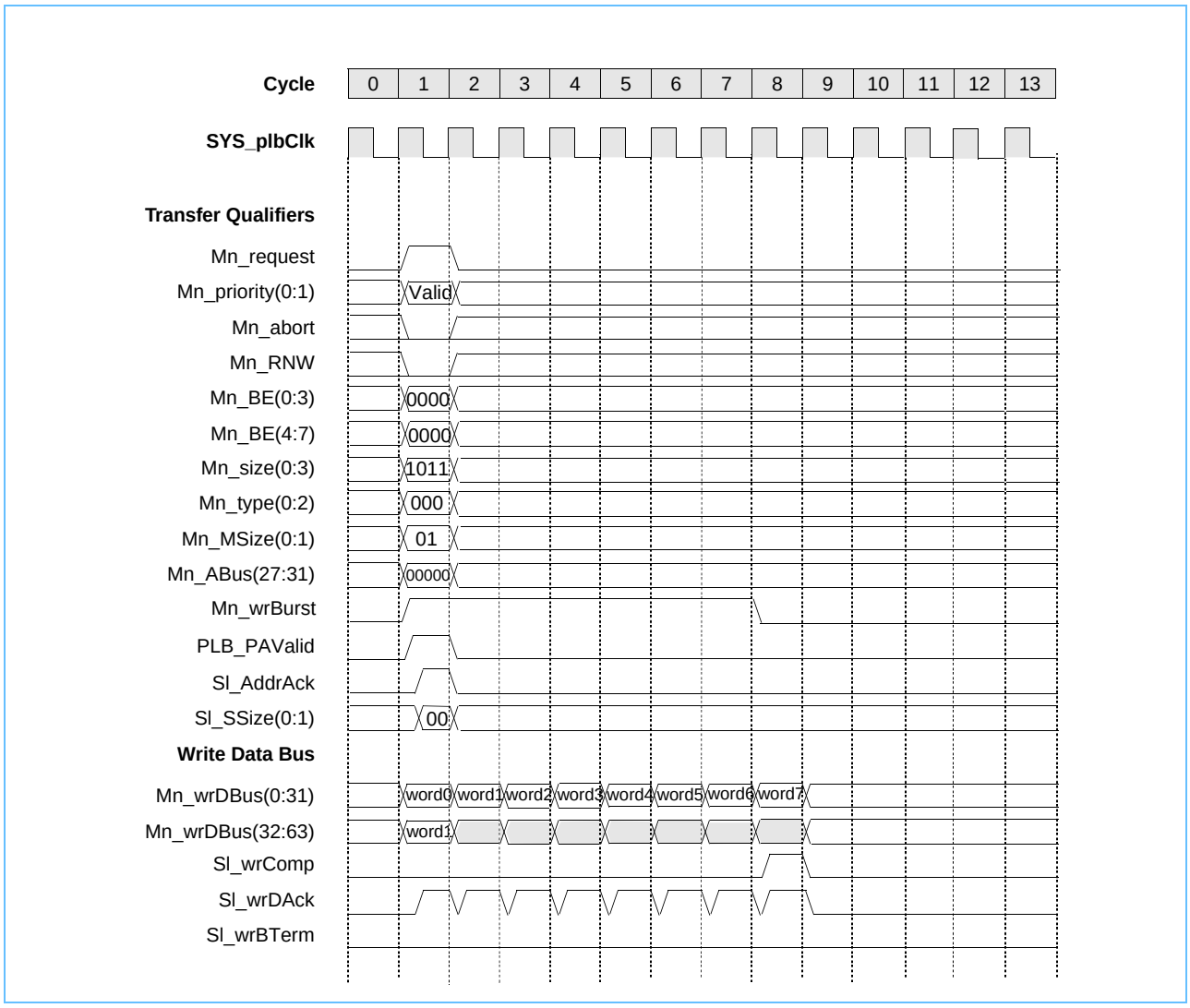
Figure 5-53. 128-Bit Master 2-Quadword Burst Read from a 64-Bit Slave



5.6.9.4 64-Bit Master 4-Doubleword Burst Write to a 32-Bit Slave

Figure 5-54 illustrates a 4-doubleword burst write to a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must expect eight assertions of Sl_wrDAck writing only 32 bits of data per data phase. The master must provide word0 and word1 on the data bus during the request phase, and then steer data bus bits 32:63 down to bits 0:31 appropriately during the rest of the transfer. The master must deassert its Mn_wrBurst signal after sampling seven assertions of the wrDAck signal.

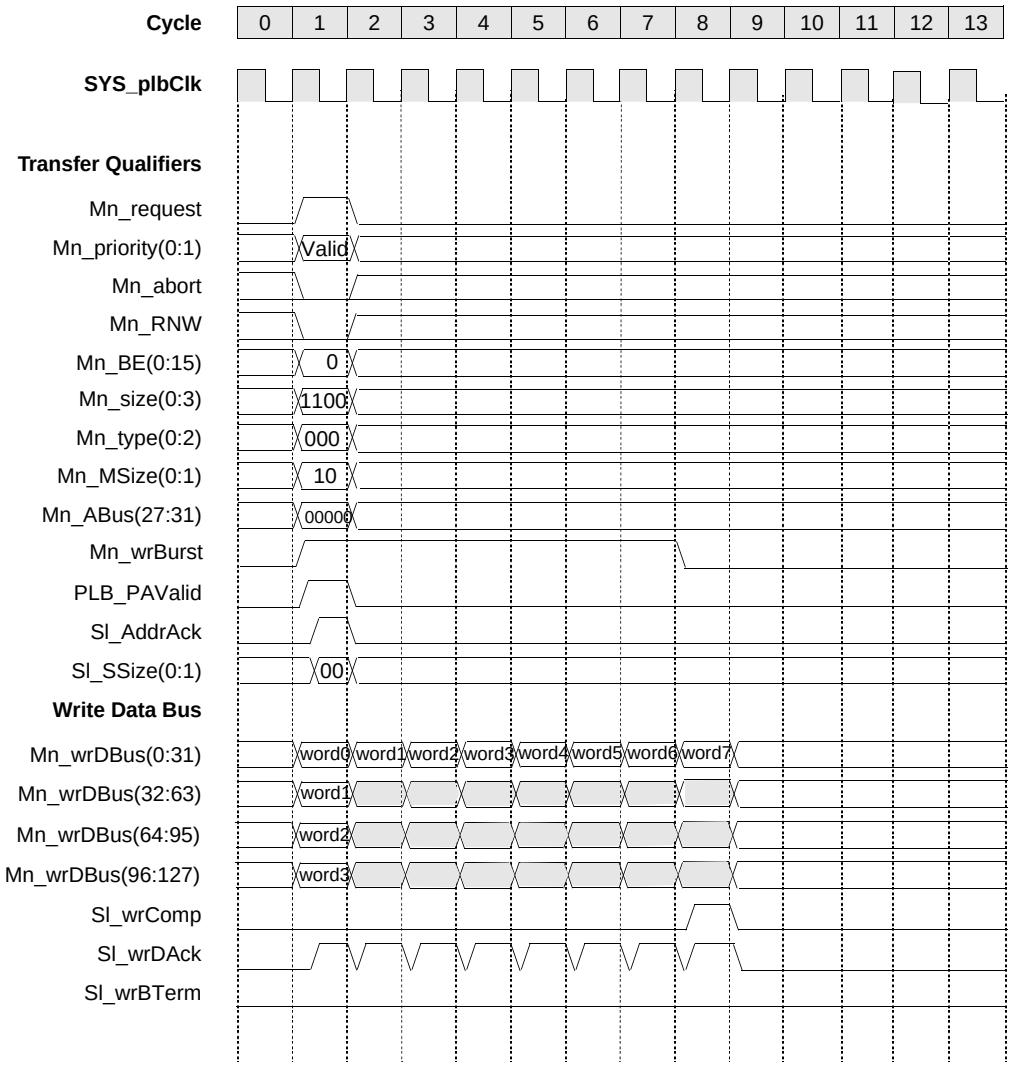
Figure 5-54. 64-Bit Master 4-Doubleword Burst Write to a 32-Bit Slave



