PCI-X Addendum to the PCI Local Bus Specification

Revision 1.0b

July 29, 2002

REVISION	REVISION HISTORY	DATE
1.0	Initial release.	9/22/99
1.0a	Clarifications and typographical corrections.	7/24/00
1.0b	Clarifications and errata.	7/29/02

PCI-SIG disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does PCI-SIG make a commitment to update the information contained herein.

Contact the PCI-SIG office to obtain the latest revision of the specification.

Questions regarding the PCI-X Addendum or membership in PCI-SIG may be forwarded to:

Membership Services
5440 SW Westgate Drive,
Suite 217
Portland, OR 97221 USA
(administration@pcisig.com)
Phone: 503-291-2569
Fax: 503-297-1090

PCI SIG Specification Distribution
5440 SW Westgate Drive,
Suite 217
Portland, OR 97221 USA
1-800-433-5177 (Domestic Only)
(425) 803-1191 (International)
(503) 222-6190 (Fax)

DISCLAIMER

This PCI-X Addendum is provided "as is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. PCI-SIG disclaims all liability for infringement of proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Contents

Figures

Tables

Preface

Since the introduction of the 66 MHz timing parameters of the *PCI Local Bus Specification* in 1994, bandwidth requirements of peripheral devices have steadily grown. Devices are beginning to appear on the market that support either a 64-bit bus, 66 MHz clock frequency or both, with peak bandwidth capabilities up to 533 MB/s. Because of faster I/O technologies such as Gigabit Ethernet, Ultra 3 SCSI, and Fibre Channel, faster system-interconnect buses are required.

When an industry outgrows a widely accepted standard, that industry decides whether to replace the standard or to enhance it. Since the release of the first *PCI Local Bus Specification* in 1992, the PCI bus has become ubiquitous in the consumer, workstation, and server markets. Its success has been so great that other markets such as industrial controls, telecommunications, and high-reliability systems have leveraged the specification and the wide availability of devices into specialty applications. Clearly, the preferred approach to moving beyond today's *PCI Local Bus Specification* is to enhance it.

PCI-X enables the design of systems and devices that can operate at speeds significantly higher than today's specification allows. Just as importantly, it provides backward compatibility by allowing devices to operate at conventional PCI frequencies and modes when installed in conventional systems. Devices can be designed to meet PCI-X requirements and operate as conventional 33 MHz and 66 MHz PCI devices when installed in those systems. Similarly, if conventional PCI devices are installed in a bus capable of PCI-X operation, the clock remains at a frequency acceptable to the conventional device, and other devices on that bus are restricted to using conventional protocol. This high degree of backward compatibility enables the easy migration of systems and devices to bandwidths in excess of 1 GB/s.

Future Changes

Following publication of the *PCI-X Addendum, Revision 1.0*, there may be future approved errata, clarification, and/or modifications to this specification, prior to the issuance of another formal revision. To assure designs meet the latest level requirements, designers of PCI-X devices must refer to the PCI SIG web site at http://www.pcisig.com for the approved changes.

1. Introduction

The *PCI-X Addendum, Revision 1.0* defines enhancements to the *PCI Local Bus Specification, Revision 2.2* (PCI 2.2), the *PCI to PCI Bridge Architecture Specification, Revision 1.1* (Bridge 1.1), the *PCI Power Management Interface Specification, Revision 1.1* (PCI PM 1.1), and the *PCI Hot-Plug Specification, Revision 1.0* (PCI HP 1.0), which are the latest versions of these specifications at the time of release of this document. The reader is advised to contact the PCI Special Interest Group for any later revisions.

The PCI-X definition introduces several major enhancements that enable faster and more efficient data transfers:

1. Higher clock frequencies up to 133 MHz[1].
2. Signaling protocol changes to enable registered outputs and inputs, that is, device outputs that are clocked directly out of a register and device inputs that are clocked directly into a register. The protocol is restricted such that devices have two clocks to respond to any input changes.
3. New information passed with each transaction that enables more efficient buffer management schemes.
    - Each transaction in a Sequence (see definition in Section 1.2) identifies the total number of bytes remaining to be read or written. If a transaction is disconnected, the new transaction that continues the Sequence contains an updated remaining byte count.
    - Each transaction includes the identity of the initiator (bus number, device number, and function number) and the transaction sequence (or “thread”) to which it belongs (Tag).
    - Additional information about transaction ordering and cacheability requirements.
4. Restricted wait state and disconnection rules optimized for more efficient use of the bus and memory resources.
    - Initiators are not permitted to insert wait states.
    - Targets are not permitted to insert wait states after the initial data phase.
    - Both initiators and targets are permitted to end a burst transaction only on naturally aligned 128-byte boundaries. This encourages longer bursts and enables more efficient use of cacheline-based resources such as the host bus and main memory. Targets are also permitted to disconnect transactions after only a single data phase in address ranges where longer transactions are not necessary.
5. Delayed Transactions in conventional PCI replaced by Split Transactions in PCI-X. All transactions except memory write transactions must be completed immediately or they must be completed using Split Transaction protocol. In Split Transaction protocol, the target terminates a transaction by signaling Split Response, executes the command, and initiates its own Split Completion transaction to send the data or a completion message back to the original initiator.

[1] This specification follows the precedent of PCI 2.2 by abbreviating clock frequency notation. For example, 133 1/3 MHz (the actual maximum frequency for PCI-X devices) is written “133 MHz,” 66 2/3 MHz is written “66 MHz” and 33 1/3 MHz is written “33 MHz.” Actual clock frequencies are specified in Section 9.4.1.

6. A wider range of error recovery implementations for PCI-X devices that reduce system intervention on data parity errors.

PCI-X requirements are defined in many cases to be the same or similar to their corresponding conventional PCI requirements. This simplifies the task of converting conventional designs to PCI-X and of making PCI-X devices that work in conventional environments.

In most cases, this document does not repeat specifications that remain unchanged from PCI 2.2, Bridge 1.1, PCI PM 1.1, and PCI HP 1.0. Any requirement not specified to be different for PCI-X devices remains the same as specified in these other specifications.

The following PCI-X features are some of the features that are the same as conventional PCI:

1. Devices have either a 32- or 64-bit data bus.
2. Address and data are multiplexed on the same bus.
3. Transactions have one or two address phases.
4. Transactions have one or more data phases.
5. Devices decode address and command and assert **DEVSEL#** to claim a transaction.
6. Add-in card mechanical specification.
7. Signal names and connector pin-out (except one new pin, **PCIXCAP**).
8. Power supply voltages.
9. Maximum power consumption for add-in cards.
10. Single clock signal for all devices and transactions. The bus changes state and data transfers on the rising edge of the clock.
11. PCI hot-plug architecture and hot-insertion and hot-removal sequences.

The following PCI-X features have been kept similar to conventional PCI:

1. Signaling protocol on **FRAME#**, **IRDY#**, **DEVSEL#**, **TRDY#**, and **STOP#**.
2. Data phases complete each time **IRDY#** and **TRDY#** are both asserted. Some additional target data-phase signaling is defined for PCI-X.
3. Transaction ordering and passing rules for bridges (Split Transactions in PCI-X replace Delayed Transactions in conventional PCI).
4. Electrical signaling voltage levels are the same, except V_{il}(max) is slightly higher in PCI-X mode to provide additional noise margin. PCI-X add-in cards are keyed for 3.3V or Universal signaling.
5. Device and add-in card electrical specification ranges are generally narrower for PCI-X devices.

1.1. Documentation Conventions

In addition to the documentation conventions established in PCI 2.2, the following conventions are used in this document:

Capitalization

Names of transaction commands and target termination methods are presented with the first letter capitalized and the rest lower case, e.g., Memory Read Block, Memory Write Block, Retry, Target-Abort, and Single Data Phase Disconnect.

As in PCI 2.2, register names and the names of fields and bits in registers and attributes are presented with the first letter capitalized and the rest lower case, e.g., PCI-X Status register, PCI-X Command register, Byte Count field, Bus Number field, and No Snoop attribute bit.

Some other terms are capitalized to distinguish their definition in the context of this document from their common English meaning. These terms are listed in Section 1.2.

Words not capitalized have their common English meaning. When terms such as "memory write" or "memory read" appear completely in lower case, they include all transactions of that type. For example, transactions using the Memory Write, Memory Write Block, and Alias to Memory Write Block commands are all included by the phrase, "memory write transactions."

Numbers and number bases

Hexadecimal numbers are written with a lower case "h" suffix, e.g., FFFFh and 80h. Hexadecimal numbers larger than four digits are represented with a space dividing each group of four digits, as in 1E FFFF FFFFh.

Binary numbers are written with a lower case "b" suffix, e.g., 1001b and 10b. Binary numbers larger than four digits are represented with a space dividing each group of four digits, as in 1000 0101 0010b.

All other numbers are decimal.

Buses

As in PCI 2.2, collections of signals that are collectively driven and received are assigned the same signal name with numbers in brackets to indicate the bit or bits affected, e.g., **AD[31::00]**, **C/BE[7::4]#**, and **AD[2]**.

Reference information

Reference information is provided in various places to assist the reader and does not represent a requirement of this document. Such references are indicated by the abbreviation "(ref)." For example, in some places a clock that is specified to have a minimum period of 15 ns also includes the reference information maximum clock frequency of "66 MHz (ref)."

Requirements of other specifications also appear in various places throughout this document and are marked as reference information. Every effort has been made to guarantee that this information accurately reflects the referenced document. However, in case of discrepancy, the original document takes precedence.

Device types

As in conventional PCI, PCI -X devices are required to operate up to a maximum frequency (down to a minimum clock period). The system is permitted to operate the bus at a lower frequency to compensate for additional bus loading. This document refers to the type of the device as either conventional or PCI-X and the appropriate maximum frequency.

Conventional PCI 33

Conventional PCI 66

PCI-X 66

PCI-X 133

In all cases, the actual operating clock frequency is between the minimum and maximum specified for that device.

1.2. Terms and Abbreviations

The following terms and abbreviations are used throughout this specification:

address order

Incrementing sequentially beginning with the starting address of the Sequence. For example, Split Completions in the same Sequence (that is, resulting from a single Split Request) must be returned in address order.

allowable disconnect boundary (ADB)

A naturally aligned 128-byte boundary. Initiators and targets are permitted to disconnect burst transactions only on ADBs. (See Section 2.2 for more information.)

ADB delimited quantum (ADQ)

A portion (or all) of a transaction or a buffer that fits between two adjacent ADBs. For example, if a transaction starts between two ADBs, crosses one ADB, and ends before reaching the next ADB, the transaction includes two ADQs of data. Such a transaction fits in two buffers inside a device that divides its buffers on ADBs.

application bridge

A device that implements internal posting of memory write transactions that the device must initiate on the PCI-X interface but uses a Type 00h Configuration Space header and the Class Code of the application it performs rather than that of a bridge. See Section 8.2.

attribute

The 36-bit field contained on **C/BE[3::0]#** and **AD[31::00]** during the attribute phase of a PCI-X transaction. Used for further definition of the transaction.

attribute phase

The clock after the address phase(s). The lower bus halves (**C/BE[3::0]#** and **AD[31::00]**) contain the attributes. The upper bus halves (**C/BE[7::4]#** and **AD[63::32]**) are reserved and driven high by 64-bit initiators.

burst transaction A transaction using one of the following commands:

Memory Read Block
Memory Write Block
Memory Write
Alias to Memory Read Block
Alias to Memory Write Block
Split Completion

Burst transactions can be of any length, from 1 to 4096 bytes. (Note that if the byte count is small enough, a burst transaction contains only a single data phase.) They are permitted to be initiated both as 64-bit and 32-bit transactions.

The **C/BE#** bus contains explicit byte enables for each data phase of Memory Write transactions. For all other burst transactions, the **C/BE#** bus is reserved and driven high during all data phases.

byte count The number of bytes to be included in a Sequence. It appears in the attribute phase of all burst transactions and indicates the number of bytes affected by the transaction. (Byte enables must also be asserted for a byte to be affected by a Memory Write transaction. See Section 2.6.1.)

completer The device addressed by a transaction (other than a Split Completion).

Completer Attributes Format of the attributes of all Split Completion transactions. Includes information about the completer and the Sequence. See also Requester Attributes.

Completer ID The combination of a completer’s bus number, device number, and function number. All these numbers appear in the attribute phase of every Split Completion and uniquely identify the completer of the transaction. See also Requester ID.

In most cases, a PCI-X bridge forwards Split Completion transactions from one interface to another without modifying the Completer ID. A bridge from a bus other than PCI-X (including a PCI bus operating in conventional mode) must create its own Completer ID when creating a Split Completion transaction.

data phase Each clock in which the target signals some kind of data transfer or terminates the transaction. Transactions terminated with Split Response, Single Data Phase Disconnect, or Retry have a single data phase. Clocks in which the target signals Wait State one or more times and then signals something else are part of the same data phase.

disconnection The termination of a burst transaction after some but not all of the byte count has been satisfied. In other words, disconnection is the termination of the transaction but not the termination of the Sequence (see Section 2.1 for some exceptions). Targets are permitted to disconnect any transaction after a single data phase by signaling Single Data Phase Disconnect (see Section 2.11.2.1). Targets are also permitted to disconnect on any ADB by signaling Disconnect at Next ADB (see Section 2.11.2.2). Initiators are permitted to disconnect on any ADB four or more data phases from the starting address by deasserting **FRAME#** two clocks before the ADB (see Section 2.11.1.1).

device A component of a PCI system that connects to a PCI bus. As defined by PCI 2.2, a device can be a single function or a multifunction device. All devices must be capable of responding as a target to some transactions on the bus. Many devices are also capable of initiating transactions on the bus. A PCI-X device also supports the requirements of this document.

As in PCI 2.2, the term “device” is often used when describing requirements that apply individually to all functions within the device. Unless otherwise specified, requirements in the PCI-X definition for a device apply to single function devices and to each function individually of a multifunction device.

device boundary The first address of a device range or the first address beyond the end of a device range. Address ranges for devices with Type 00h Configuration Space headers are established with Base Address registers, as described in PCI 2.2. Address ranges for devices with Type 01h Configuration Space headers (bridges) are established with Base Address, Memory Base, I/O Base, and Prefetchable Memory Base registers, as described in Bridge 1.1.

A device boundary is always an ADB. (See Section 7.1.) To “disconnect at a device boundary” means to disconnect a transaction in such a way that the last address of the transaction corresponds to the last address of the device. To “cross a device boundary” means that the transaction includes one or more addresses of the devices on both sides of the boundary.

drive When a device has acquired exclusive ownership of a bus through a handshake with the bus arbiter or the initiator of a transaction, the device is said to drive the bus by placing its output buffers in a low impedance state to put the bits of the bus in the appropriate logic level.

DWORD Thirty-two bits of data on a naturally aligned four-byte boundary (i.e., the least significant two bits of the address are 00b).

DWORD transaction

A transaction using one of the following commands:

Interrupt Acknowledge
Special Cycle
I/O Read
I/O Write
Configuration Read
Configuration Write
Memory Read DWORD

DWORD transactions address no more than a single DWORD and are permitted to be initiated only as 32-bit transactions (**REQ64#** must be deasserted). During the attribute phase, the Requester Attributes contain valid byte enables. During the data phase, the **C/BE#** bus is reserved and driven high by the initiator.

ending address

Last address included in the Sequence. For burst transactions, it is calculated by adding the starting address and the byte count and subtracting one and is permitted to be aligned to any byte. For DWORD transactions, it is the last byte (**AD[1::0]** = 11b) of the DWORD addressed by the starting address.

float

When a device has finished driving a bus or a control signal and it places the output buffers in the high-impedance state, the device is said to float the bus or the control signal.

Immediate Transaction

A transaction that terminates in a way that includes transferring data or that terminates with an error that completes the Sequence. Transactions in which the target signals Data Transfer, Single Data Phase Disconnect, Disconnect at Next ADB, Master-Abort, or Target-Abort are Immediate Transactions. Transactions that terminate with Retry or Split Response are not Immediate Transactions.

initiator

A device that initiates a transaction by requesting the bus, asserting **FRAME#**, and driving the command and address. A bridge forwarding a transaction is an initiator on the destination bus.

PCI-X bridge

Unless otherwise specified, a bridge between two buses that are capable of operating in PCI-X mode. If a conventional PCI device is connected to a general-purpose PCI-X bridge interface, that interface must operate in conventional mode.