5.6.9.5 128-Bit Master 2-Quadword Burst Write to a 32-Bit Slave

Figure 5-55 illustrates a 128-bit master 2-quadword burst write to a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must expect eight assertions of Sl_wrDAck writing only 32 bits of data per data phase. The master must provide word0 word3 the data bus during the request phase, and then steer data bus bits 32:127 down to bits 0:31 appropriately during the rest of the transfer. The master must deassert its Mn_wrBurst signal after sampling seven assertions of the wrDAck signal.

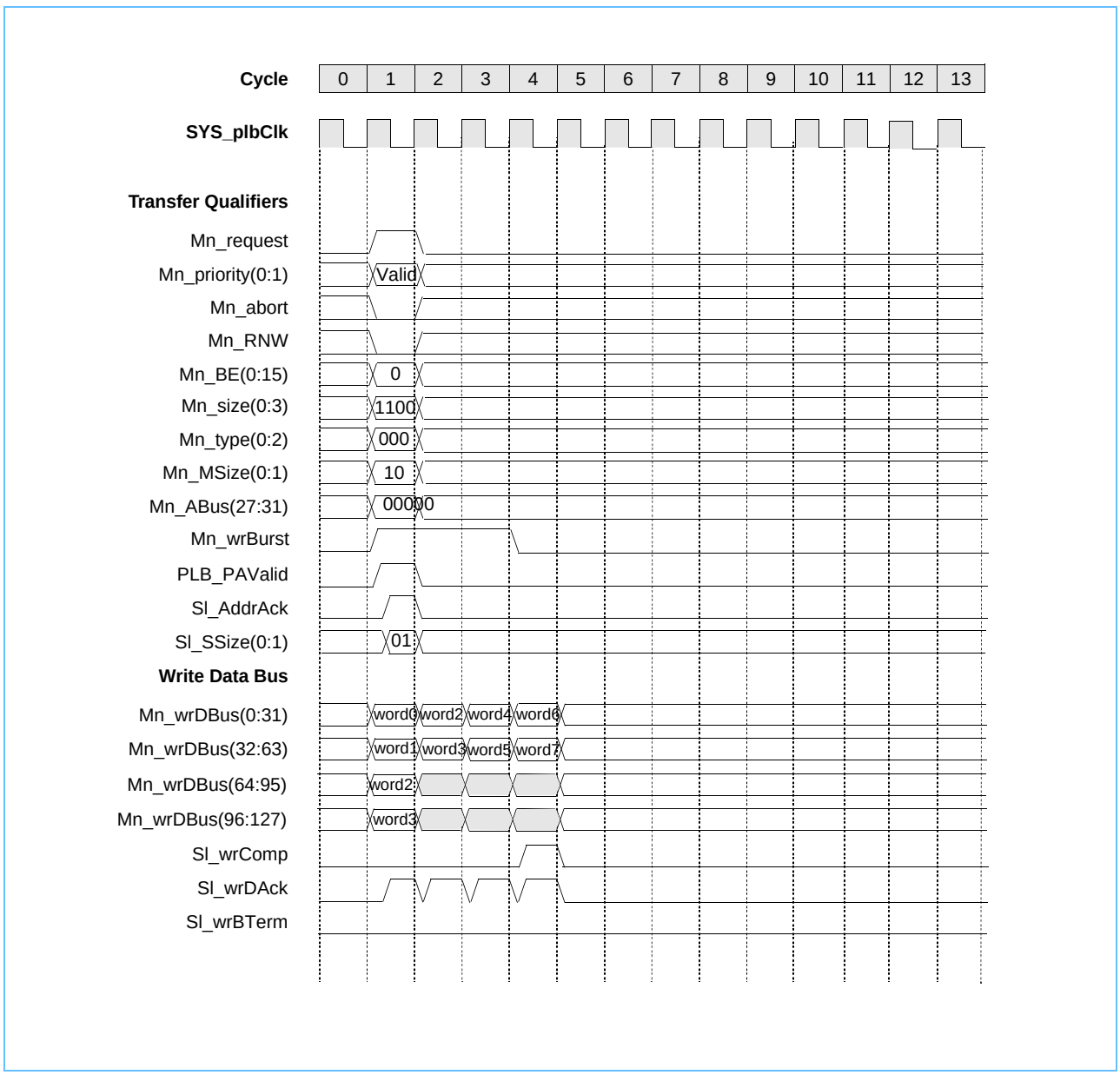
Figure 5-55. 128-Bit Master 2-Quadword Burst Write to a 32-Bit Slave



5.6.9.6 128-Bit Master 2-Quadword Burst Write to a 64-Bit Slave

Figure 5-56 illustrates a 128-bit master 2-quadword burst write to a 64-bit slave. The slave acknowledges the request, through the Sl_addrAck signal, and indicates to the master that it is a 64-bit slave with Sl_SSize(0:1) equal to '01'. The master must expect four assertions of Sl_wrDAck writing only 64 bits of data per data phase. The master must provide word0 word3 the data bus during the request phase, and then steer data bus bits 64:127 down to bits 0:63 appropriately during the rest of the transfer. The master must deassert its Mn_wrBurst signal after sampling three assertions of the wrDAck signal.

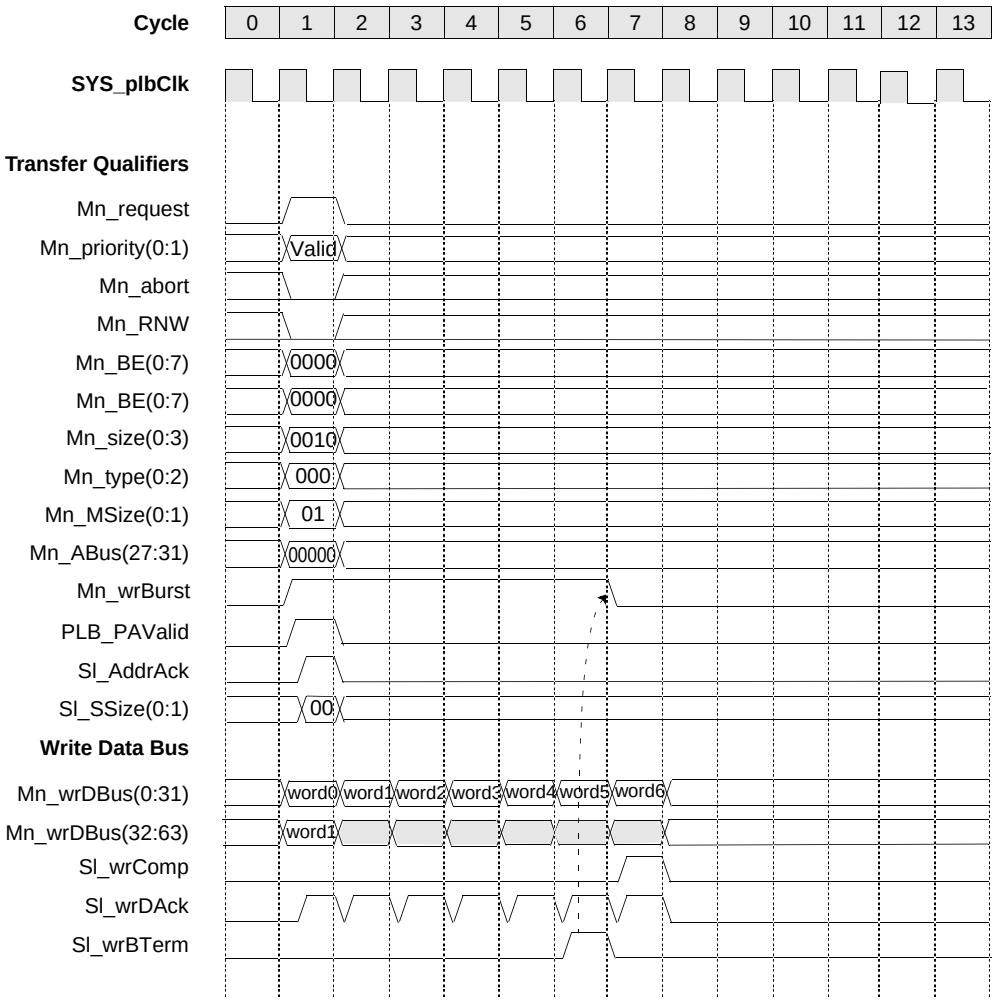
Figure 5-56. 128-Bit Master 2-Quadword Burst Write to a 64-Bit Slave



5.6.9.7 Slave Terminated 64-Bit Master Burst Write to a 32-Bit Slave

Figure 5-57 illustrates a 64-bit master burst write to a 32-bit slave. The slave acknowledges the request, through the Sl_addrAck signal and indicates to the master that it is a 32-bit slave with Sl_SSize(0:1) equal to '00'. The master must expect only 32 bits of data per data phase. The master must provide word0 and word1 on the data bus during the request phase, and then steer data bus bits 32:63 down to bits 0:31 appropriately throughout the transfer. The master must deassert its Mn_wrBurst signal after sampling Sl_wrBTerm asserted. One more transfer occurs. The master cannot request another doubleword burst because it is no longer aligned on a doubleword boundary.

Figure 5-57. Slave Terminated 64-Bit Master Burst Write to a 32-Bit Slave



5.7 PLB Parity

The PLB architecture allows for parity on the address, byte enable, and read and write data buses. Parity is optional for each master and slave. Masters that support parity work with slaves that do not support parity and slaves that support parity. Similarly, slaves that support parity work with masters that do not support parity and masters that support parity. Systems that support parity must use a PLB arbiter that routes the parity signals to the masters and slaves. The PLB arbiter does not perform any parity checking.

All parity bits generated and checked are considered to be odd parity. The sum of logical ones in a bus for which parity is generated and the parity bit itself is an odd number.

Data bytes that are not valid data (no byte-enable asserted) are not required to have valid parity.

5.7.1 Parity Checking and Reporting in Masters

When parity is generated by a PLB slave for read data, the Sln_rdDBusParEn and Sln_rdDBusPar signals are driven with the Sln_rdDBus. These signals are routed through the PLB arbiter, and the PLB master checks the PLB_MnRdDBus with the PLB_MnRdDBusPar, if the PLB_MnRdDBusParEn is asserted. If a read data parity error is detected, the master must assert an interrupt and record the error address and status in a syndrome register.

5.7.2 Parity Checking and Reporting in Slaves

When parity is generated by a PLB master for the transfer qualifiers, Mn_UABusParEn and Mn_UABusPar are driven coincident with Mn_UABus; Mn_ABusParEn and Mn_ABusPar are driven coincident with Mn_ABus; and Mn_BEParEn and Mn_BEPar are driven coincident with Mn_BE. These signals are routed through the PLB arbiter, and the PLB slave checks PLB_UABus with PLB_UABusPar if PLB_UABusParEn is asserted. The PLB checks PLB_ABus with PLB_ABusPar if PLB_ABusParEn is asserted. And the PLB checks the PLB_BEs with PLB_BEPar if PLB_BEParEn is asserted. If the PLB slave determines that there is a parity error on one or more of these buses, it might choose to ignore the transfer and allow the transfer to timeout. The slave must then assert either the Sln_MIRQ signal to the master that made the request or a master independent interrupt, and record the parity error address and status in a syndrome register.

The PLB slave that determines that there is a parity error on either the UABus, ABus, or BE signals, might also choose to accept the transfer with the Sln_addrAck signal. If this is the case, and the transfer is a read, the slave must assert the Sln_rdErr signal with any corresponding Sln_rdDAck signal. If the transfer is accepted with the Sln_addrAck signal, and the transfer is a write, the PLB slave can assert the Sln_wrErr signal with a corresponding Sln_wrDAck signal. In either of these cases, the slave must record the parity error address and status in a syndrome register, but it is not necessary for the slave to assert an interrupt because the requesting master is notified of the problem through the PLB_MnRdErr or PLB_MnWrErr signals.

If the transfer is accepted with the Sln_addrAck signal, and the transfer is a write, the PLB slave can wait until all of the Sln_wrDAck signals have been asserted, and then assert the Sln_MIRQ signal to the master that made the request. Then, the slave can record the parity error address and status in a syndrome register.

When parity is generated by a PLB master for the write data bus, Mn_wrDBusParEn and Mn_wrDBusPar are driven coincident with Mn_wrDBus. These signals are routed through the PLB arbiter, and the PLB slave checks the PLB_wrDBus with PLB_wrDBusPar if PLB_wrDBusParEn is asserted. If a PLB slave determines that there is a write data parity error on the PLB_wrDBus when the PLB_PAValid signal is asserted, it might

choose to ignore the write and allow the write transfer to timeout. The slave must then assert either the Sln_MIRQ signal to the master that made the request, or a master independent interrupt, and record the parity error address and status in a syndrome register.