PCI-X initialization pattern

A combination of bus control signals that is used to place PCI-X devices in PCI-X mode at the rising edge of **RST#**. Also indicates the range of frequency of the clock. See Section 6.2.

QWORD

Sixty-four bits of data on a naturally aligned eight-byte boundary (i.e., the least significant three bits of the address are 000b).

read side effects

Changes to the state of a device that occur if a location within the device is read; for example, the clearing of a status bit or advancing to the next data value in a sequential buffer.

requester Initiator that first introduces a transaction into the PCI-X domain. If the completer or a bridge terminates the transaction with Split Response, the requester becomes the target of the subsequent Split Completion.

A PCI-X bridge is required to terminate all transactions that cross it (except for memory write transactions) with Split Response. The bridge forwards the request toward the completer and forwards the completion in the opposite direction. A bridge from some domain other than PCI-X (including conventional PCI) becomes the requester for the transaction in the PCI-X domain. If the device that originated the request in the other domain is referred to as a “requester,” the term must include a modifier such as “conventional requester” to avoid confusion with the bridge.

Requester Attributes Attributes of all transactions except Split Completion transactions. Includes information about the requester and the Sequence. There are different Requester Attribute formats for burst, DWORD, and Type 0 configuration transactions. See also Completer Attributes.

Requester ID The combination of a requester’s bus number, device number, and function number. All these numbers appear in the attribute phase of every transaction except Split Completions. They appear in the address phase of a Split Completion. The Requester ID uniquely identifies the requester of the transaction. See also Completer ID.

In most cases, a PCI-X bridge forwards transactions from one interface to another without modifying the Requester ID. A bridge from a bus other than PCI-X (including a PCI bus operating in conventional mode) must use its own Requester ID when forwarding a transaction to a bus operating in PCI-X mode.

Sequence One or more transactions associated with carrying out a single logical transfer by a requester. See Section 2.1 for additional details.

Sequence ID The combination of the Requester ID (requester bus number, device number, and function number) and Tag attributes. This combination uniquely identifies transactions that are part of the same Sequence and is used in buffer-management algorithms and some ordering rules.

Sequence size The number of ADQs required for the data of the Sequence. For example, the size of a DWORD Sequence is one ADQ. The size of a Sequence that starts between two ADBs and ends before reaching the third ADB is two ADQs.

Sequence termination The termination of a transaction in such a way that the Sequence does not resume with additional transactions.

source bridge The bridge that creates a bus segment capable of PCI-X operation in a system hierarchy. The source bridge is required to initiate Type 0 configuration transactions on that bus segment. (Other devices optionally initiate Type 0 configuration transactions.) A host bridge is the source bridge for the PCI bus it creates. A PCI-X bridge is the source bridge for its secondary bus.

Split Completion When used in the context of the bus protocol, this term refers to a transaction using the Split Completion command. It is used by the completer to send the requested data (for read transactions completed without error) or a completion message back to the requester.

When used in the context of transaction ordering and the transaction queues inside the requester, completer, and bridges, the term refers to a queue entry corresponding to a Split Completion transaction on the bus.

Split Completion address Information driven by the completer or a bridge on the **AD** bus during the address phase of a Split Completion transaction. The Split Completion address includes the Requester ID and is used to route the Split Completion to the requester.

Split Completion Message In the context of bus transactions, a Split Completion Message is a Split Completion transaction that notifies the requester that a request (either read or write) encountered an error, or that a write request completed without an error.

In the context of the Split Completion address, the term refers to the attribute bit that indicates the Split Completion is a message rather than data for a read request.

Split Request When used in the context of the bus protocol, this term refers to a transaction terminated with Split Response. When used in the context of transaction ordering and the transaction queues inside the requester, completer, and bridges, the term refers to a queue entry corresponding to a Split Request transaction on the bus. When the completer executes the Split Request, it becomes a Split Completion.

Split Response Protocol for terminating a transaction, whereby the target indicates that it will complete the transaction as a Split Transaction. The target may optionally terminate any transaction except a memory write transaction with Split Response.

Split Transaction A single logical transfer containing an initial transaction (the Split Request) that the target (the completer or a bridge) terminates with Split Response, followed by one or more transactions (the Split Completions) initiated by the completer (or bridge) to send the read data (if a read) or a completion message back to the requester.

starting address Address indicated in the address phase of all transactions except Split Completions, Interrupt Acknowledges, and Special Cycles. This is the first byte of the transaction. The starting address of all transactions except configuration transactions is permitted to be aligned to any byte (i.e., it uses the full address bus). The starting address of configuration transactions must be aligned to a DWORD boundary (i.e., **AD[1::0]** must be 0).

Split Completion transactions use only a partial starting address as described in Section 2.10.3. As in conventional PCI, Interrupt Acknowledge and Special Cycle transactions have no address.

Tag

A 5-bit number assigned by the initiator of a Sequence to distinguish it from other Sequences. It appears in the attribute field and is part of the Sequence ID.

An initiator must not reuse a Tag for a new Sequence until the original Sequence is complete (i.e., byte count satisfied, error condition encountered, etc.). (See Section 2.1 for more details.)

target

A device that responds to bus transactions. A bridge forwarding a transaction is the target on the originating bus.

target data phase signaling

The target signals one of the following on each data-phase clock:

Split Response
Target-Abort
Single Data Phase Disconnect
Wait State
Data Transfer
Retry
Disconnect at Next ADB

target response phase

One or more clocks after the attribute phase until the target claims the transaction by asserting **DEVSEL#** (also used but not named in conventional PCI).

transaction

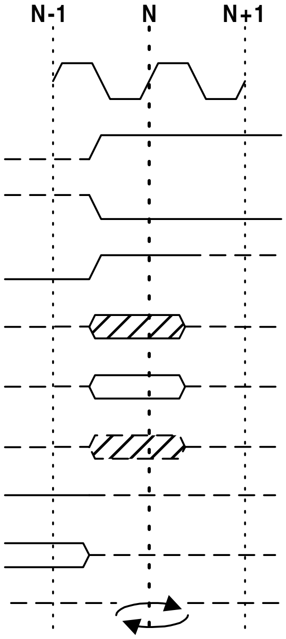
A combination of address, attribute, target response, data, and bus turn-around phases associated with a single assertion of **FRAME#**.

transaction termination

The protocol for ending a transaction either by the initiator or the target.

1.3. Figure Legend

The following conventions for the state of a signal at clock N are used in timing diagrams throughout this document.



system clock rising edge triggered

driven high signal

driven low signal

low signal state being driven high before being floated

driven signal to a valid but unspecified state

driven signal to a specified state

driven signal to a valid but unspecified state or floated

floated signal, high state maintained by system pull-up resistors

floated signal, state unknown

reserved for signal or bus turn-around. Neither owner drives the signal or bus on this clock. The time at which the previous owner stops driving and the next owner starts driving is shown by the state of the signal or bus in previous and subsequent clocks, respectively.

1.4. Transaction Comparison Between PCI-X and Conventional PCI

The following illustrations compare a typical conventional PCI write transaction with a typical PCI-X write transaction. Both illustrations depict a write transaction with initiator termination, identical device selection timing and wait states, and six data phases.

Figure 1-1 shows a typical conventional PCI write transaction with six data phases. The initiator and target insert minimum wait states for **DEVSEL#** timing “medium,” and the transaction completes in nine clocks including the bus turn-around.

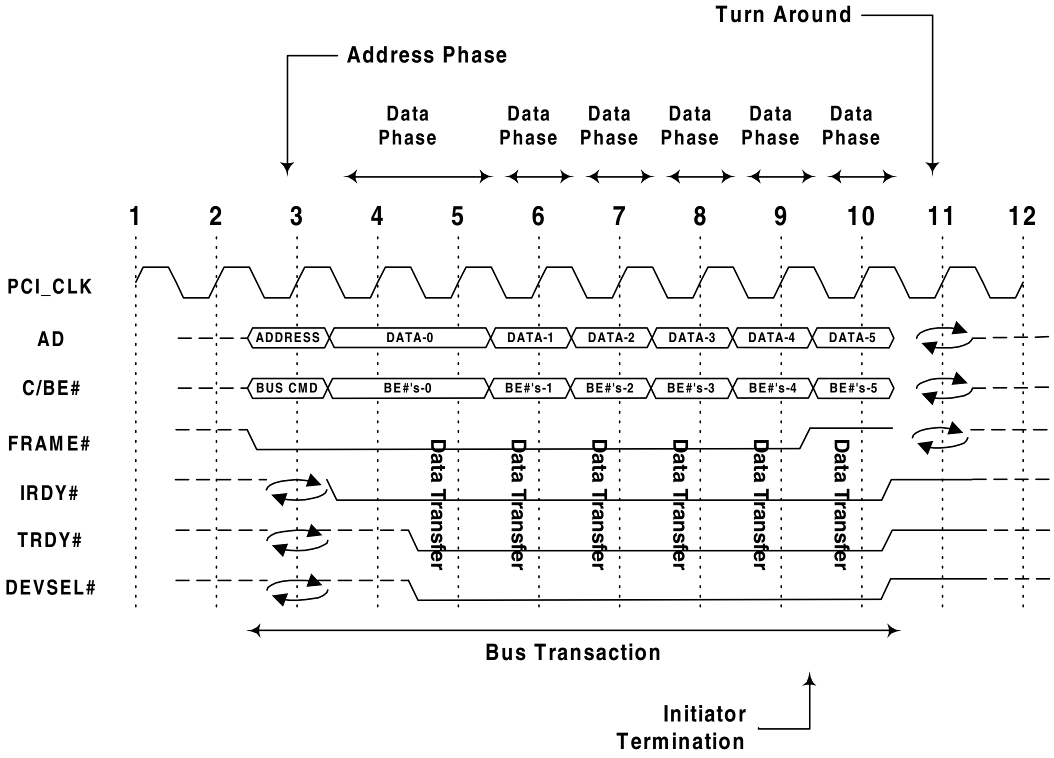


Figure 1-1: Typical Conventional PCI Write Transaction

Table 1-1 lists the phase definitions for conventional PCI bus transactions as shown in Figure 1-1.

Table 1-1: Conventional PCI Transaction Phase Definitions

Conventional PCI Phases	Description
Address Phase	One clock for single address cycle, two clocks for dual address cycle.
Data Phase	The clocks after the address phase in which the target inserts wait states, transfers data, or signals the end of the transaction.
Initiator Termination	Initiator signals the end of the transaction on the last data phase.
Turn-Around	Idle clock for changing from one signal driver to another.

Figure 1-2 shows a typical PCI-X write transaction with six data phases. The initiator and target insert minimum wait states for **DEVSEL#** timing A, and the transaction completes in ten clocks including the bus turn-around.

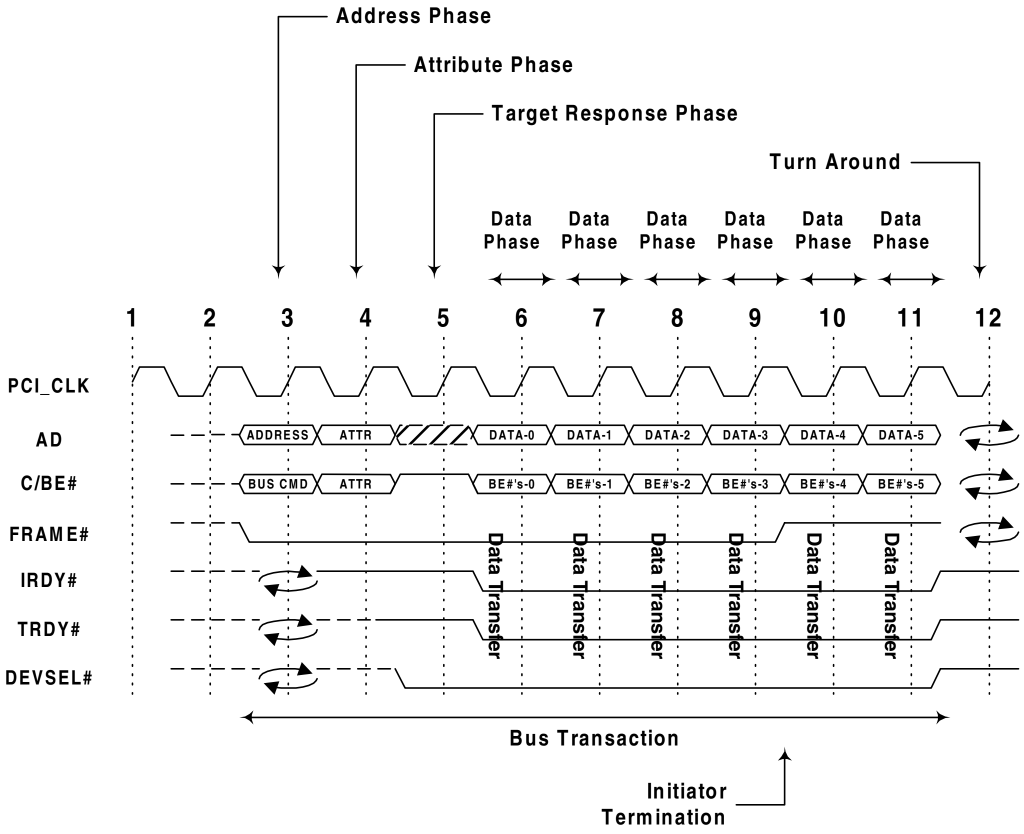


Figure 1-2: Typical PCI-X Write Transaction

The two previous figures illustrate the similarities between conventional PCI and PCI-X. The protocol differences have been kept to a minimum to lessen the effect of the change on designs and on tools for design and debug.

The figures also show the effect of PCI-X protocol on the length of the transaction. Both transactions show the target responding by asserting **DEVSEL#** two clocks after **FRAME#** and moving a total of six data phases. The transaction takes nine clocks for conventional PCI and ten clocks for PCI-X.

Table 1-2 lists the transaction phases of PCI -X as shown in Figure 1-2.

Table 1-2: PCI-X Transaction Phase Definitions

PCI-X Transaction Phases	Description
Address Phase	One clock for single address cycle, two clocks for dual address cycle. (See Section 2.12.1 for more information about dual address cycles.)
Attribute Phase	One clock. This phase provides further information about the transaction.
Target Response Phase	One or more clocks after the attribute phase until the target claims the transaction by asserting **DEVSEL#** (also used but not named in conventional PCI).
Data Phase	The clocks after the target response phase in which the target transfers data or signals the end of the transaction.
Initiator Termination	Initiator signals the end of the transaction one clock before the last data phase.
Turn-Around	Idle clock for changing from one signal driver to another.

1.5. Burst and DWORD Transactions

Like conventional PCI, PCI-X supports transactions with one or more data phases. Transactions using the memory commands (except Memory Read DWORD) are called burst transactions and are permitted to have any number of data phases (from one to the maximum necessary to satisfy the byte count). The other PCI-X commands are limited to a single data phase (an aligned DWORD or less) and are called DWORD transactions. DWORD transactions are limited to a 32-bit transaction width (**REQ64#** must be deasserted).

Table 1-3 compares burst and DWORD transactions.

Table 1-3: Comparison of Burst Transactions and DWORD Transactions

Burst Transactions	DWORD Transactions
Commands: Memory Read Block Memory Write Block Memory Write Alias to Memory Read Block Alias to Memory Write Block Split Completion	Commands: Interrupt Acknowledge Special Cycle I/O Read I/O Write Configuration Read Configuration Write Memory Read DWORD
64- or 32-bit data transfers.	32-bit data transfers only.
Starting address specified on **AD** bus down to a byte address (includes all **AD** bits).	Starting address specified on **AD** bus down to a byte address (includes all **AD** bits), except for configuration transactions, which are DWORD aligned (**AD[1::0]** indicate configuration transaction type).
Supports one or more data phases and always in address order.	Supports only single data phase.
During the data phases, the **C/BE#** bus is reserved and driven high by the initiator for all transactions except Memory Write. The **C/BE#** bus contains valid byte enables for Memory Write transactions. Any byte enable pattern is permitted (between the starting and ending address, inclusive), including no byte enables asserted.	During the attribute phase, the Requester Attributes contain valid byte enables. Any byte enable pattern is permitted, including no byte enables asserted. During the data phase, the **C/BE#** bus is reserved and driven high by the initiator.