If a PLB slave detects a parity error on write data that is accepted after the assertion of the Sln_addrAck signal, the PLB slave might assert Sln_wrErr with corresponding the Sln_wrDAck signal. If this is the case, the slave must record the parity error address and status in a syndrome register, but it is not necessary for the slave to assert an interrupt because the master is notified of the problem through the PLB_MnWrErr signals. The PLB slave can wait until all of the Sln_wrDAck have been asserted, and then assert Sln_MIRQ to the master that made the request, and record the parity error address and status in a syndrome register.

Table 5-10 summarizes the types of parity errors that can be detected by the PLB master or slave, and the resulting actions that can be taken.

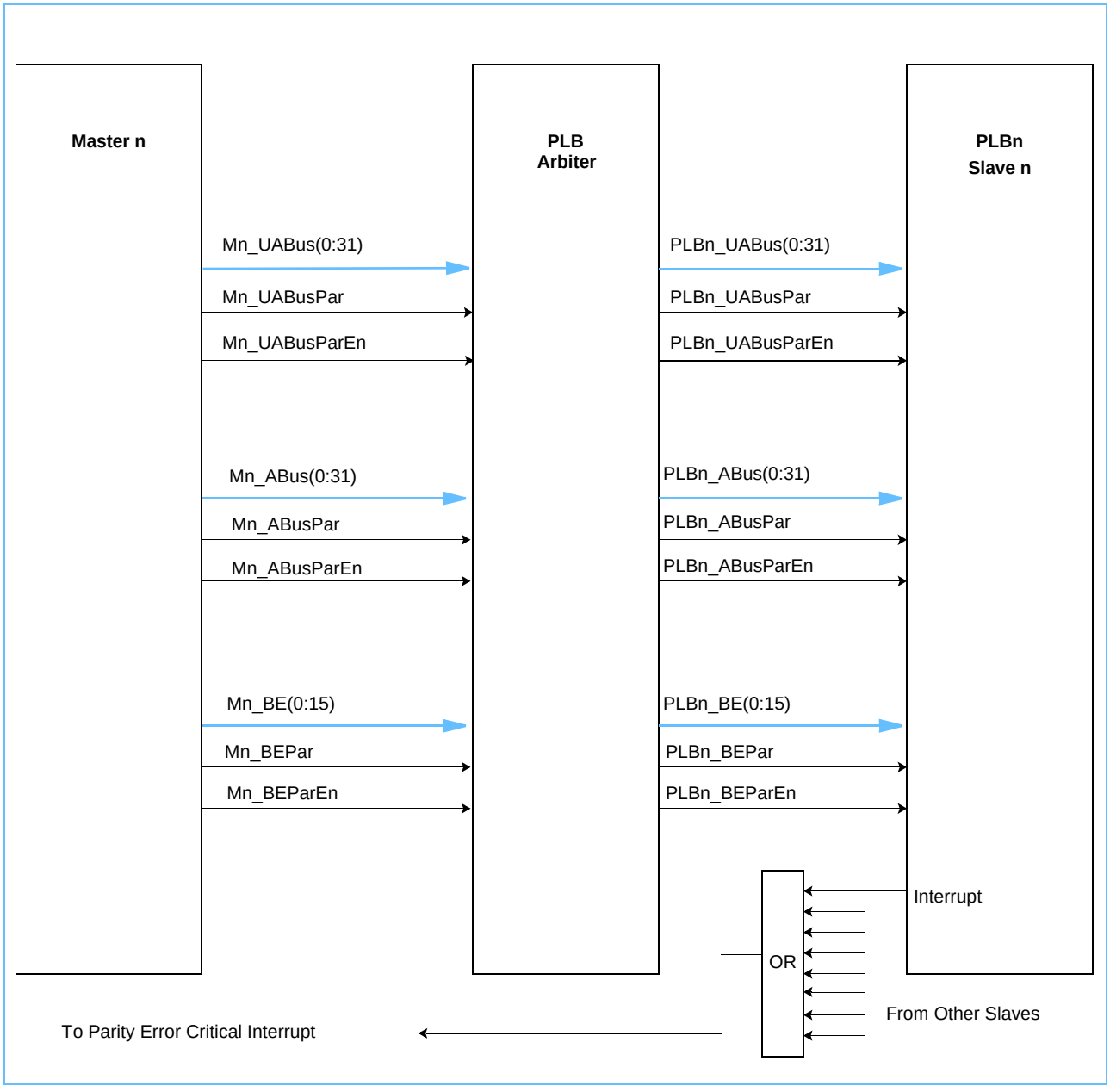
Table 5-10. PLB Parity Error

Read/ Write	Parity error detected by the	on the	results in a	causing the	to assert
Read	PLB master	PLB_MnRdDBus		master	an interrupt
Read/ Write	PLB slave	PLB_UABus, PLB_ABus, PLB_BE, or PLB_wrDBus	PLB_MnTimeout	slave	Sln_MIRQ(m) or an interrupt
Read	PLB slave	PLB_UABus, PLB_ABus, or PLB_BE	PLB_MnAddrAck	slave	PLB_MnRdErr
Write	PLB slave	PLB_UABus, PLB_ABus, PLB_BE, or PLB_wrDBus	PLB_MnAddrAck	slave	PLB_MnWrErr or Sln_MIRQ(m)

5.7.3 Address and Byte Enable Parity

Figure 5-58 illustrates the address and byte enable parity connections to the PLB arbiter. This assumes PLB slaves report a single interrupt when bad parity is detected in PLBn_UABus, PLBn_ABus, or PLBnBE. Also, when only some bits of PLBn_UABus are driven by a master, parity can be generated across those bits only on the condition that all other bits received by any slave are wired to zero going into the arbiter. All slaves must check at least as many PLBn_UABus(0:31) bits as can be driven by any master in properly check parity.

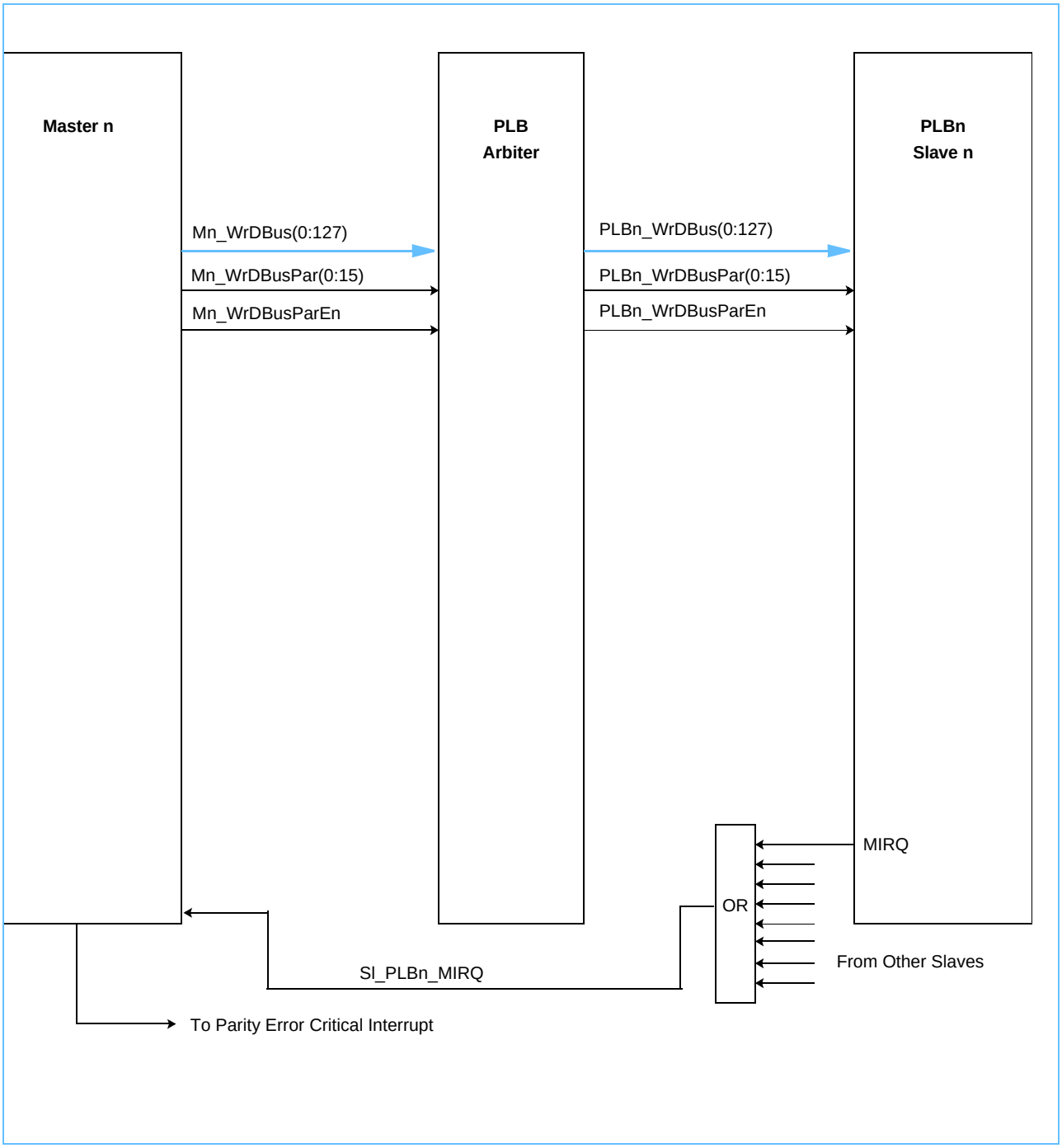
Figure 5-58. Address and Byte Enable Parity



5.7.4 Write Data Parity

The slave that detects the parity error on the PLBn_WrDBus must latch the PLB address for the (first) data byte with bad parity, signal a parity error on the MIRQ, and ideally not forward or use the error data byte. A master that cannot process the master interrupt request (MIRQ) input as a parity error must drive a critical interrupt to the system interrupt controller.

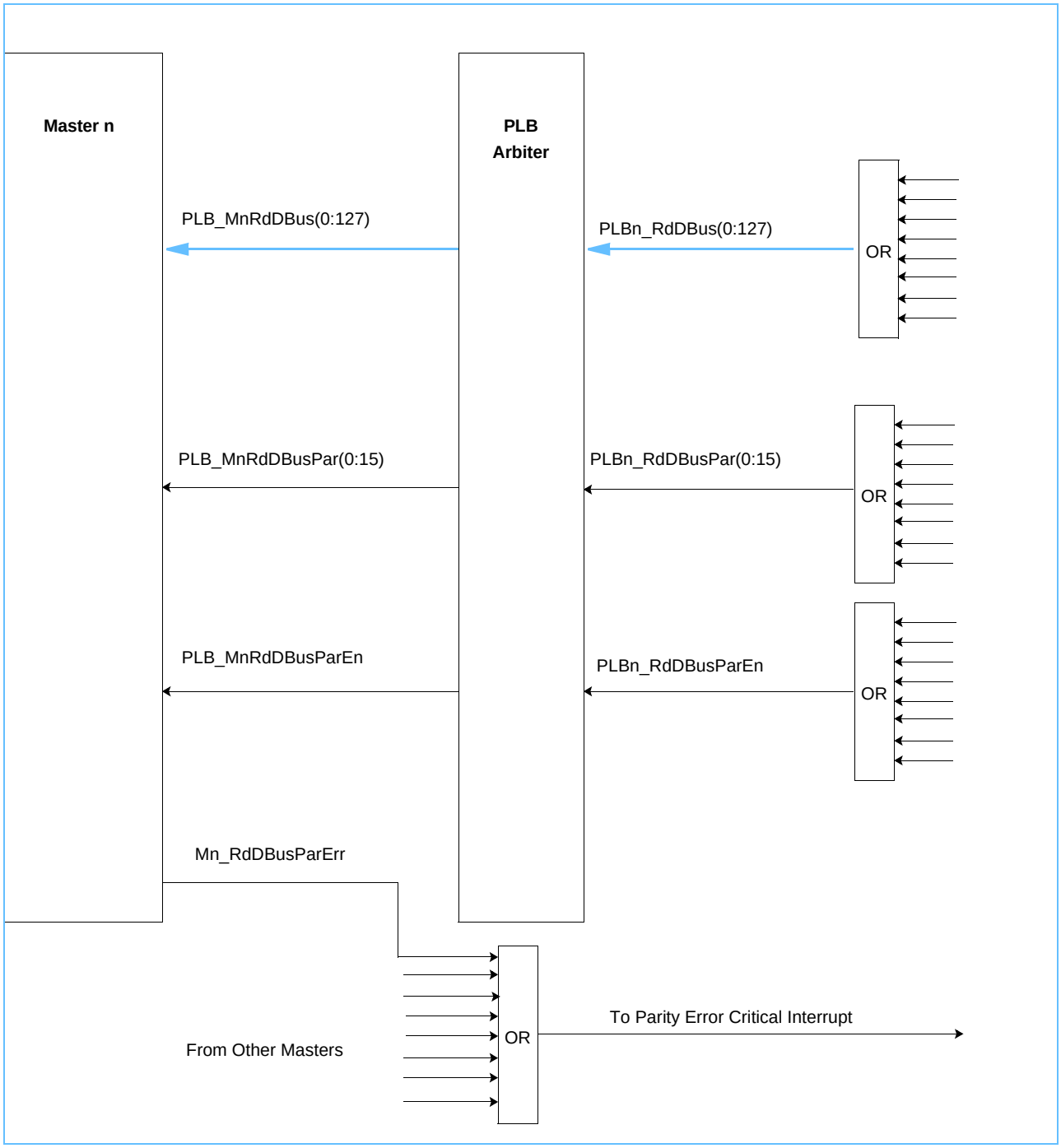
Figure 5-59. Write Data Parity



5.7.5 Read Data Parity

The master that detects the parity error on the PLB_MnRdDBus must latch the PLB address for the first data byte with bad parity, signal a parity error on Mn_RdDBusParErr, and not forward or use the error data byte. The PLBn_RdDBusPar(0:15) and PLBn_RdDBusParEn are activated by a slave coincident with the data and must not be driven active by a slave that does not own the bus for the purpose of driving read data.