1.6. Wait States

PCI-X initiators are not permitted to insert wait states on any data phase. PCI-X targets are permitted to insert wait states on the initial data phase only. No wait states are allowed on subsequent data phases. Target initial wait states for memory write and Split Completion transactions must come in pairs for transactions that successfully transfer data. See Section 2.9 for details.

1.7. Split Transactions

Split Transactions in PCI-X replace Delayed Transactions in conventional PCI. All transactions except memory write transactions are permitted to be executed as Split Transactions. If a target cannot complete a transaction (other than a memory write transaction or a Split Completion) within the target initial latency limit, the target must complete that transaction as a Split Transaction. (See Section 2.13 for exceptions when the target is permitted to signal Retry.) If the target meets the target initial latency limits, the target optionally completes the transaction immediately (not as a Split Transaction).

A Split Transaction starts when an initiator (called the requester) initiates a transaction other than a memory write or Split Completion (called the Split Request). If a target (called the completer) chooses to complete the transaction as a Split Transaction, it signals Split Response termination by using the signaling handshake shown in

Section 2.11.2.4. The completer then executes the transaction. The completer then asserts its **REQ#** signal to request the bus. When the arbiter asserts **GNT#** to the completer, the completer initiates a Split Completion transaction to send read data (at least up to an ADB) or a completion message to the requester. Notice that for a Split Completion transaction, the requester and the completer switch roles. The completer becomes the initiator of the Split Completion transaction and the requester becomes the target.

1.8. Bus Width

The following PCI-X bus width requirements are the same as conventional PCI:

1. Devices are permitted to support either a 64-bit interface or a 32-bit interface.
2. Addresses greater than 4 GB require a dual address cycle regardless of the width of the devices.
3. Data transfer width is negotiated for each transaction. The initiator and target are permitted to be either size and the transaction proceeds at the width of the smaller.
4. The state of **REQ64#** at the rising edge of **RST#** indicates the width of the bus.

The following PCI-X requirements are different from conventional PCI:

1. All devices that initiate memory transactions must be capable of generating 64-bit addresses.
2. Each device includes a configuration status bit that indicates the width of that device's interface.

1.9. Compatibility and System Initialization

This document defines two frequency ranges of PCI-X devices, PCI-X 66 and PCI-X 133. Both kinds of devices have identical requirements except for electrical differences specified in Section 9, such as the minimum clock period (maximum clock frequency), and the add-in card identification requirements. PCI-X 66 devices operate in PCI-X mode with clock frequencies from 50 MHz to 66 MHz. PCI-X 133 devices operate in PCI-X mode with clock frequencies from 50 MHz to 133 MHz. If only PCI-X 133 devices are installed on a bus, the bus operates in PCI-X mode and the clock operates up to 133 MHz. If all devices on the bus are PCI-X 133 and PCI-X 66, the bus operates in PCI-X mode and the clock operates up to 66 MHz.

All PCI-X devices and systems also support conventional PCI 33 mode. They optionally support conventional PCI 66 mode.

PCI-X devices initialize in conventional or PCI-X mode depending on the state of a combination of bus control signals (called the PCI-X initialization pattern and shown in Table 6-2) at the rising edge of **RST#**, as described in Section 6.2. One pattern causes the device to enter conventional mode. Other patterns cause it to enter PCI-X mode.

The source bridge for each bus drives the PCI-X initialization pattern during **RST#**. The host bridge that begins a PCI bus hierarchy and a PCI-X bridge that extends it have slightly different requirements that are presented separately in Section 6.2.3.1 and Section 8.9 respectively.

The PCI Hot-Plug Controller must provide the PCI-X initialization pattern on the bus during a hot-insertion event as described in PCI HP 1.0. Coordination between the PCI

Hot-Plug Controller and the source bridge for asserting the pattern is permitted, but the details of such an implementation are beyond the scope of this specification.

1.10. Summary of Protocol Rules

Protocol rules are divided into the following categories:

- General bus rules
- Initiator rules
- Target rules
- Bus arbitration rules
- Configuration transaction rules
- Parity error rules
- Bus width rules
- Split Transaction rules

The following sections summarize the protocol rules for PCI-X transactions according to these categories. Later subsections of this document describe details of these rules.

1.10.1. General Bus Rules

The following rules generally apply to all transactions:

1. As in conventional PCI, the first clock in which **FRAME#** is asserted is the address phase. In the address phase, the **AD** bus contains the starting address (except Split Completion, Interrupt Acknowledge, or Special Cycle) and the **C/BE#** bus contains the command. (See Section 2.12.1 for dual address cycles.)
2. Except as listed below, the starting address of all transactions is permitted to be aligned to any byte. As in conventional PCI, the starting address of Configuration Read and Configuration Write transactions is aligned to a DWORD boundary. Split Completion transactions use only a partial starting address as described in Section 2.10.3. As in conventional PCI, Interrupt Acknowledge and Special Cycle transactions have no address.
3. The attribute phase follows the address phase(s). **C/BE[3::0]#** and **AD[31::00]** contain the attributes. **C/BE[7::4]#** and **AD[63::32]** are reserved and driven high by 64-bit initiators. The attributes include additional information about the transaction.
4. The **C/BE#** bus is reserved (driven high) the clock after the attribute phase.
5. Burst transactions include the byte count in the attributes. The byte count indicates the number of bytes between the first byte of the transaction and the last byte of the Sequence, inclusive.
6. DWORD transactions do not use a byte count.
7. The target response phase is one or more clocks after the attribute phase and ends when the target asserts **DEVSEL#**.
8. As in conventional PCI, there are no data phases if the target does not assert **DEVSEL#**, resulting in a Master-Abort. All other transactions have one or more data phases following the target response phase.

9. As in conventional PCI, transactions using the I/O Read, I/O Write, Configuration Read, Configuration Write, Interrupt Acknowledge, and Special Cycle commands are initiated only as 32-bit transactions (**REQ64#** deasserted). Memory Read DWORD commands also have the same restriction in PCI-X mode. In PCI-X, the length of all these transactions is limited to one data phase. Transactions using the Memory Write, Memory Read Block, Memory Write Block, Alias to Memory Read Block, Alias to Memory Write Block, and Split Completion are permitted by both 64- and 32-bit initiators and are permitted to have one or more data phases, up to the maximum required to satisfy the byte count.

10. As in conventional PCI, data is transferred on any clock in which both **IRDY#** and **TRDY#** are asserted.

11. The following rules apply to the use of byte enables:

    a. Byte enables are included in the Requester Attributes for all DWORD transactions. Byte enables are included on the **C/BE#** bus during the data phases of all Memory Write burst transactions. Byte enables further qualify the bytes affected by the transaction. Only bytes for which the byte enable is asserted are affected by the transaction.

    b. The **C/BE#** bus is reserved and driven high during the single data phase of all DWORD transactions and throughout all data phases of all burst transactions except Memory Write.

    c. DWORD transactions are permitted to have any combination of byte enables, including no byte enables asserted. See Section 2.3 for restrictions on starting address and byte enables.

    d. Memory Write transactions are permitted to have any combination of byte enables between the starting and ending addresses, inclusive. Byte enables must be deasserted for bytes before the starting address and after the ending address (if those addresses are not aligned to the width of the bus). See Section 2.12.3 for exceptions and additional requirements when a 64-bit initiator addresses a 32-bit target.

    e. The byte count of Memory Write transactions is *not* adjusted for bytes whose byte enables are deasserted within the transaction. In other words, the byte count is the same whether all or none of the byte enables were asserted.

12. Device state machines must not be confused by target control signals (**DEVSEL#**, **TRDY#**, and **STOP#**) asserting while the bus is idle (**FRAME#** and **IRDY#** both deasserted). (In some systems, the PCI-X initialization pattern appears on the bus when another device is being hot-inserted onto the bus. See Section 6.2.3.2.)

13. Like conventional PCI, no device is permitted to drive and receive a bus signal at the same time. (See Section 3.1.)

1.10.2. Initiator Rules

The following rules control the way a device initiates a transaction:

1. As in conventional PCI, a PCI-X initiator begins a transaction by asserting **FRAME#**. (See Section 2.7.2.1 for differences for configuration transactions.)

2. In most cases, the initiator asserts **FRAME#** within two clocks after **GNT#** is asserted and the bus is idle. If the transaction uses a configuration command, the initiator must assert **FRAME#** six clocks after **GNT#** is asserted and the bus is idle.

3. The initiator asserts and deasserts control signals as follows:

   a. The initiator asserts **FRAME#** to signal the start of the transaction. It deasserts **FRAME#** on the later of the following two conditions:

      1) one clock before the last data phase

      2) two clocks after the target asserts **TRDY#** (or terminates the transaction in some other way as described in Section 2.11.2)

      The two conditions for the deassertion of **FRAME#** cover two cases discussed in Section 2.11. The first case 1) is if the transaction has four or more data phases. The second case 2) is if the transaction has less than four data phases.

   b. Initiator wait states are not permitted. The initiator asserts **IRDY#** two clocks after the attribute phase. It deasserts it on the later of the following two conditions:

      1) one clock after the last data phase

      2) two clocks after the target asserts **TRDY#** (or terminates the transaction in some other way as described in Section 2.11.2)

      The two conditions for the deassertion of **IRDY#** cover two cases discussed in Section 2.11. The first case 1) is if the transaction has three or more data phases. The second case 2) is if the transaction has less than three data phases.

4. If no target asserts **DEVSEL#** on or before the Subtractive decode time, the initiator ends the transaction as a Master-Abort.

5. For write and Split Completion transactions, the initiator must drive data on the **AD** bus two clocks after the attribute phase. If the transaction is a burst with more than one data phase, the initiator advances to the second data value two clocks after the target asserts **DEVSEL#**, in anticipation of the target asserting **TRDY#**. If the target also inserts wait states, the initiator must toggle between its first and second data values until the target asserts **TRDY#** (or terminates the transaction). See Section 2.12.3 for requirements for a 64-bit initiator writing to 32-bit targets.

6. The initiator is required to terminate the transaction when the byte count is satisfied.

7. The initiator is permitted to disconnect a burst transaction (before the byte count is satisfied) only on an ADB. If the initiator intends to disconnect the transaction on the first ADB, and the first ADB is less than four data phases from the starting address, the initiator must adjust the byte count to terminate the transaction on that ADB.

8. If a burst transaction would otherwise cross the next ADB, and the target signals Disconnect at Next ADB four data phases before an ADB or on the first data phase, the initiator deasserts **FRAME#** two clocks later and disconnects the transaction on the ADB. The initiator treats Disconnect at Next ADB the same as Data Transfer in all other data phases.

9. If the transaction has four or more data phases, the initiator floats the **C/BE#** bus on the clock it deasserts **IRDY#**. If the transaction has less than four data phases, the initiator floats the **C/BE#** bus either on the clock it deasserts **IRDY#** or one clock after that.

10. If the transaction is a write with four or more data phases, the initiator floats the **AD** bus on the clock it deasserts **IRDY#**. If the transaction is a write with less than four data phases, the initiator floats the **AD** bus either on the clock it deasserts **IRDY#** or one clock after that.

11. The default Latency Timer value for initiators in PCI-X mode is 64. Initiators must disconnect the current transaction on the next ADB if the Latency Timer expires and **GNT#** is deasserted.

1.10.3. Target Rules

The following rules apply to the way a target responds to a transaction:

1. Memory address ranges (including those assigned through Base Address registers) for all devices must be no smaller than 128 bytes. System configuration software assigns the memory range of each function of each device (that requests Memory Space) to different ranges aligned to ADBs. No two device-functions respond to addresses between the same two adjacent ADBs.
2. The target claims the transaction by asserting **DEVSEL#** and leaving **TRDY#** and **STOP#** deasserted, using decodes A, B, C, or Subtractive, as given in Table 2-8.
3. After a target asserts **DEVSEL#**, it must complete the transaction with one or more data phases by signaling one or more of the following: Split Response, Target-Abort, Single Data Phase Disconnect, Wait State, Data Transfer, Retry, or Disconnect at Next ADB. See Table 2-16.
4. The target is not permitted to signal Wait State after the first data phase. If the target signals Split Response, Target-Abort, or Retry, the target must do so within eight clocks of the assertion of **FRAME#**. If the target signals Single Data Phase Disconnect, Data Transfer, or Disconnect at Next ADB, the target must do so within 16 clocks of the assertion of **FRAME#**. All PCI-X targets (including the host bridge) are subject to the same target initial latency limits.
5. If a PCI-X target signals Data Transfer (with or without preceding wait states), the target is limited to disconnecting the transaction only on an ADB (until the byte count is satisfied). To disconnect the transaction on an ADB, the target signals Disconnect at Next ADB on any data phase. Once the target has signaled Disconnect at Next ADB, it must continue to do so until the end of the transaction.

   If the target signals Disconnect at Next ADB four or more clocks before an ADB, the initiator disconnects the transaction on that ADB. If the transaction starting address is less than four data phases from an ADB and the target signals Disconnect at Next ADB on the first data phase (with or without preceding wait states), the transaction ends on that ADB. (See Section 2.11.2.2.)
6. The target is permitted to signal Single Data Phase Disconnect only on the first data phase (with or without preceding wait states). It is permitted to do so both on burst transactions (even if the byte count is small enough to limit the transaction to a single data phase) and DWORD transactions (which are always a single data phase). (See Section 2.11.2.1.)
7. The target is permitted to signal Target-Abort on any data phase regardless of its relationship to an ADB.
8. The target deasserts **DEVSEL#**, **STOP#**, and **TRDY#** one clock after the last data phase (if they are not already deasserted) and floats them one clock after that.
9. If the transaction is a read, the target floats the **AD** bus on the clock after the last data phase, regardless of the number of data phases in the transaction or the type of termination. That is, the target floats the **AD** bus on the clock it deasserts **DEVSEL#**, **STOP#**, and/or **TRDY#** after signaling the last Data Transfer or target termination.

1.10.4. Bus Arbitration Rules

The following protocol rules apply to bus arbitration:

1. As in conventional PCI, the arbitration algorithm is not specified. The arbiter is required to be fair to all devices (see Section 4.1). If a device signals Split Response, arbitration within that device must fairly allow the initiation of the Split Completion (see Section 4.1.1).
2. All **REQ#** and **GNT#** signals are registered by the arbiter as well as by initiators. That is, they are clocked directly into and out of flip-flops at the arbiter and device interfaces.
3. An initiator is permitted to assert and deassert **REQ#** on any clock. Unlike conventional PCI, there is no requirement to deassert **REQ#** after a target termination (**STOP#** asserted). (The arbiter is assumed to monitor bus transactions to determine when a transaction has been target terminated if the arbiter uses this information in its arbitration algorithm.)
4. An initiator is permitted to deassert **REQ#** on any clock independent of whether **GNT#** is asserted. An initiator is permitted to deassert **REQ#** without initiating a transaction after **GNT#** is asserted.
5. If all the **GNT#** signals are deasserted, the arbiter is permitted to assert any **GNT#** on any clock. After the arbiter asserts **GNT#**, the arbiter must keep it asserted for a minimum of five clocks while the bus is idle, or until the initiator asserts **FRAME#** or deasserts **REQ#**. (This provides the opportunities for all devices to execute configuration transactions.)
6. If only one **REQ#** is asserted, it is recommended that the arbiter keep **GNT#** asserted to that device.
7. If the arbiter deasserts **GNT#** to one device, it must wait until the next clock to assert **GNT#** to another device.
8. An initiator is permitted to start a new transaction (drive the **AD** bus, etc.) on any clock N in which the initiator's **GNT#** was asserted on clock N-2, and either the bus was idle (**FRAME#** and **IRDY#** were both deasserted) on clock N-2 or **FRAME#** was deasserted and **IRDY#** was asserted on clock N-3 (see Section 4.1.1). An initiator is permitted to start a new transaction on clock N even if **GNT#** is deasserted on clock N-1.
9. All fast back-to-back transactions as defined in PCI 2.2 are not permitted in PCI-X mode.
10. In PCI hot-plug systems, the arbiter must coordinate with the Hot-Plug Controller to prevent hot-plug operations from interfering with other bus transactions. See Section 4.3.

1.10.5. Configuration Transaction Rules

The following protocol rules apply to configuration transactions:

1. PCI-X initiators must drive the address for four clocks before asserting **FRAME#** for configuration transactions when in PCI-X mode.
2. In addition to the information required in conventional PCI for a Type 0 configuration transaction, in PCI-X the transaction must include the target device

number in **AD[15::11]** of the address phase. (See Section 2.7.2.2.) The target device bus number is provided in **AD[7::0]** of the attribute phase. The target of a Type 0 Configuration Write transaction stores its device number and bus number in its internal registers.