Figure 5-60. Read Data Parity



6. Double Data Rate Protocol

6.1 Introduction

Double data rate (DDR) transfers are possible only between master and slave devices that support this feature. This makes sense for devices that need higher bandwidth, such as peripheral component interconnect (PCI) express or GB Ethernet. This protocol is defined under a restrictive subset of processor local bus (PLB) architecture. It does not change the latency of master requests and slave address acknowledgements. For data movement during burst transfers, the rate is doubled by using a 2x clock of the PLB clock. Therefore, two beats of data are transferred for each data acknowledgement received instead of the standard one beat per data acknowledgement.

This DDR protocol is implemented to ensure that new DDR cores are fully compatible with all existing cores. Because the new DDR transfers are transparent to the PLB arbiters, no changes are needed in either arbiter. Thus both non-DDR cores and DDR cores can coexist in a system. And only DDR cores can perform double-data-rate burst transfers between each other. Otherwise, DDR cores transfer data in the standard manner for non-DDR cores or nonburst transfer. Also, DDR cores can choose which transfers they want to be DDR or non-DDR.

When a DDR burst is initiated by two DDR cores, it finishes as a DDR transfer.

6.2 Additional Signals

A 2x clock input is needed for cores that perform DDR. This clock is derived from the PLB clock and is thus edge aligned but at two times the frequency of the PLB clock. This is used to clock or launch the second beat of data for DDR. The first beat of data is still latched or launched using the rising edge of the PLB clock that is coincident with the data acknowledgement signal.

6.3 Restrictions on DDR Transfers

1. DDR is only allowed between 128-bit PLB slaves and 128-bit PLB masters.
2. DDR transfers are only allowed for fixed-length read and write bursts of quadword size (with PLB_size[0:3] b’1100’). Fixed-length bursts are indicated by placing the number of transfers that are requested on the byte enables signals by the master. See*Table 2-7 Byte Enable Signals during Burst Transfers for (64-Bit and above PLB)* on page 40 for more information.
3. DDR beats that are transferred are always in multiples of two quadwords because for every data acknowledgement, two quadword beats are transferred.
4. DDR masters are not allowed to terminate a DDR transfer early. They are also not allowed to extend the DDR transfer by not deasserting their appropriate Mn_rdBurst or Mn_wrBurst signals at the correct time. Additionally, masters must not request DDR transfers that exceed 512 bytes.
5. DDR masters are not allowed to assert the Mn_abort signal for DDR requests.
6. A DDR slave is allowed to terminate a DDR burst early, but must capture or launch both beats of quadword data for the last data acknowledgement.
7. For a DDR read burst, the earliest data that can be returned to the master is two clock cycles after the slave address acknowledgement.

8. For a DDR write burst, the earliest data that is received by the slave is the clock cycle after address acknowledgement.
9. DDR burst are not allowed for guarded transfers; that is, with PLB_TAttribute (3) ‘1’.
10. Slave devices that support the DDR protocol are discouraged from pacing during DDR transfers because this effectively negates the benefit derived from performing DDR transfers in the first place. Pacing is allowed for certain situations, such as a memory controller refresh, or to add a cycle for a row boundary.

    Note: If pacing is performed, it is on a system-clock-boundary to system-clock-boundary.

11) If a master requests a DDR transfer, but requests an odd number of transfers instead of the even multiples of two as described in item number 3, this is an error condition. The intended slave device that responds as a DDR device must log this as an error, It can then perform of the following actions:

11. It can choose to respond as a non-DDR device and perform the transfer, but must still set an error bit in an error status register.
12. It cannot respond, forcing a timeout, and set an error status register bit. The rationale for this is it represents either a software error or a master error.

Note: The toolkit PLB monitor can also detect this error condition from a real device and issue an error message.

6.4 Execution of DDR Transfers

To execute a DDR transfer, DDR PLB master makes a request with the following settings:

- the Mn_Msize[0:1] ‘11’
- Mn_size[0:3] ‘1100’
- Mn_BE[0:7] must be in multiples of two, starting with a minimum of two quadwords.

The handshaking is complete when a PLB slave responds with Sl_Ssize[0:1] = ‘11’. In other words, the slave responds with the slave size in the same cycle as the Sl_addrAck signal. Then, DDR transfer then begins and completes as a DDR transfer.

If the slave responds with any other value for the Sl_Ssize signal, the transfer is a regular *one beat per clock* transfer. See *Table 2-10 Mn_MSize(0:1) Master Size* on page 43 and *Table 2-11 Sl_SSize(0:1) Slave Size* on page 44 to see the various encodings for these signals.

6.4.1 Master Requests DDR Transfer but Slave Responds as Non-DDR Device

If a master requests a DDR transfer with the previous criteria listed in *Section 6.4*, and the slave responds with Sl_SSize[0:1] ‘10’, 128-bit device, then proceeds as a regular one beat per clock transfer.

6.4.2 DDR Read Burst Example

Figure 6-1 DDR Read Burst of 8 Quadwords on page 168 gives an example of a DDR read burst of 8 quadwords. The read-burst request of the master DDR is broadcast by the PLB arbiter in PLB cycle 1 and acknowledged by a slave device in PLB cycle 2 as a DDR transfer. Read quadword data-beat transfers begin in PLB 2xCycle 7 and end in PLB 2xCycle 14. The slave performs an early read-complete operation by driving it read-burst terminated in PLB cycle 6.

For each data acknowledgement response, the actual transfer of data is broken into two quadwords. The first quadword is transferred in the first half of PLB cycle coincident with the rising edge of the Sl_rdDAck signal. The second quadword is transferred on the rising edge of the 2x clock in the PLB 2xCycle.

6.4.3 DDR Write Burst Example

In a DDR write burst of 8 quadwords, the write-burst request of the master DDR is broadcast by the PLB arbiter in PLB cycle 1 and acknowledged by a slave device in PLB cycle 2 as a DDR transfer. For a DDR write burst, the earliest that the first data beat can arrive is the clock cycle after the Sl_addrAck signal, which is in PLB cycle 3. Write quadword data beat transfers begin in PLB 2xCycle 5 and end in PLB 2xCycle 12.

For each data acknowledgement response, the actual transfer of data is broken up into two quadwords. The first quadword is transferred in the first half of PLB cycle coincident with the rising edge of the Sl_rdDAck signal. The second quadword is transferred on the rising edge of the 2x clock in PLB 2xCycle.

Notes:

1. When the master issues a request in PLB cycle 1, it drives only Q1 on its write data bus until the address acknowledgement. This is because the slave can respond with either a standard 128-bit transfer or as a DDR transfer.
2. When the address acknowledgement has occurred and the transfer is a DDR transfer, the master that is performing a DDR write transfer launches the first quadword on the rising edge of a PLB clock and the second quadword on the next 2x clock. In the example in *Figure 6-2 DDR Write Burst of 8 Quadwords* on page 169, this sequence begins on the PLB cycle after the address acknowledgement (cycle 3).

   If the master does not receive a data acknowledgement, the data beats Q1 and Q2, toggles from Q1 to Q2, back to Q1-Q2, until the master receives a write-data acknowledgement from the slave. This is not shown in this example. On the next rising PLB clock edge, Q3 is launched followed by Q4. This is predictable because PLB slaves performing DDR transfers are not allowed to pace. As a power saving measure, the master can start this by toggling the PLB clock after receiving the Sl_addrAck signal.

6.4.4 Read Burst Example of 2-Quadwords

In *Figure 6-2 DDR Write Burst of 8 Quadwords* on page 169, a DDR read burst of 2 quadwords is given. A DDR burst of only two quadwords behaves as a regular one beat per clock-cycle transfer in that the PLB master does not assert its burst signal. This is because there is only one data acknowledgement from the slave that is performing the DDR transfer.

Note: This transfer shows a special case that the requesting PLB master must handle for a DDR request of two quadwords, the minimum numbers of quadwords allowed for a DDR request. This is because the master must to assert its Mx_rdBurst signal in case the slave device responds as a non-DDR device and thus provides two data acknowledgements for two regular one beat per clock-cycle transfers to satisfy the request of two quadwords. If the slave device responds as a DDR entity, the master must immediately deassert its Mx_rdBurst signal at the end of the address acknowledgement because the slave device issues one data acknowledgement for the two DDR quadwords. A DDR write burst request of two quadwords has the same requirement for the handling of the Mx_wrBurst signal. In the example, both the PLB_rdBurst signal and Mx_rdBurst signal are shown to highlight this. The arbiter only routes the Mx_rdBurst signal out on the bus at the end of the address acknowledgement.