3. Software is required to write to the Configuration Space of every device on a bridge's secondary bus after changing that bridge's secondary bus number. This occurs as part of the device enumeration process.
4. Type 1 configuration transactions flow through the bus hierarchy the same as in conventional PCI.

1.10.6. Parity Error Rules

The following rules apply to parity error conditions:

1. If a device receiving data (i.e., the target of a write or Split Completion and the initiator of a read) detects a data parity error, it must assert **PERR#** (if enabled) on the second clock after **PAR64** and **PAR** are driven (one clock later than conventional PCI).
2. During read transactions, the target drives **PAR64** (if responding as a 64-bit device) and **PAR** on clock N+1 for the read data it drives on clock N and the byte enables driven by the initiator on clock N-1. During write transactions, the initiator drives **PAR64** (if initiating as a 64-bit device) and **PAR** on clock N+1 for the write data and the byte enables it drives on clock N.
3. All PCI-X devices are required to service data parity error conditions for their transactions. See Section 5.4.1.
4. If a device detects a parity error on an attribute phase, the device asserts **SERR#** (if enabled), independent of whether the device decodes its address during the address phase.
5. For Split Transactions, the requester sets the Master Data Parity Error bit in the Status register for data parity errors on either the Split Request or the Split Completion.
6. If data parity error recovery is disabled, the device asserts **SERR#** when a data parity error occurs (see Section 5.4.1).
7. Other requirements for asserting **SERR#** and setting status bits for address-phase and data-phase errors are the same as for conventional PCI.

1.10.7. Bus Width Rules

The following rules apply to the width of the transaction:

1. As in conventional PCI, PCI-X devices are permitted to implement either a 64-bit or a 32-bit interface.
2. The width of the address is independent of the width of the data transfer.
3. All devices that initiate memory transactions must be capable of generating 64-bit memory addresses.
4. If a device requests a memory range through a Base Address register, that Base Address register must be 64-bits wide.

5. If an address is greater than 4 GB, all initiators (including 64-bit devices) generate a dual address cycle.
6. The attribute phase is always a single clock long for both 64-bit and 32-bit initiators.
7. Only burst transactions (memory commands other than Memory Read DWORD) use 64-bit data transfers. (This maximizes similarity with conventional PCI, in which only memory transactions use 64-bit data transfers.) The width of each transaction is determined with a handshake protocol on **REQ64#** and **ACK64#** that is similar to conventional PCI.

1.10.8. Split Transaction Rules

The following rules apply to Split Transactions:

1. Any transaction that is terminated with Split Response results in one or more Split Completion transactions.
2. Split Completions contain either read data or a Split Completion Message but not both.
3. If the completer returns read data, the Completer must return all the data (the full byte count) unless an error occurs. The read data is delivered in multiple Split Completion transactions if either the initiator or the target disconnects it at ADBs. The initiator (completer) is also permitted to adjust the byte count of the Split Completion to terminate it on the first ADB. Each time the Split Completion resumes after a disconnection, the initiator adjusts the byte count (and starting address) to indicate the number of bytes remaining in the Sequence.
4. The requester must accept all data phases of a Split Completion. The requester must terminate a Split Completion with Data Transfer or Target-Abort. (Initial wait states are permitted. See Section 2.10.5 for restrictions on signaling Target-Abort.) The requester must never terminate a Split Completion transaction with Split Response, Single Data Phase Disconnect, Retry, or Disconnect at Next ADB. A PCI-X bridge forwarding a Split Completion from one PCI bus to another (when both are operating in PCI-X mode) is permitted to disconnect the Split Completion or terminate it with Retry under certain conditions (see Section 8.4.5).
5. If the request is a write transaction, or if the completer encounters an error while executing the request, the completer sends a Split Completion Message to the requester. Although Split Completion transactions are considered burst transactions (i.e., they include the byte count), a Split Completion Message is always a single data phase. The Split Completion Message includes not only an indication of how the transaction completed, but if an error occurred during a read operation, the message includes an indication of the length of the Sequence that remains unsent. Intervening bridges optionally use this information to manage their internal buffers.

1.11. PCI-X Transaction Flow

The following two figures illustrate how transactions flow through a PCI-X system. Figure 1-3 illustrates transaction flow from a requester directly to a completer (no intervening bridge). Figure 1-4 illustrates transaction flow across a bridge.

As shown in Figure 1-3, transactions start at the requester's initiator interface. The completer's target interface terminates the transaction in one of several ways. If the completer signals Split Response, the completer's initiator interface initiates one or more Split Completion transactions addressing the requester's target interface.

A bridge between the requester and completer allows the transactions on the requester's side and the completer's side to execute independently on their respective buses. Figure 1-4 shows transactions starting at the requester's initiator interface as before and flowing upstream across a bridge to a completer on the primary bus. The bridge's secondary target interface signals termination of the transaction the same as the completer did in the previous example. (See Section 8.4 for cases in which the bridge must respond with Split Response.) The bridge's primary initiator interface forwards the transaction. Depending upon the response from the completer, the bridge either creates a Split Completion transaction or simply reserves buffer space for a Split Completion from the completer. If the completer creates the Split Completion, its initiator interface returns it to the bridge's primary target interface. When the bridge has the Split Completion (either it created it or received it from the completer), the bridge's secondary initiator interface initiates the Split Completion addressing the requester's target interface.

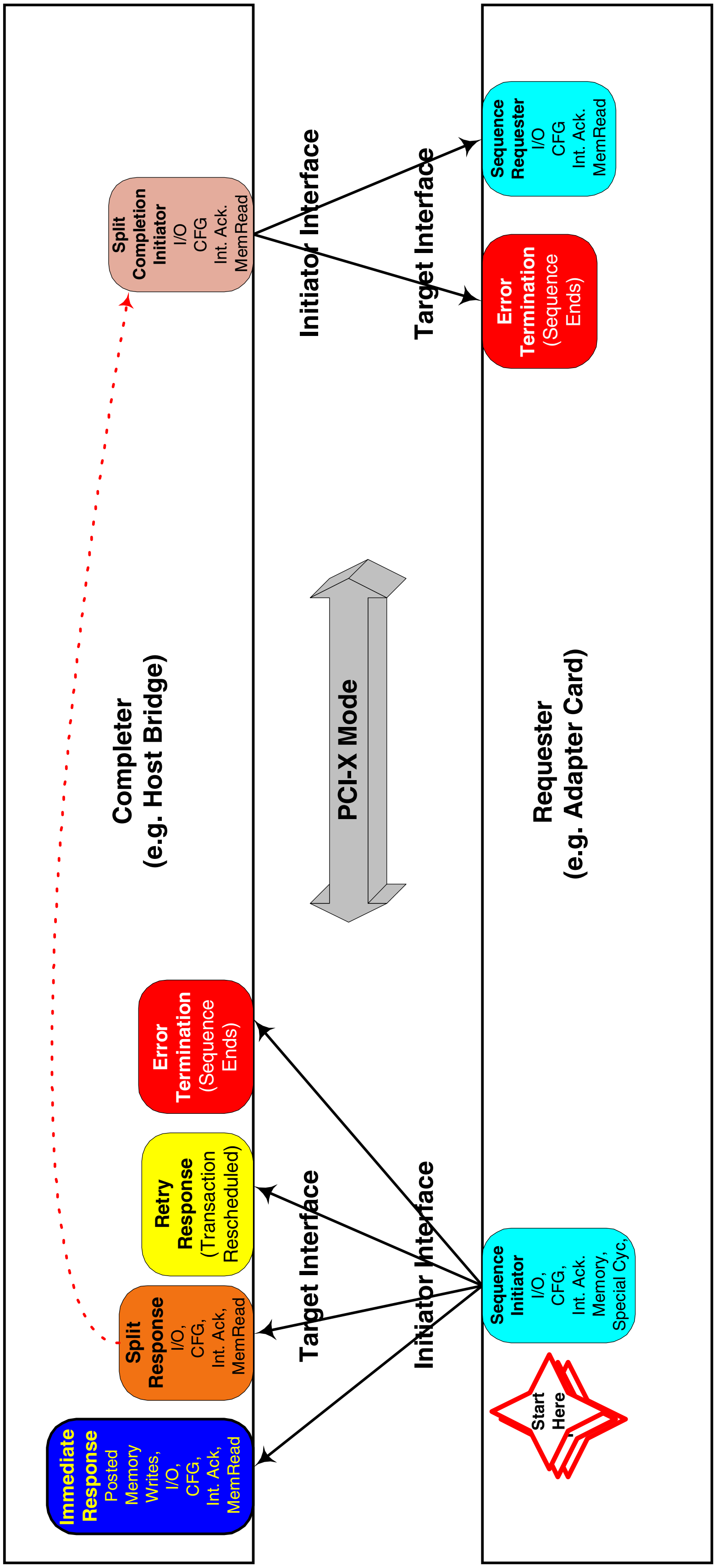


Figure 1-3: Transaction Flow without Crossing a Bridge

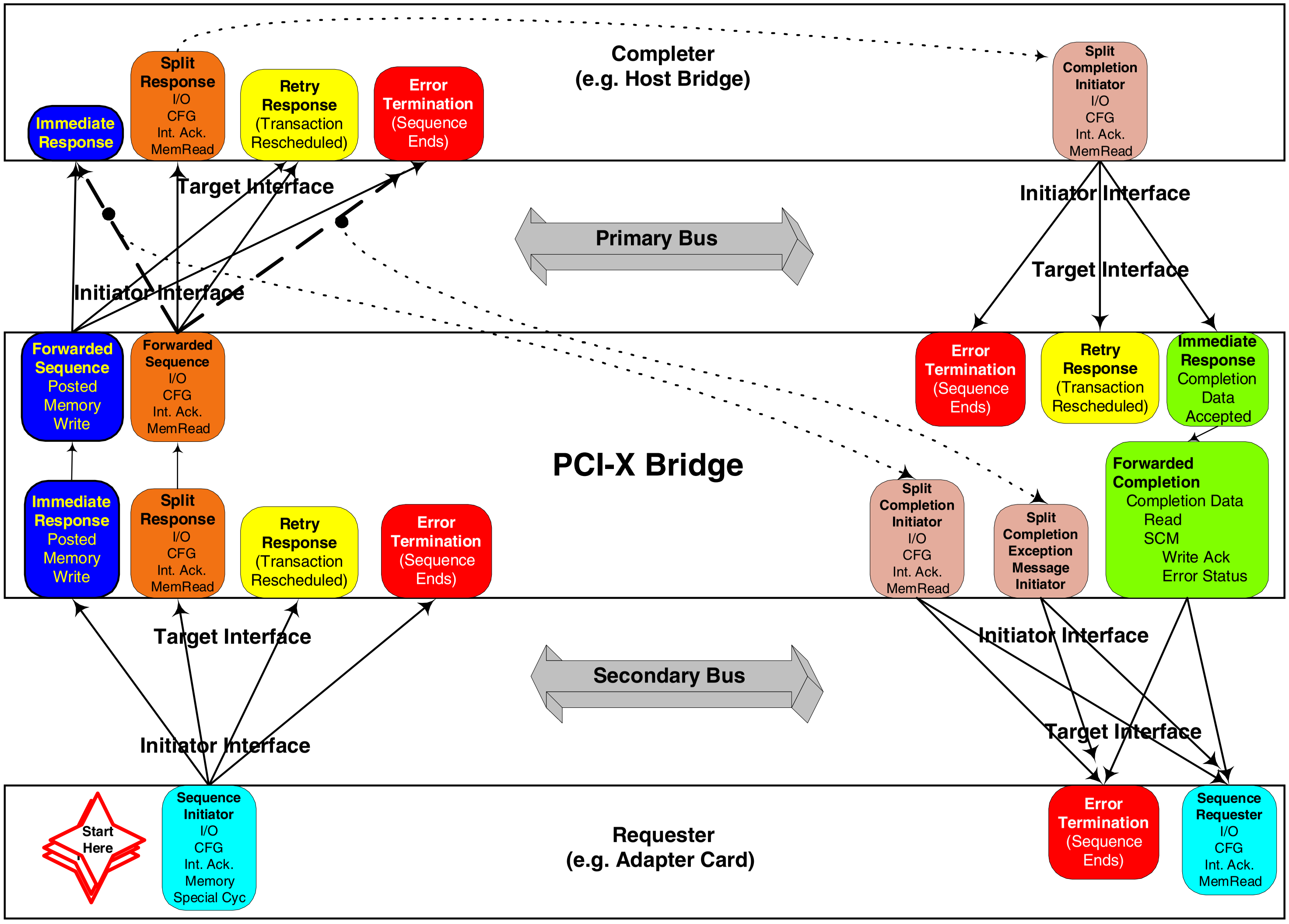


Figure 1-4: Transaction Flow Across a Bridge

2. PCI-X Transaction Protocol

2.1. Sequences

A Sequence is one or more transactions associated with carrying out a single logical transfer by a requester. A Sequence originates with a single request. If a Sequence is broken into more than one transaction, the bytes of all these transactions are included in the byte count of the request that started the Sequence.

Each transaction in the same Sequence carries the same unique Sequence ID (i.e., same Requester ID and Tag). A requester must not initiate a new Sequence using a Tag until the previous Sequence using that Tag is complete.

If the Sequence is a burst write and is disconnected either by the initiator or target, the Sequence has more than one transaction. After a disconnection, the initiator must resume the sequence by initiating another burst write transaction using the same command and adjusting the starting address and byte count for the data already sent. The initiator must deliver the full byte count of the Sequence no matter how many times the Sequence is disconnected and regardless of whether continuations after a disconnection are terminated with Retry. If the Sequence is a burst write and the target signals Target-Abort or no target responds (Master-Abort), the Sequence ends when the transaction terminates. If the Sequence is a burst write and the target signals Retry on the first data phase of the Sequence, the Sequence ends immediately. The requester is not obligated to repeat a Sequence terminated with Retry on the first data phase (unless it is required by the application, e.g., a PCI-X bridge forwarding a memory write Sequence). However, if the requester repeats the burst write, it is considered a new Sequence and is permitted to use the same or a different command and attributes (byte count, Tag, etc.), within the limits specified in Section 2.5. The requester is permitted to reuse a Tag as soon as the memory write Sequence completes on the requester's bus, independent of whether there are one or more PCI-X bridges between the requester and completer, and if so, when those bridges forward the Sequence to the completer.

If the Sequence is a burst read that is terminated with a Retry or executes as an Immediate Transaction (i.e., the target delivers data or terminates with Target-Abort or no target responds (Master-Abort)), the Sequence terminates when the transaction terminates. The requester is not required to repeat a burst read transaction terminated with a Retry, or to resume an immediate read Sequence. If the requester still needs the data for the remainder of the Sequence (e.g., a bridge forwarding a Split Transaction), it must repeat the transaction terminated with Retry or continue reading from the disconnection point. However, this is considered a new Sequence and is permitted to use the same or a different Tag (see Section 2.5). If a target responds immediately with data (no Split Response) to a burst read transaction, and that transaction is later disconnected (either by the initiator or the target), the target must discard any state information associated with that transaction and any undelivered data, unless the target guarantees that the data will not become stale. Delayed Transactions as defined in PCI 2.2 are not permitted.

If the Sequence executes as a Split Transaction, the Sequence has exactly one Split Request transaction (for each bus it crosses) and one or more Split Completion transactions. Split Transactions do not complete until the Split Completions satisfy the byte count or indicate that an error occurred. If the target (completer) terminates a burst read transaction with Split Response, the completer assumes the responsibility for delivering the entire byte count (except for error conditions described in

Section 2.10.6.2). Read data is sent in one or more Split Completion transactions, each of which includes in the address phase the Sequence ID of the original request. If the completer initiates multiple Split Completions for a single Sequence, the completer is required to initiate them in address order.

If a Sequence crosses a PCI-X bridge, the number of transactions within the Sequence on each bus is determined by the behavior of the devices on each bus. The Sequence may have the same or a different number of transactions on each bus. The bridge is required to keep Split Completions for the same Sequence in address order.

2.2. Allowable Disconnect Boundaries and Buffer Size

An allowable disconnect boundary (ADB) is a naturally aligned 128-byte address; that is, an address whose lower 7 bits are zeros. After a burst data transfer starts and the target signals that it will accept more than a single data phase, the transaction can only be stopped in one of the following ways:

- target or initiator disconnection at an ADB
- the byte count is satisfied
- Target-Abort

If a burst transaction is disconnected either by the initiator or the target on an ADB, the address of the last byte transferred modulo 80h is 7Fh. That is, the lower seven bits of the address of the last byte transferred are 7Fh.

ADBs are the same regardless of the width of the transaction. A burst transaction requires 16 data phases to go from one ADB to the next on a 64-bit bus and 32 data phases on a 32-bit bus.

Burst transactions are permitted to start on any byte address. Both the initiator and the target are permitted to disconnect a burst transaction on any ADB. A target is permitted also to disconnect after a single data phase. (See Section 2.11.1.1 for the special case for the initiator when the starting address is less than four data phases from an ADB.) Therefore, the minimum buffer size that both the initiator and target must use is 128 bytes.

Implementation Note: Efficient Partitioning of Large Operations

ADBs are naturally aligned to improve the efficiency of aligned-address devices like host bridges. Host bridges generally function much more efficiently when requests are aligned to their cacheline boundaries.

A device reading or writing a large block of data is permitted to use transactions of any size. However, performance is generally better if the device uses the largest allowable block size and disconnects on efficient boundaries. If a device partitions a large transfer into multiple Sequences (e.g., if the Maximum Memory Read Byte Count register is programmed to a value smaller than the size of the operation (see Section 7.2.3) or the transfer is larger than 4096 bytes), those Sequences generally execute faster and more efficiently in the host bridge if they are partitioned on ADBs.

Therefore, whenever an initiator breaks a large operation into multiple Sequences, the initiator is encouraged to break it only on ADBs.

Implementation Note: Atomic Operations

Unlike conventional PCI, the restricted disconnection boundaries of PCI-X transactions guarantees that certain transactions complete atomically. That is, within certain limits, a PCI-X requester is guaranteed that all data phases of a burst transaction complete as a single transaction and are not interrupted by other transactions.

If the requester and completer restrict the transactions as described below, the transaction executes atomically on the bus:

1. The transaction length is one ADQ.
2. The completer does not signal Single Data Phase Disconnect.
3. The transaction does not cross a bus segment operating in conventional PCI mode.

If these restrictions are met, the transaction completes atomically, even if it crosses one or more PCI-X bridges between the requester and completer.

If these restrictions are met, devices are permitted to define their programming model to make use of atomic operations. For example, a device could include a 64-bit pointer that must be read and written in a single operation. Even if such a device (or an intervening bus segment) is only 32 bits wide, the pointer is updated as a single 64-bit operation, provided that the requester always initiates the operation as a single transaction, and the device is always used in a PCI-X system.

Conventional PCI does not provide this capability, since conventional PCI transactions are permitted to be disconnected on any boundary. Therefore, care must be taken to limit the use of conventional PCI devices in any system in which this PCI-X feature is required. The presence of a conventional PCI device on a bus segment requires that bus segment to operate in conventional mode.

2.3. Dependencies Between Address, Byte Count, and Byte Enables

As in conventional PCI, PCI -X Memory, I/O, and Configuration Spaces are addressable at the byte level. That is, each byte has a unique address, and each address space uses different commands. Furthermore, transactions are naturally aligned to byte lanes. That is, bytes appear on byte lanes according to their individual byte address and the width of the transfer (independent of the starting address of the transaction). Table 2-1 shows how bytes are assigned to byte lanes.

Table 2-1: Byte Lane Assignments

	Data Byte Lane	
Byte Address AD[2::0]	64-Bit Transfer	32-Bit Transfer
0	AD[7::0]	AD[7::0]
1	AD[15::08]	AD[15::08]
2	AD[23::16]	AD[23::16]
3	AD[31::24]	AD[31::24]
4	AD[39::32]	AD[7::0]
5	AD[47::40]	AD[15::08]
6	AD[55::48]	AD[23::16]
7	AD[63::56]	AD[31::24]

For burst transactions, the PCI-X requester uses the full address bus to indicate the starting byte address (including **AD[1::0]**) and includes the byte count in the attribute field. All bytes from the starting address through the end of the byte count are included in the transaction or subsequent continuations of the Sequence (but see also the use of byte enables on Memory Writes transactions described below). (The byte count sometimes spans more that one transaction. See Section 2.5.) Bytes prior to the starting address or beyond the ending address (i.e., starting address + byte count - 1) are not included in the Sequence. These addresses are not affected by write transactions, and data is not predictable on read transactions. For burst transactions, the starting address and ending address are not required to have any alignment to the bus width.

For I/O and memory DWORD transactions, the PCI-X initiator uses the full address bus to indicate the starting address (including **AD[1::0]**). (Note that this is the same as conventional I/O transaction but not conventional memory transactions.) If at least one byte enable is asserted, the starting address is the address of the first enabled byte, as shown in Table 2-2. If no byte enables are asserted, the starting address is permitted to be any byte in the DWORD. The ending address is always the last byte of the DWORD. The completer is permitted to use either **AD[1::0]** or the byte enables to determine the first byte of the transaction.

Table 2-2: AD[1::0] and Byte Enable Encodings for I/O and DWORD Memory Transactions

AD[1::0]	Valid Byte Enable Combinations (Note 1)
00b	xxx0b or 1111b
01b	xx01b or 1111b
10b	x011b or 1111b
11b	0111b or 1111b

Notes:

1. "1" indicates the byte enable is deasserted. "0" indicates the byte enable is asserted. "x" indicates the byte enable is permitted to be either asserted or deasserted.

Like conventional PCI, the lower two address bits for a PCI-X configuration transaction indicate the configuration transactions type (see Section 2.7.2.2). The starting and ending addresses are considered to be the first and last bytes of the DWORD, respectively.

As in conventional PCI, Interrupt Acknowledge and Special Cycle transactions have no address and are permitted to drive any stable value during the address phase.

Byte enables are included in the Requester Attributes of all DWORD transactions. Byte enables are also included in the **C/BE[7::0]#** bus for 64-bit transfers and the **C/BE[3::0]#** bus for 32-bit transfers in the data phases of all Memory Write transactions. The **C/BE#** bus is reserved and driven high by the initiator throughout all data phases of all other transactions.

Byte enables further qualify the bytes affected by the transaction. For those transactions that include byte enables, only bytes for which the byte enable is asserted are affected by the transaction. DWORD transactions are permitted to have any combination of byte enables, including no byte enables asserted. Memory Write transactions are permitted to have any combination of byte enables between the starting and ending addresses, inclusive, including no byte enables asserted. Byte enables must be deasserted for bytes before the starting address and after the ending address (if those addresses are not aligned to the width of the bus) as shown in Table 2-3 and Table 2-4. See Section 2.12.3 for exceptions and additional requirements when a 64-bit initiator addresses a 32-bit target and **AD[2]** is 1.

Table 2-3: Starting Address and Byte Enable Dependencies for 32-bit Transactions Using the Memory Write Command

AD[1::0]	Valid Byte Enable Combinations C/BE[3::0]# (Note 1)
00b	xxxxb
01b	xxx1b
10b	xx11b
11b	x111b

Notes:

1. "1" indicates the byte enable is deasserted. "x" indicates the byte enable is permitted to be either asserted or deasserted.

Table 2-4: Starting Address and Byte Enable Dependencies for 64-bit Transactions Using the Memory Write Command

AD[2::0]	Valid Byte Enable Combinations C/BE[7::0]# (Note 1, 2)
000b	xxxx xxxxb
001b	xxxx xxx1b
010b	xxxx xx11b
011b	xxxx x111b
100b	xxxx xxxxb
101b	xxx1 xxx1b
110b	xx11 xx11b
111b	x111 x111b

Notes:

1. "1" indicates the byte enable is deasserted. "x" indicates the byte enable is permitted to be either asserted or deasserted.
2. **C/BE[7::4}#** are required to be copied to C/BE[3::0]# for 32-bit targets when AD[2] of the starting address is 1. See Section 2.12.3.

2.4. PCI-X Command Encoding

Table 2-5 shows the PCI-X command encodings. Conventional PCI command encodings are shown for reference. Initiators must not generate reserved commands. Targets must ignore (must not assert **DEVSEL#** or change any state) any transactions using a reserved command.

Table 2-5: PCI-X Command Encoding

C/BE[3::0]# or C/BE[7::4]#	Conventional PCI Command (ref)	PCI-X Command	Length	Byte-Enable Usage	Notes 1
0000b	Interrupt Acknowledge	Interrupt Acknowledge	DWORD	attr	
0001b	Special Cycle	Special Cycle	DWORD	attr	4
0010b	I/O Read	I/O Read	DWORD	attr	
0011b	I/O Write	I/O Write	DWORD	attr	
0100b	Reserved	Reserved	na	na	
0101b	Reserved	Reserved	na	na	
0110b	Memory Read	Memory Read DWORD	DWORD	attr	
0111b	Memory Write	Memory Write	Burst	dp	
1000b	Reserved	Alias to Memory Read Block	Burst	none	2
1001b	Reserved	Alias to Memory Write Block	Burst	none	3
1010b	Configuration Read	Configuration Read	DWORD	attr	4
1011b	Configuration Write	Configuration Write	DWORD	attr	4
1100b	Memory Read Multiple	Split Completion	Burst	none	
1101b	Dual Address Cycle	Dual Address Cycle	na	na	1
1110b	Memory Read Line	Memory Read Block	Burst	none	
1111b	Memory Write and Invalidate	Memory Write Block	Burst	none	

Legend:

DWORD	Transaction must be a single DWORD (or less), and **REQ64#** must not be asserted.
Burst	Transaction permitted to be any length (from 1 to 4096 bytes), and **REQ64#** is asserted at the option of the initiator.
na	Not applicable.
attr	The byte enables appear in the Requester Attributes. All bit combinations ar e legal, including no byte enables asserted. The **C/BE#** bus is reserved and driven high during the data phase.
dp	The **C/BE#** bus contains valid byte enables for each data phase. All bit combinations between the starting and ending address inclusive are legal, including no byte enables asserted.
none	All bytes between the starting and ending address inclusive are affected. The **C/BE#** bus is reserved during all data phases and driven high by the initiator.

Notes:

1. For all commands other than Dual Address Cycle, the transaction command appears on **C/BE[3::0]#** during the (single) address phase. For transactions with dual address cycles, **C/BE[3::0]#** contain the Dual Address Cycle command (1101b) in the first address phase and the transaction command in the second address phase. For 64-bit transactions, **C/BE[7::4]#** contain the transaction command in both address phases.

2. This command is reserved for use in future versions of this specification. Current initiators must not generate this command. Current targets must treat this command as if it were Memory Read Block.

3. This command is reserved for use in future versions of this specification. Current initiators must not generate this command. Current targets must treat this command as if it were Memory Write Block.

4. These commands require special protocol. See Sections 2.7.2 and 2.7.3.

2.5. Attributes

Attributes are additional information included with each transaction that further defines the transaction. The initiator of every transaction drives attributes on the **C/BE[3::0]#** and **AD[31::00]** buses in the attribute phase. For burst transactions, all attribute bits are high-true. That is, an attribute bit value of 1 appears on both the **C/BE[3::0]#** and **AD[31::00]** buses as a high logic voltage, and a bit value of 0 appears as a low logic voltage. For DWORD transactions, all attribute bits are high-true except the byte enables, which are low-true. The attribute phase is always a single clock regardless of the width of the data transfer or the width of the address (single or dual address cycle). The upper buses (**AD[63::32]** and **C/BE[7::4]#**) of 64-bit devices are reserved and driven high during the attribute phase.

There are three different attribute formats for requesters and one format for completers. The Requester Attribute formats for burst and DWORD transactions are presented in this section. The Requester Attribute format for Type 0 configuration transactions is presented in Section 2.7.2.2 and Completer Attributes for Split Completions in Section 2.10.4.

Figure 2-1 and Figure 2-2 show the bit assignments for the Requester Attribute for burst and DWORD transactions, respectively. Table 2-6 describes the bit definitions.

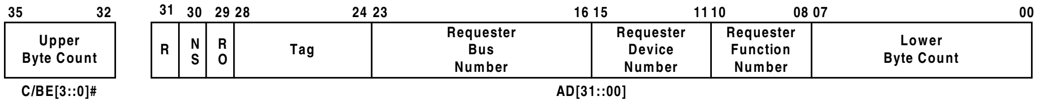


Figure 2-1: Burst Transaction Requester Attribute Bit Assignments

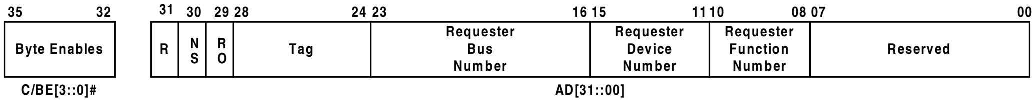


Figure 2-2: DWORD Transaction Requester Attribute Bit Assignments

Table 2-6: Burst and DWORD Requester Attribute Field Definitions

Attribute	Function
Reserved (R)	Must be set to 0 by the requester and ignored by the completer (except for parity checking). PCI-X bridges forward this bit unmodified.
Byte Enables	DWORD transactions include the byte enables for the transaction in the upper four bits of the attributes (**C/BE[3::0]#**). A byte is affected by the transaction if its byte enable in the attribute phase is a 0 (low logic voltage), and a byte is not affected by the transaction if its byte enable is a 1 (high logic voltage).
No Snoop (NS)	If a requester sets this bit, the requester guarantees that the locations between the starting and ending address, inclusive, of this Sequence are not stored in any cache in the system. How the requester guarantees this is beyond the scope of this specification. Examples of transactions that could benefit from setting this bit are transactions that read or write non-cacheable sections of main memory, or sections that have previously been flushed from the cache through hardware or software means. Note that PCI-X, like conventional PCI, does not require systems to support coherent caches for addresses accessed by PCI-X requesters; but for those systems that do, this bit allows device drivers to avoid cache snooping on a Sequence-by-Sequence basis to improve performance. If a write transaction is disconnected, the requester must not change the value of this attribute on any subsequent transaction in the same Sequence. (If an immediate read transaction disconnects, the Sequence ends.) This attribute is used only for memory transactions that are not Message Signaled Interrupts (as defined in PCI 2.2). The requester must not set this bit if the transaction is a Message Signaled Interrupt, I/O, or Special Cycle transaction. (See Section 2.7.2.2 for configuration transactions and Section 2.10.4 for Split Completions.) The use of this bit is optional for completers. If a completer does not implement the use of this bit, it treats all transactions as if the bit is not set. This bit is ignored by PCI-X bridges and forwarded unmodified with the transaction.
Relaxed Ordering (RO)	A requester is permitted to set this bit only if its programming model and device driver guarantee that the particular memory write or read transactions are not required to remain in strict order. In general, devices are permitted to set the Relaxed Ordering attribute bit for payload Sequences and must clear it for control and status Sequences. See Section 11 for a complete discussion of usage models for relaxed transaction ordering. No requester is permitted to set this bit if the Enable Relaxed Ordering bit in the PCI-X Command register is not set.

Attribute	Function
	A requester is permitted to set this bit on a read Sequence if its usage model does not require Split Read Completions for this transaction to stay in order with respect to posted memory writes moving in the same direction. Split Read Requests are unaffected by this bit. (Split Completion transactions from a single Sequence always stay in address order with respect to each other.) A requester is permitted to set this bit on a memory write Sequence if its usage model does not require this memory write Sequence to stay in order with respect to other memory write Sequences moving in the same direction. (Memory write data for the same Sequence always stay in address order.) If a transaction is disconnected, the requester must not change the value of this attribute on any subsequent transaction in the same Sequence. This attribute is used only for memory transactions that are not Message Signaled Interrupts (as defined in PCI 2.2). The requester must not set this bit if the transaction is a Message Signaled Interrupt, I/O, or Special Cycle transaction. (See Section 2.7.2.2 for configuration transactions and Section 2.10.4 for Split Completions.) Use of this bit is optional for targets. If the target (completer or an intervening bridge) implements this bit, and the bit is set for a read transaction, the target is permitted to allow read-completion transactions for this Sequence to pass posted memory write transactions moving in the same direction. If the bit is not set or if the target (completer or bridge) does not implement the bit, the target keeps all read-completion transactions in strict order relative to memory write transactions moving in the same direction. If the bit is set for a memory write transaction, the host bridge is permitted to allow this memory write transaction to pass previously posted memory write transactions moving in the same direction. The host bridge is also permitted to allow bytes within the transaction to be written to system memory in any order. (The bytes must be written to the correct system memory locations. Only the order in which they are written is unspecified). PCI-X bridges ignore this bit for memory write transactions and forward them in the order in which they were received.