Figure 6-1. DDR Read Burst of 8 Quadwords

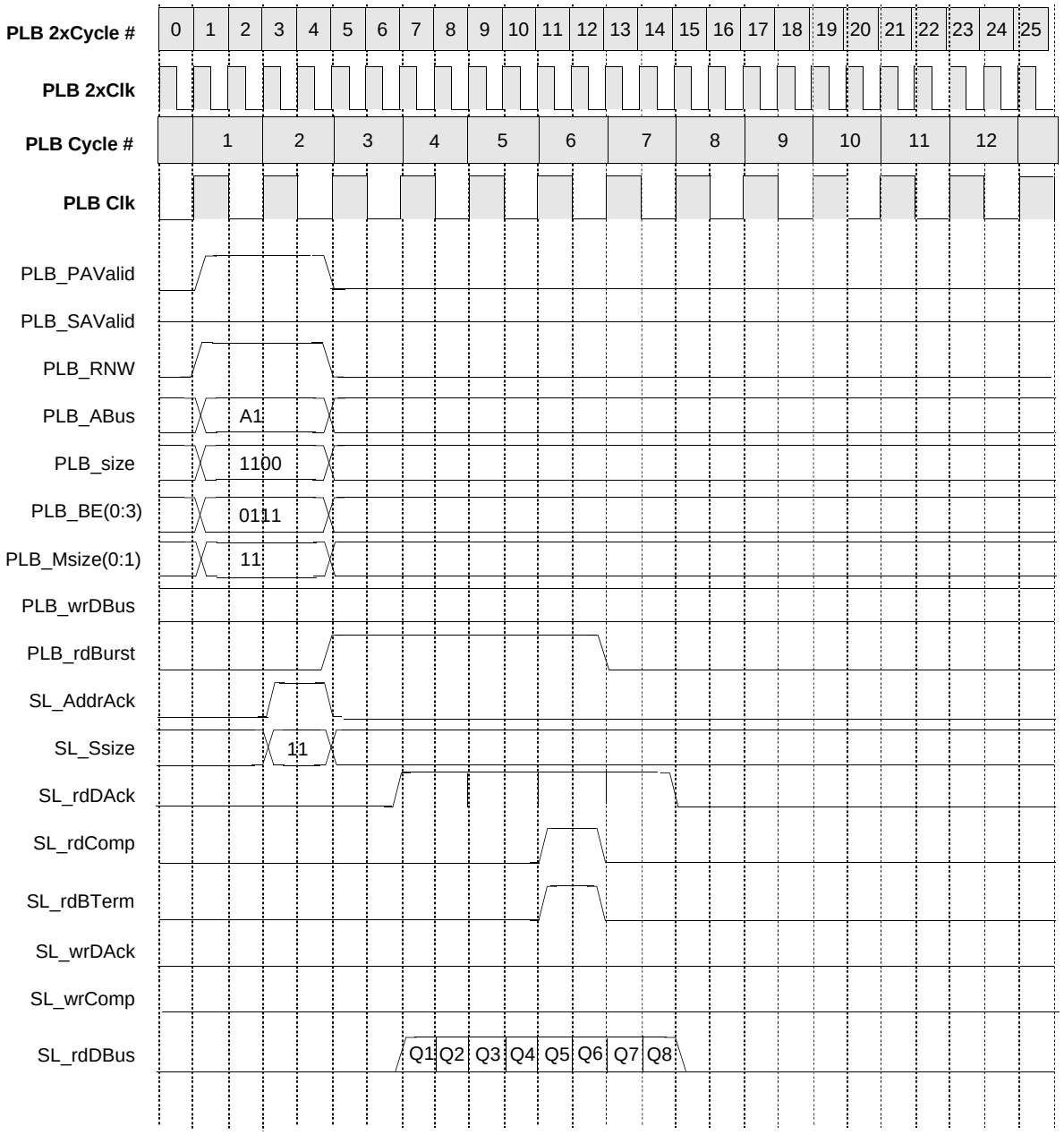


Figure 6-2. DDR Write Burst of 8 Quadwords

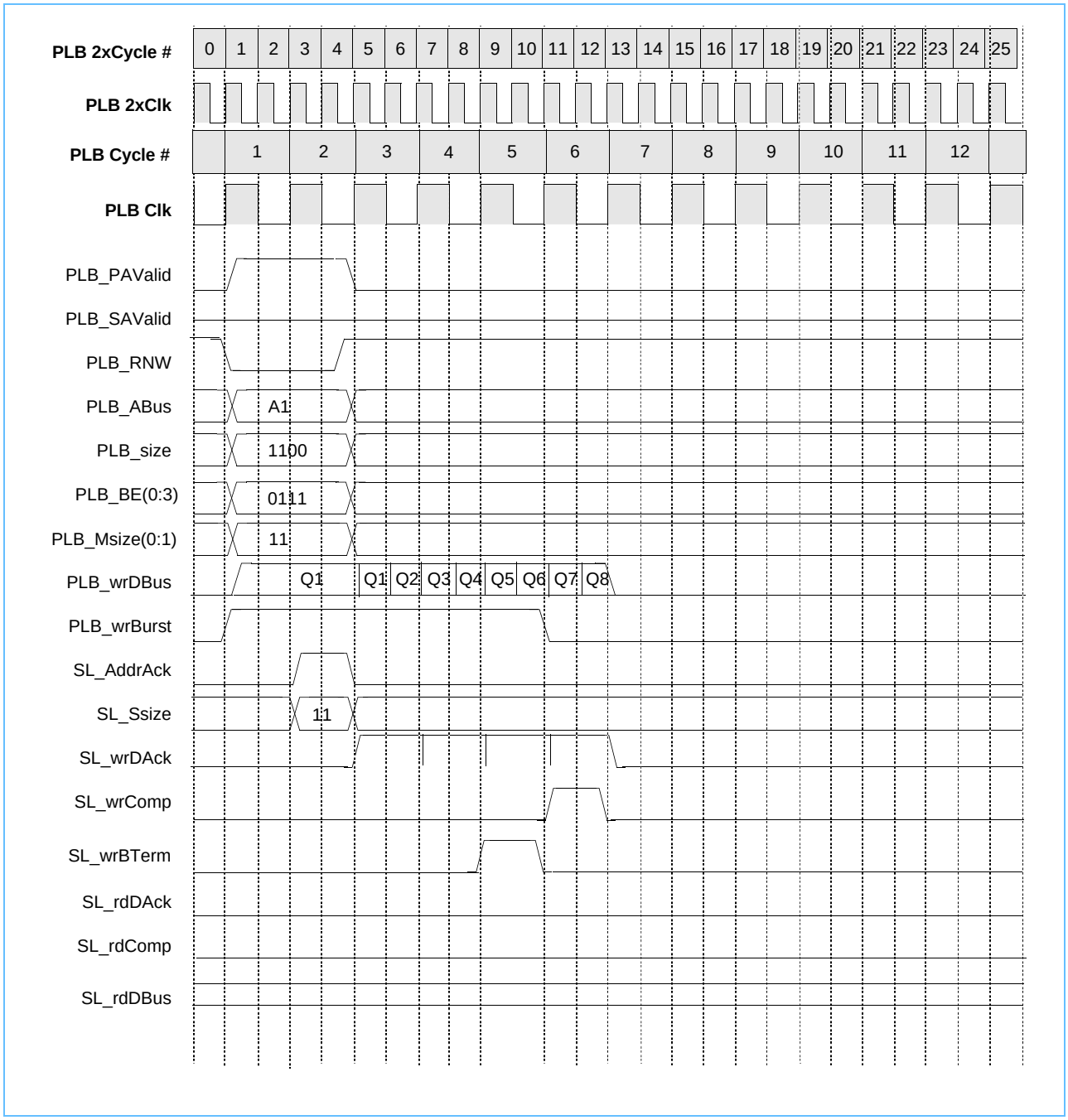
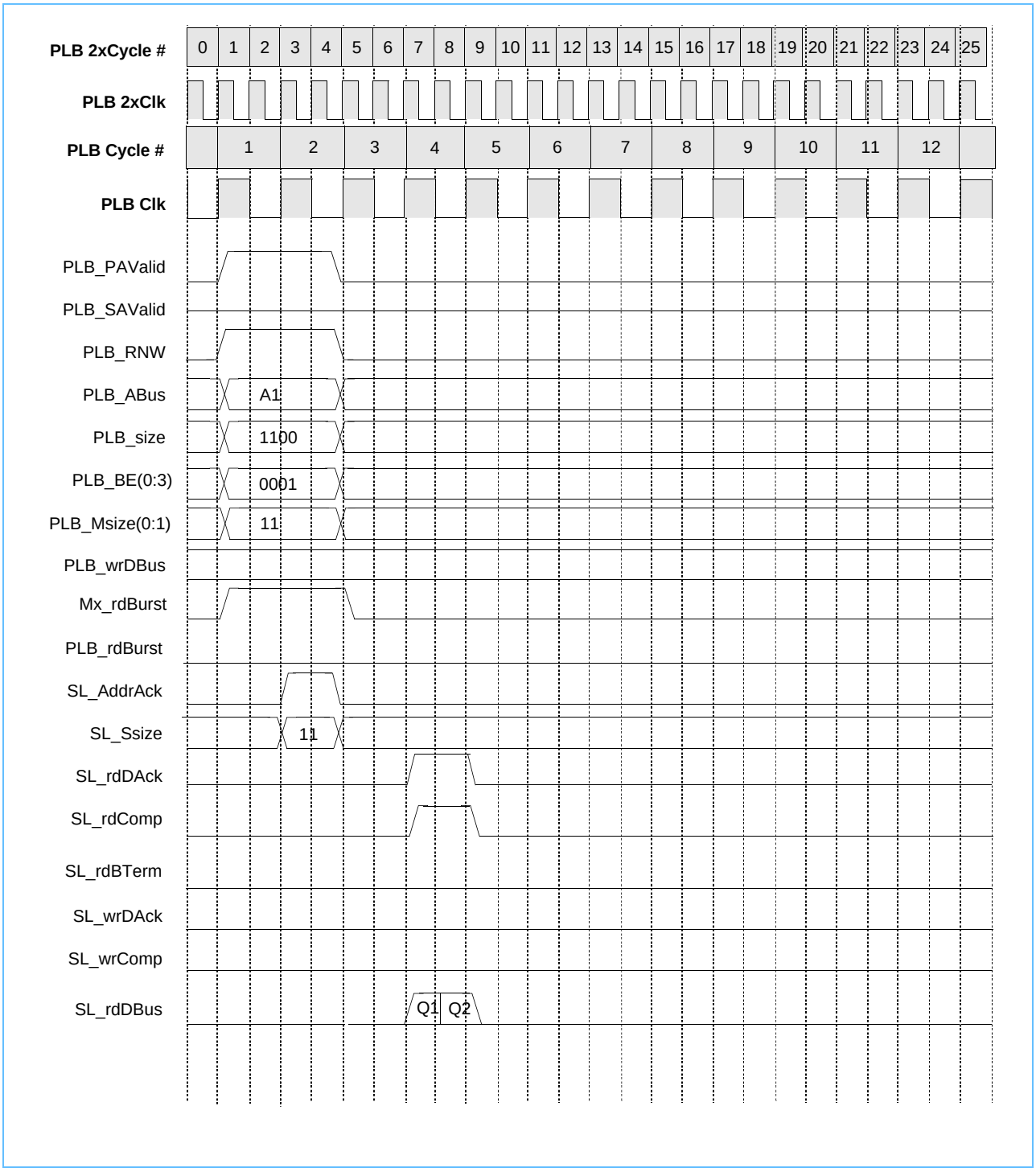


Figure 6-3. DDR Read Burst of 2 Quadwords



Index

Numerics

B

C

D

F

L

M

N

O

P

R

S

T

U, V, W