Attribute	Function
Tag	This 5-bit field uniquely identifies up to 32 Sequences from a single initiator. The initiator assigns a unique Tag to each Sequence that begins before previous ones end. Other than the requirement for uniqueness, the PCI-X definition does not control how the initiator assigns these numbers. Requesters use this field to identify the appropriate Split Completion transaction. The combination of the Requester ID and Tag is referred to as the Sequence ID.
Requester Bus Number	This 8-bit field identifies the requester's bus number. Requesters supply this number from the Bus Number register in the PCI-X Status register. The value FFh is reserved and means the requester's PCI-X Status register has not been initialized. The combination of the Requester Bus Number, Requester Device Number, and Requester Function Number is referred to as the Requester ID.
Requester Device Number	This 5-bit field contains the device number assigned to the requester. Requesters supply this number from the Device Number register in the PCI-X Status register. The value 1Fh is reserved and means the requester's PCI-X Status register has not been initialized. The Device Number of the source bridge is always 00h. The combination of the Requester Bus Number, Requester Device Number, and Requester Function Number is referred to as the Requester ID.
Requester Function Number	This 3-bit field contains the function number of the requester within the device. This is the function number in the configuration address to which the function responds. Unlike the Device Number and Bus Number fields in the PCI-X Status register, the value of the Function Number field is assigned to the function by design and needs no initialization. The combination of the Requester Bus Number, Requester Device Number, and Requester Function Number is referred to as the Requester ID.

Attribute	Function
Upper Byte Count, Lower Byte Count	Burst transactions include the byte count in the Requester Attributes. This 12-bit field is divided between the Upper Byte Count in the **C/BE[3::0]#** bus and the Lower Byte Count in the **AD[7::0]** bus. It indicates the number of bytes the initiator (requester or bridge) plans to move in the remainder of this Sequence. There is no guarantee that the initiator will successfully move the entire byte count in a single transaction. If this transaction is disconnected for any reason and the initiator continues the Sequence, the initiator must adjust the contents of the Byte Count field in the subsequent transactions of the same Sequence to be the number of bytes remaining in this Sequence. The Byte Count is specified as a binary number, with 0000 0000 0001b indicating 1 byte, 1111 1111 1111b indicating 4095 bytes, and 0000 0000 0000b indicating 4096 bytes. The byte count is *not* included in the Requester Attributes of DWORD transactions. See Section 2.7.2.2 for the use of these fields in configuration transactions.

2.6. Burst Transactions

A burst transaction is a transaction that uses one of the following commands:

- Memory Read Block
- Memory Write Block
- Memory Write
- Alias to Memory Read Block
- Alias to Memory Write Block
- Split Completion

The Alias to Memory Read Block and Alias to Memory Write Block commands are not generated by initiators and are treated as Memory Read Block and Memory Write Block, respectively, by targets. These commands are reserved for future use by the PCI Special Interest Group.

Burst transactions transfer data on one or more data phases, up to that required to satisfy the maximum byte count. They include the byte count for the remainder of the Sequence in the Byte Count field of the attributes (see Section 2.5). Burst transactions are permitted to be initiated both as 64-bit and 32-bit transactions.

Burst transactions use the full address bus (including **AD[2::0]**) to specify the starting byte address of the transaction. There are no restrictions on the starting address. The transaction is permitted to begin on any byte address. (Byte lanes are always naturally aligned to the address. That is, the data for an address with **AD[2::0]** of 000b is always on byte lane 0. The data for an address with **AD[2::0]** of 001b is always on byte lane 1, etc. See Table 2-1.)

There are few boundary restrictions for burst transactions. For example, burst transactions can start on one side and end on the other side of the following:

- An ADB
- A memory page boundary
- The first 4 GB memory address boundary

Implementation Note: Crossing the First 4 GB Address Boundary

If a burst transaction crosses the first 4 GB boundary (i.e., the boundary between memory locations for which the upper 32-bit of the address are 0 and locations for which they are not), the Sequence begins with a single address cycle (see Section 2.12.1). If the Sequence is disconnected and resumes beyond the first 4 GB boundary, the continuation transaction requires a dual address cycle.

PCI-X bridges have range registers that concatenate the ranges of all devices on their subordinate buses and so, in some cases, could have a range that crosses the 4 GB boundary. Since no single device on the secondary bus straddles the boundary, in normal use no single transaction crosses the boundary. Although initiating a transaction that crosses from one device to another is not intended to be a normal operation and in some cases causes a Split Completion Exception Message, initiators are not prohibited from doing so. Such behavior would occur, for example, if a bridge combined write transactions intended for different targets at adjacent addresses. A PCI-X bridge must forward a transaction that crossed the first 4 GB boundary without causing errors.

Host bridges commonly respond to addresses on both sides of the 4 GB address boundary. However, in many systems, the addresses immediately below the 4 GB boundary are assigned to special system functions rather than system memory. In such systems, no initiator is ever assigned a memory buffer that straddles the 4 GB boundary, since the addresses below the 4 GB boundary are not general-purpose memory. Host bridges designed exclusively for such systems are never the target of a burst transaction that crosses the 4 GB boundary and, therefore, have no need for special hardware for this case.

A transaction using any of the memory write commands is permitted to cross a device boundary. In such a case, the first target disconnects the transaction at the ADB that corresponds to its device boundary, and another target (if present) responds when the Sequence resumes. Memory write transactions commonly cross device boundaries as a consequence of combining smaller write transactions of different Sequences. (See Section 8.4.6.)

Read commands generally cross a device boundary only under abnormal conditions. A normally functioning requester understands the address range of the completer it is attempting to read and does not request data that is out of range. Combining separate read Sequences by bridges is generally not allowed. (See Section 8.4.2.) A requester that initiates a burst read that crosses a device boundary must be prepared for it to complete in any of the following ways:

- Completion as an Immediate Transaction to the device boundary. The completer must be on the same bus segment as the requester for this case to occur. How a requester would discover what bus segment the completer is on is beyond the scope of the PCI-X definition.

- A Split Completion Message indicating the request is out of range or that the Sequence has crossed a device boundary:
    - ◊ Byte Count Out of Range from the completer. (See Section 2.10.6.)
    - ◊ Master-Abort from an intervening bridge. (See Section 8.8.)
    - ◊ Target-Abort from an intervening bridge. (See Section 8.8.)
- Target-Abort. (See Section 2.11.2.5.)

Burst transactions are not permitted to go beyond the end of the 64-bit memory address space. In other words, the address plus the byte count minus one must not exceed FFFF FFFF FFFF FFFFh.

After the attribute phase and during the data phases of a burst transaction, the **C/BE#** bus is reserved and driven high by the initiator for all transactions except Memory Write. See Section 2.6.1 for the behavior of the **C/BE#** bus for a Memory Write transaction.

The following two sections describe basic burst memory write and read transactions. All figures illustrate a single-address-cycle transaction between an initiator and target with the same bus width. See Section 2.12 for dual-address-cycle transactions and the requirements when the initiator and target are of different widths.

2.6.1. Burst Writes and Split Completions

Burst write transactions use the Memory Write, Memory Write Block, or Alias to Memory Write Block commands. Burst write and Split Completion transactions are the same in most respects. For these transactions, the initiator is the source not only of the command, address, and attributes but also the data.

The target is permitted to respond to a burst write transaction with any of the following:

- Target-Abort
- Single Data Phase Disconnect
- Wait State
- Data Transfer
- Retry
- Disconnect at Next ADB (Some restrictions apply to bridges. See Section 8.4.6.)

The target is permitted to respond to a Split Completion transaction with any of the following:

- Target-Abort (See Section 2.10.5 for restrictions.)
- Wait State
- Data Transfer
- Retry (Allowed only for bridges. See Sections 2.13 and 8.4.5.)
- Disconnect at Next ADB (Allowed only for bridges. See Sections 2.13 and 8.4.5.)

The target is not permitted to respond with Split Response. Burst write and Split Completions are always executed as Immediate Transactions. In some cases, signaling Target-Abort for a Split Completion causes the system to halt execution. See Section 2.10.5 for restrictions on signaling Target-Abort for a Split Completion transaction. The use of Retry, Single Data Phase Disconnect, and Disconnect at Next

ADB has some restrictions. See Sections 2.10.5 and 2.13 for restrictions for simple devices and Section 8.4.5 for restrictions for PCI-X bridges.

The **C/BE#** bus is reserved and driven high after the attribute phase of all burst write and Split Completion transactions except Memory Write. All bytes between the starting and ending address inclusive are included in these transactions. See Section 2.12.3 for requirements for 64-bit initiators addressing 32-bit targets.

Transactions using the Memory Write command include explicit byte enables on the **C/BE#** bus for each data phase. A byte is not affected by the transaction in any way if its byte enable is not asserted. Except for a case of a 64-bit initiator when **AD[2]** is 1, which is described in Section 2.12.3, the byte enables are deasserted for all bytes before the starting address or after the ending address (if those addresses are not aligned to the width of the bus). All byte enable patterns are permitted (between the starting and ending address, inclusive), including no byte enables asserted. The byte count is not affected if byte enables are deasserted. In other words, the byte count would be the same whether all byte enables were asserted or no byte enables were asserted. The initiator is required to drive all bits of the **AD** bus on every data phase, even if some byte enables are deasserted. The value of the data driven on all the byte lanes is included in the generation of bus parity, even if some of the byte enables are deasserted.

Implementation Note: Completing the Byte Count of a Memory Write Sequence.

PCI-X requesters are required to deliver the full byte count for a memory write Sequence (both for Memory Write and Memory Write Block commands). (See Section 2.1.) Many implementations start a write Sequence on the PCI bus before all the data has arrived at the interface from its remote location. In such implementations, if an error occurs that prevents the arrival of the rest of the data, the requester is still obligated to finish the full byte count of the write on the PCI bus. The value of the data used by the requester in such situations is beyond the scope of this specification.

If the requester uses the Memory Write command, the requester has the option of deasserting byte enables for bytes that it supplied after the error occurred.

After such an error condition, the device must use other means beyond the scope of this specification to notify its device driver that the problem occurred and that not all of the data is valid.

Implementation Note: Comparison of PCI-X "Memory Write Block" and Conventional PCI "Memory Write and Invalidate"

The PCI-X Memory Write Block command has some similarities and some differences with the conventional PCI Memory Write and Invalidate command. This section offers some guidelines for the use of the Memory Write Block command for those applications that benefit from the characteristics of the Memory Write and Invalidate command.

PCI 2.2 supports the Memory Write and Invalidate command as a means to improve system performance when used in conjunction with a cache coherency policy. When using this command in a conventional PCI system, the initiating device must guarantee that complete cachelines are transferred during the write operation. All devices capable of generating a Memory Write and Invalidate command must support the Cacheline Size register. Additionally, all Memory Write and Invalidate transactions are required to start and end on cacheline boundaries, to have all byte enables asserted, and to consist of one or more cachelines. The host bridge is assured that data held in the processor's cache can be invalidated without requiring a write-back into system memory thus improving system performance.

The PCI-X definition provides similar capabilities with the Memory Write Block command. It requires that all bytes between the starting and ending address areincluded in the write operation. With additional usage restrictions, the Memory Write Block command can be made to function identically to the Memory Write and Invalidate command. If a PCI-X device that is capable of bursting data into system memory wants to optimize for cache operation, that device should follow the same guidelines as listed above for conventional PCI devices:

- Align the start of the transaction to the beginning of a cacheline.
- Ensure that the length of the transaction is a multiple of the Cacheline Size register.
- Do not set the No Snoop attribute bit (unless the cache coherency policy guarantees that this line is not in any cache).
- Use the Memory Write Block command.

There is no requirement that PCI-X devices capable of bursting data to memory follow the above guidelines. Host bridges must be capable of accepting Memory Write Block commands with any starting address and any length supported by PCI-X. Although the use of the Memory Write Block command has no directly specified relationship to the value programmed in the Cacheline Size register, improved system performance when bursting to system memory can be obtained in many systems by following the above guidelines.

The initiator is not permitted to insert wait states, so the initiator must drive write (or Split Completion) data on the bus two clocks after the attribute phase. If the target inserts initial wait states, it must do so in pairs of clocks (for transactions that successfully transfer data), and the initiator must toggle between the first and second data patterns until the target begins accepting data. See Section 2.9.2 for a complete discussion of the effects of different **DEVSEL#** decode times and target initial wait states.

Implementation Note: Don't-Care Clock on Write Data

The data is not required to be valid in the clock after the attribute phase of a write (or Split Completion) transaction. This clock could be used as a turn-around clock by multi-package host bridges that source the address from one package and the data from another.

The next two figures show burst write transactions using the Memory Write command. Memory Write Block and Split Completion transactions behave the same way as illustrated except that the **C/BE#** bus is driven high by the initiator during each data phase. The top portion of the diagram shows the signals as they appear on the bus. The middle and lower portions of the diagram show the bus from the viewpoint of the initiator and target, respectively. Signal names preceded by "s1_" indicate a signal that is internal to the device after the signal has been sampled. For example, the initiator asserts **FRAME#** on clock 3. The target samples **FRAME#**, so s1_FRAME# is asserted on clock 4.

Figure 2-3 shows the minimum **DEVSEL#** decode time and target initial latency and eight data phases.

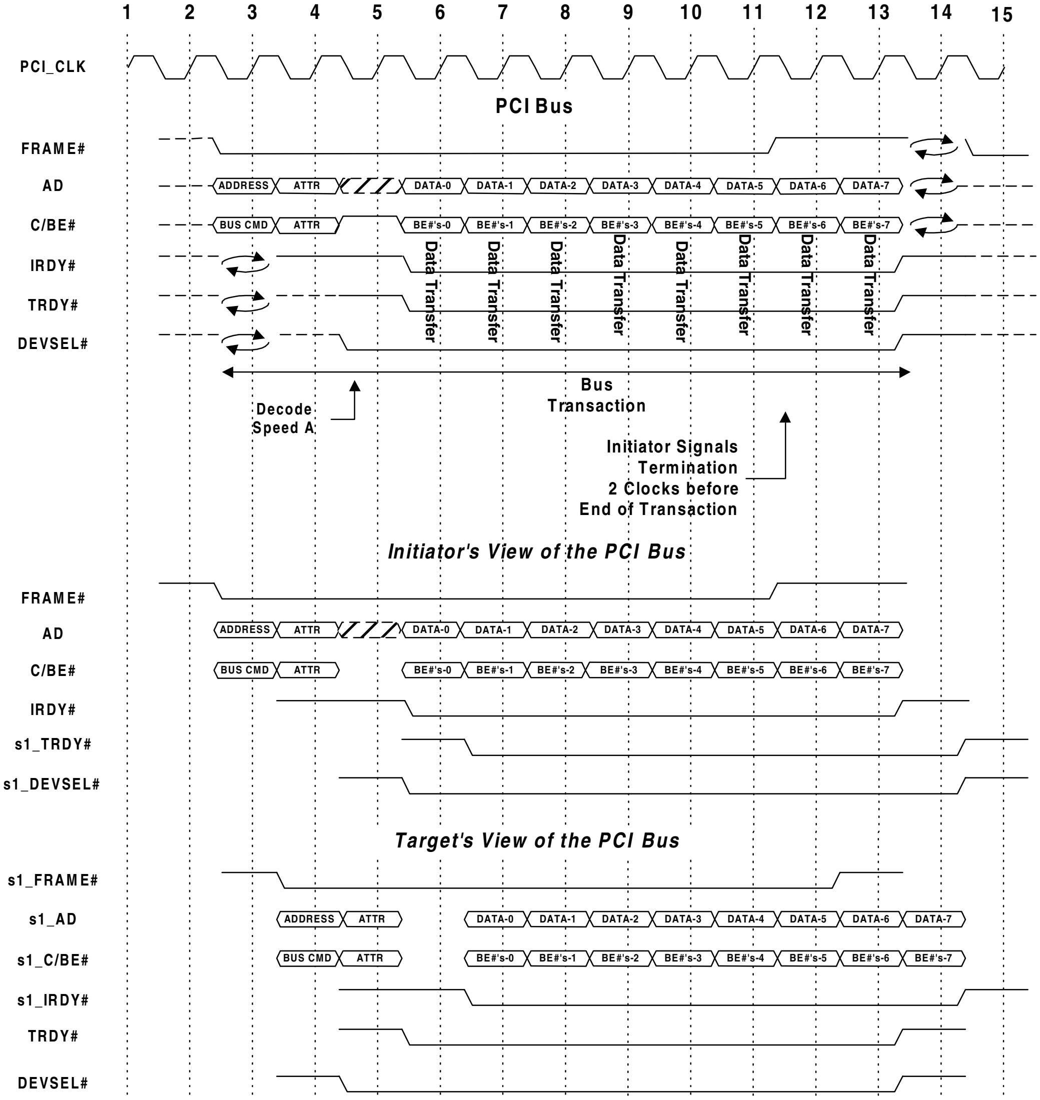


Figure 2-3: Burst Memory Write Transaction with No Target Initial Wait States

Figure 2-4 shows a similar transaction with minimum **DEVSEL#** decode timing but with only six data phases and a target initial latency of five clocks (two wait states). Notice at clocks 6 and 7, the initiator toggles between DATA-0 and DATA-1. This toggling starts one clock after the target asserts **DEVSEL#** (clock 5) and continues until the target asserts **TRDY#** (clock 8). (Note that this data toggling is why target initial wait states must occur in pairs for memory write and Split Completion transactions.) See Section 2.9.2 for a complete discussion of the effects of different **DEVSEL#** decode times and target initial wait states.

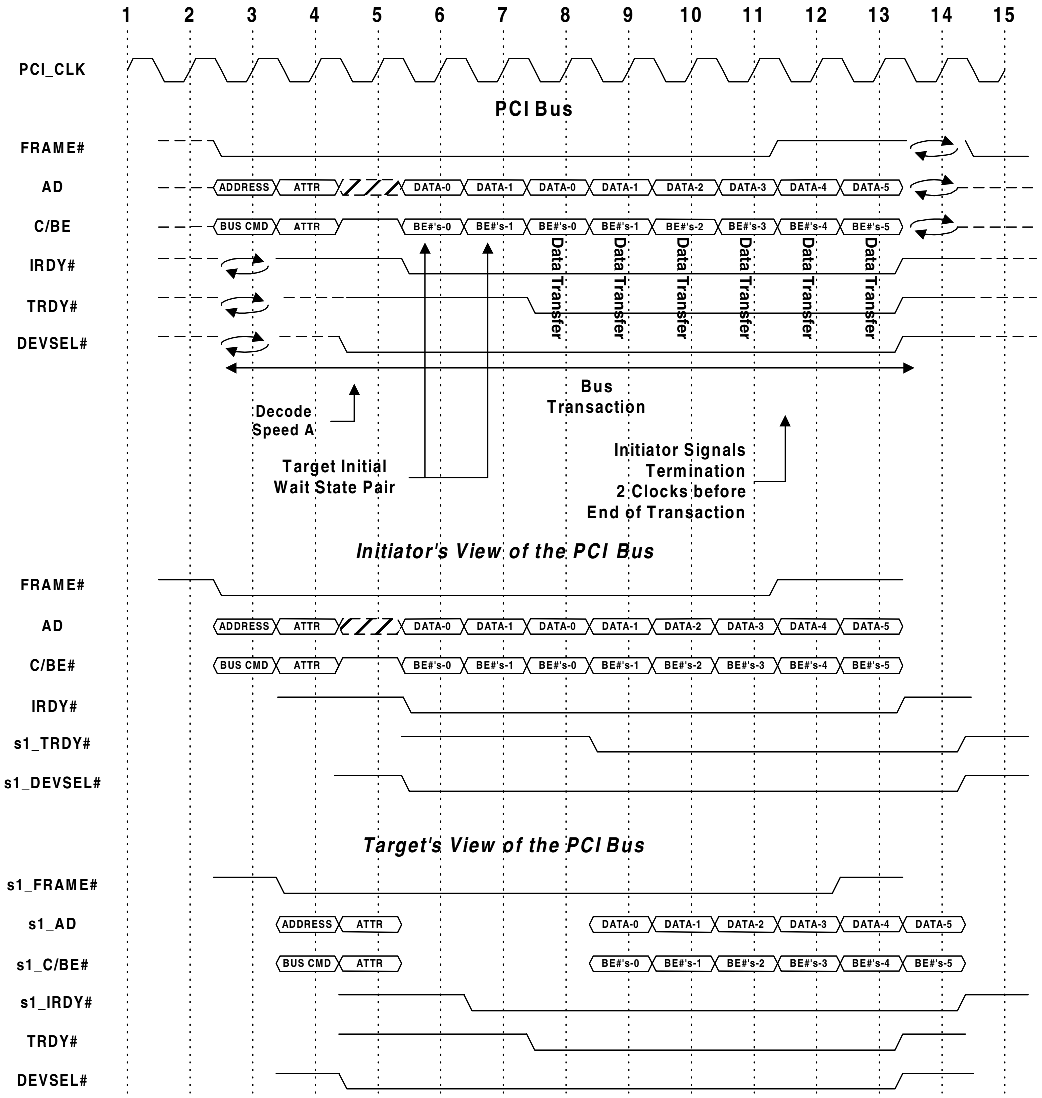


Figure 2-4: Burst Memory Write Transaction with Two Target Initial Wait States

2.6.2. Burst Reads

Burst read transactions use the Memory Read Block or Alias to Memory Read Block commands. For these transactions, the initiator is the source of the command, address, and attributes, and the completer is the source of the data.

The target is permitted to respond to a burst read transaction with any of the following:

- Split Response
- Target-Abort
- Single Data Phase Disconnect
- Wait State
- Data Transfer
- Retry
- Disconnect at Next ADB

If the target responds by signaling Single Data Phase Disconnect, Data Transfer, or Disconnect at Next ADB, data transfers during the read transaction. If the target responds with Split Response, no data transfers during the read transaction but is transferred later in one or more Split Completion transactions.

Implementation Note: Immediate Response from Memory Address with Read Side Effects

If a target responds immediately with data for more than a single data phase (i.e., signals Data Transfer or Disconnect at Next ADB) to a burst read transaction, the target and initiator are permitted only to disconnect the transaction on ADBs. If the address range being read includes any locations with read side effects (i.e., locations whose state changes when they are read), the target must not read those locations except when it is guaranteed that the initiator will accept the data. Therefore, in this case, the target cannot fetch beyond an ADB until the transaction crosses the ADB.

The target is encouraged to complete a read of such a location as a Split Transaction. The initiator is obligated always to accept the entire byte count of a Split Transaction. Split Transactions use the bus more efficiently than transactions with immediate response that are disconnected on each data phase or each ADB.

The **C/BE#** bus is reserved and driven high by the initiator after the attribute phase of all burst read transactions. All bytes between the starting and ending address inclusive are included in these transactions.

Figure 2-5 shows a burst read transaction in which the target signals Data Transfer. The target responds with the minimum **DEVSEL#** timing and target initial latency. The top portion of the diagram shows the signals as they appear on the bus. The middle and lower portions of the diagram show the bus from the viewpoint of the initiator and target, respectively. Signal names preceded by "s1_" indicate a signal that is internal to the device after the bus signal has been sampled.

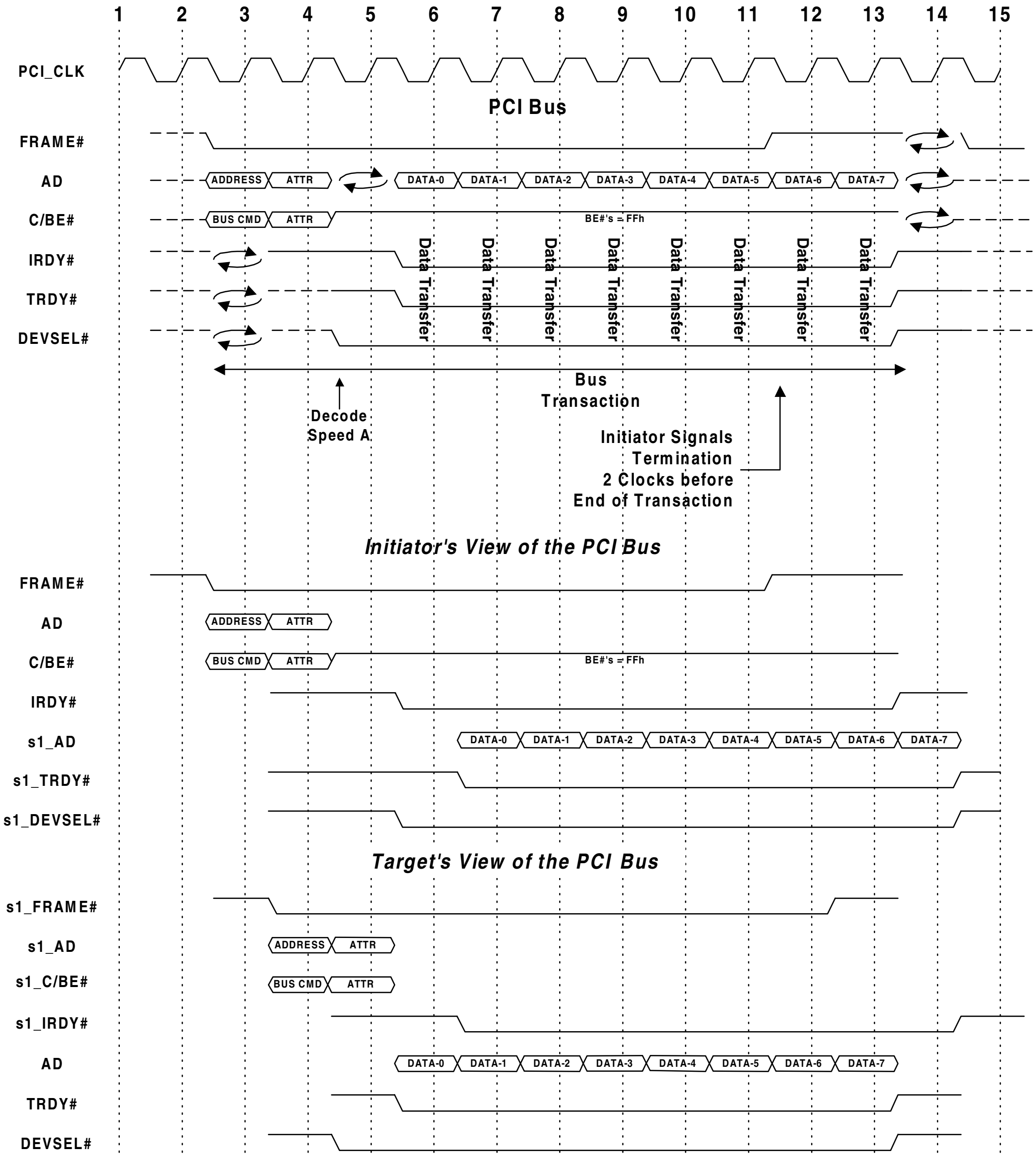


Figure 2-5: Burst Memory Read Transaction with No Target Initial Wait States

Figure 2-6 shows a similar transaction with minimum **DEVSEL#** decode timing but with only six data phases and an initial target latency of five clocks (two wait states).

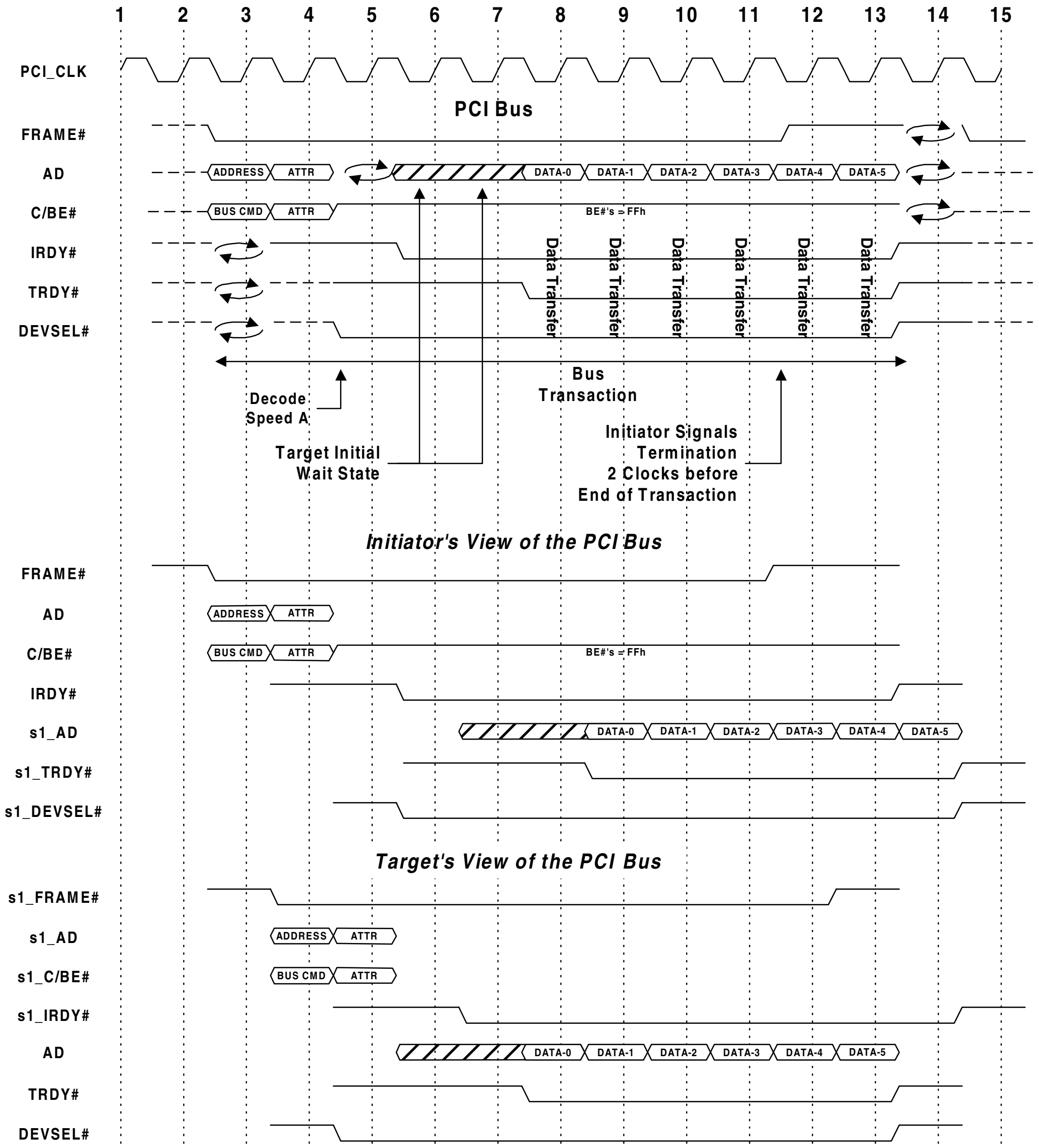


Figure 2-6: Burst Memory Read Transaction with Target Initial Wait States

2.7. DWORD Transactions

A DWORD transaction is any transaction that uses one of the following commands:

- Interrupt Acknowledge
- Special Cycle
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- Memory Read DWORD

DWORD transactions always have a single data phase and affect no more than a single DWORD. DWORD transactions do not include a byte count.

DWORD transactions must be initiated as 32-bit transfers. (**REQ64#** must be deasserted.) They do not use the upper bus halves (**AD[63::32]**, **C/BE[7::4]#**, **PAR64**) even when initiated by 64-bit devices.

DWORD transactions include explicit byte enables in the Requester Attributes. A byte is not affected by the transaction in any way if its byte enable is not asserted. All byte enable patterns are permitted, including no byte enables asserted. For I/O and DWORD memory transactions, **AD[1::0]** must correspond to the first byte enable asserted as specified in Section 2.3. The device that sources the data is required to drive all bits of the **AD[31::00]** bus during the data phase, even if some byte enables are deasserted. The value of the data driven on all the byte lanes is included in the generation of bus parity, even if some of the byte enables are deasserted. The **C/BE#** bus is reserved and driven high by the initiator after the attribute phase of all DWORD transactions.

All target terminations are permitted on DWORD transactions (see Section 2.11.2).

2.7.1. DWORD Memory and I/O Transactions

Memory transactions using the Memory Read DWORD command and I/O transactions are DWORD transactions. (Other DWORD transactions are discussed separately.)

Figure 2-7 shows an I/O write transaction for which the target signals Data Transfer (an Immediate Transaction). In this figure, the target does not insert any wait states, signaling Data Transfer on clock 6. As in conventional PCI, data is transferred when **TRDY#**, **IRDY#**, and **DEVSEL#** are asserted. However, the initiator continues driving the **AD[31::00]** and **C/BE[3::0]#** buses, and **FRAME#** and **IRDY#** remain asserted in clock 7, one clock past the end of the data phase. (A burst write with a single data phase looks identical.)

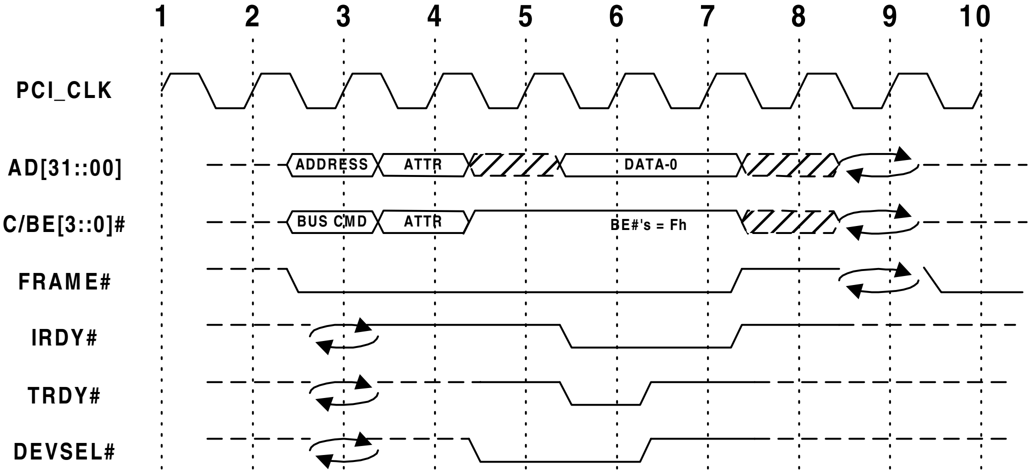


Figure 2-7: DWORD Write Transaction with No Wait States and Data Transfer

Figure 2-8 shows a DWORD read in which data is transferred (an Immediate Transaction). In this figure, the target inserts two wait states at clocks 6 and 7, then signals Data Transfer on clock 8. As in conventional PCI, data is transferred when **TRDY#**, **IRDY#**, and **DEVSEL#** are asserted. However, the initiator continues driving the **C/BE#** bus, and **FRAME#** and **IRDY#** remain asserted in clock 9, one clock past the end of the data phase. (A burst read with a single data phase and two initial wait states looks identical.)

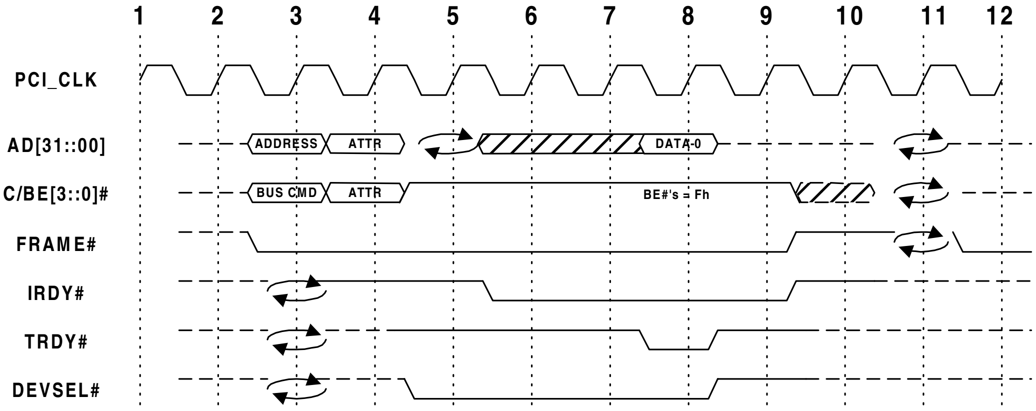


Figure 2-8: DWORD Read with Two Target Initial Wait States and Data Transfer

2.7.2. Configuration Transactions

As in conventional PCI, a Type 0 configuration transaction executes on a single bus segment and does not cross a PCI-X bridge. It must be claimed by a completer on that bus segment or terminated with Master-Abort. Type 1 configuration transactions cross PCI-X bridges and are converted to Type 0 configuration transactions or Special Cycle transactions the same as in conventional PCI.

In most respects, transactions using the Configuration Read and Configuration Write commands are the same as other DWORD transactions, however, they differ in two requirements. First, their timing is different in that the initiator drives the address on the **AD[31::0]** bus before it asserts **FRAME#**. Second, additional information is driven in the attribute phase of Type 0 configuration transactions. The following sections describe these requirements in more detail.

2.7.2.1. Configuration Transaction Timing

PCI-X initiators are required to drive the address of all configuration transactions on the **AD[31::00]** bus four clocks before asserting **FRAME#**. This allows additional propagation time for the **IDSEL** input signal for Type 0 configuration transactions. In those systems in which the **IDSEL** inputs are tied to **AD** bits, this allows time for the signal to rise through a series resistor on the system board. In those systems in which the **IDSEL** inputs are driven from separate outputs from the source bridge, this allows time for the signal to propagate from the **AD** bus through the source bridge to the **IDSEL** pins, if the requester is a device other than the source bridge. Timing for Type 1 configuration transactions is the same as Type 0, even though **IDSEL** is used only for Type 0. Once **FRAME#** is asserted, the rest of the transaction proceeds like any other DWORD transaction.

Figure 2-9 and Figure 2-10 illustrate a PCI-X Configuration Write transaction and a Configuration Read transaction respectively. Both figures show **DEVSEL#** decode timing A with two target initial wait states. All other **DEVSEL#** decode speeds and initial wait state combinations are also valid for configuration transactions. Transactions (both read and write) must not affect registers if that register's byte enable is deasserted.

Figure 2-9 and Figure 2-10 illustrate **IDSEL** valid at clock 7 with a valid configuration command. This is the **IDSEL** input of the device addressed by a Type 0 configuration transaction. The state of **IDSEL** at any other clock and during Type 1 configuration transactions must be ignored by the target.

As for all transactions, **GNT#** must be asserted in clock N-2 for the device to start a configuration transaction on clock N. (See Section 4.1 for a discussion of the arbiter.) However, for arbitration purposes, the bus appears idle while a device is driving the **AD** bus before asserting **FRAME#** for a configuration transaction. If the arbiter asserts **GNT#** to a device on clock N-2, the device starts driving the bus for a configuration transaction (on clock N, N+1, or N+2), and the arbiter deasserts **GNT#** before clock N+3, the device must not continue the configuration transaction. It must float the bus two clocks after **GNT#** is deasserted. In the following figures, this means that if the arbiter deasserts **GNT#** before clock 6, the device must discontinue the configuration transaction.

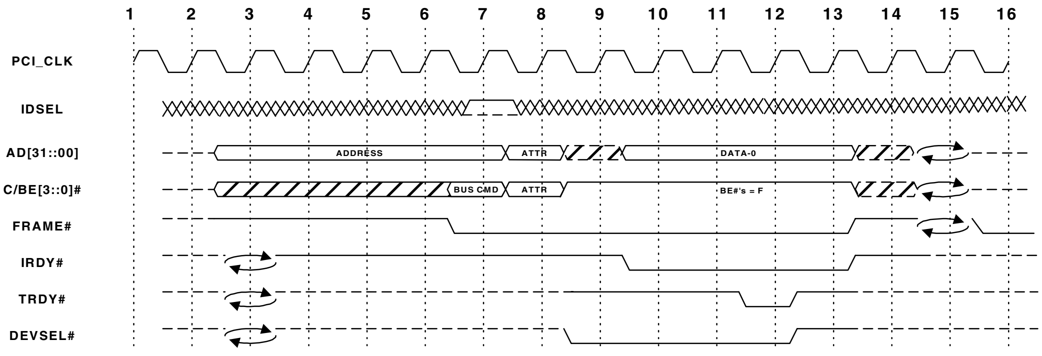


Figure 2-9: Configuration Write Transaction

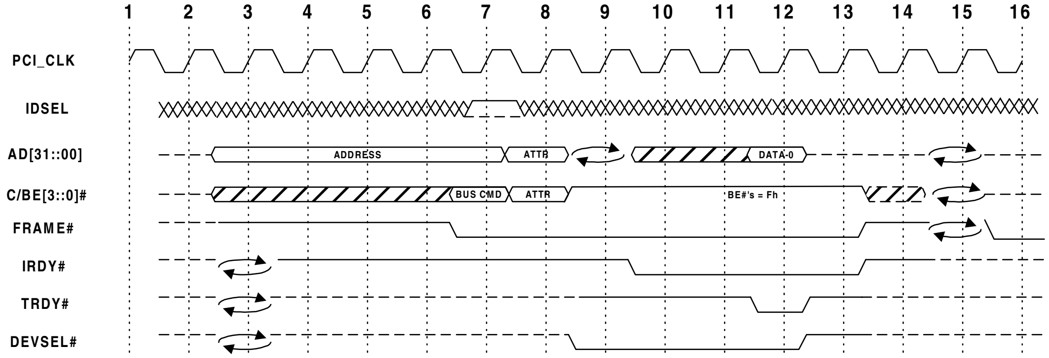


Figure 2-10: Configuration Read Transaction

2.7.2.2. Configuration Transaction Address and Attributes

PCI-X devices use Type 0 Configuration Write transactions to semi-automatically program their Bus Number and Device Number fields in the PCI-X Status register (see Section 7.2.4). PCI-X devices use the contents of these registers in the attribute fields of all transactions they initiate. To program these registers, all Type 0 configuration transactions (both read and write) include additional information in their address and attribute phases.

Figure 2-11 shows the format of the address for both Type 0 and Type 1 configuration transactions. The format for the conventional PCI Type 0 address is included for reference. Notice that the Device Number field is required both for Type 0 and Type 1 PCI-X configuration transactions. In Type 0 configuration transactions, bridges (including host bridges) not only drive the Device Number field during the address phase, but also decode the Device Number field and assert a single address bit in the range **AD[31::16]** during the address phase (for device numbers in the range 0 0000b-0 1111b) according to Table 2-7. (The target device updates the Bus Number and Device Number fields in its PCI-X Status register on every Configuration Write transaction. See Section 7.2.4.) The single bit enables the system designer to connect a different address bit to the **IDSEL** input of each device. The source bridge is not required to connect the **IDSEL** pins to **AD** bits electrically. For example, the source bridge is permitted to use separate output pins for individual **IDSEL** signals. However, the source bridge must guarantee that these pins are driven to the proper states with sufficient setup time during Type 0 configuration transactions from any initiator on the bus.

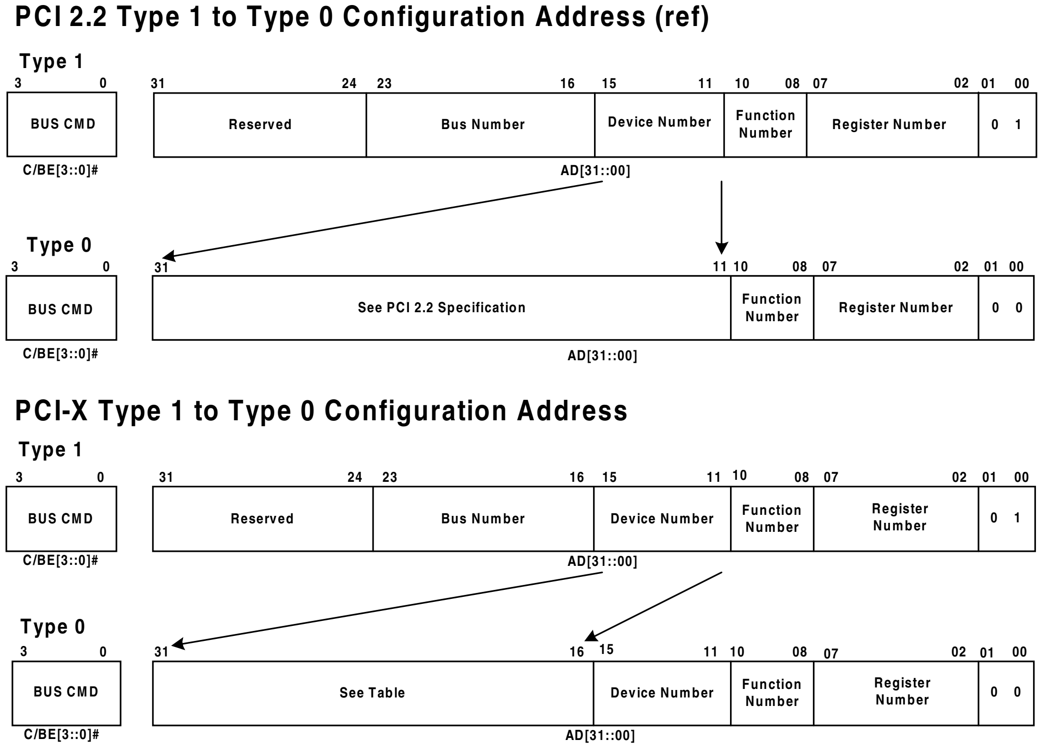
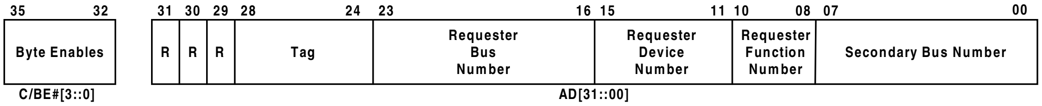


Figure 2-11: Configuration Transaction Address Format

Table 2-7: IDSEL Generation

Device Number	Address AD[31::16]
0 0000b	0000 0000 0000 0001b
0 0001b	0000 0000 0000 0010b
0 0010b	0000 0000 0000 0100b
0 0011b	0000 0000 0000 1000b
0 0100b	0000 0000 0001 0000b
0 0101b	0000 0000 0010 0000b
0 0110b	0000 0000 0100 0000b
0 0111b	0000 0000 1000 0000b
0 1000b	0000 0001 0000 0000b
0 1001b	0000 0010 0000 0000b
0 1010b	0000 0100 0000 0000b
0 1011b	0000 1000 0000 0000b
0 1100b	0001 0000 0000 0000b
0 1101b	0010 0000 0000 0000b
0 1110b	0100 0000 0000 0000b
0 1111b	1000 0000 0000 0000b
1 xxxxb	0000 0000 0000 0000b

Figure 2-12 shows the format of the Requester Attributes that are driven during the attribute phase of all Type 0 configuration transactions. (Type 1 configuration transactions use the standard DWORD format for the Requester Attributes described in Section 2.5.) The Requester Attributes for Type 0 configuration transactions are identical to Requester Attributes for other DWORD transactions except that bits 7-0 contain the Secondary Bus Number field. (The figure also shows the No Snoop and Relaxed Ordering bits as reserved, since the initiator is permitted to set these bits only for memory transactions. See Section 2.5.) The Secondary Bus Number field contains the number of the bus on which the Type 0 configuration transaction is executing. As in all Requester Attributes, the Requester Bus Number is the number of the bus on which the configuration transaction originated. If the transaction originated as a Type 1 configuration transaction and was converted to a Type 0 by a PCI-X bridge, the Requester Bus Number and the Secondary Bus Number are different. In this case, the PCI-X bridge inserts the contents of its Secondary Bus Number register in the Secondary Bus Number field of the Requester Attributes, when it converts the Type 1 transaction to a Type 0.



35 32	31	30	29	28 24	23 16	15 11	10 08	07 00
Byte Enables	R	R	R	Tag	Requester Bus Number	Requester Device Number	Requester Function Number	Secondary Bus Number
C/BE#[3::0]	AD[31::00]							

Figure 2-12: Type 0 Configuration Transaction Requester Attribute Bit Assignments

2.7.3. Special Cycle Transactions

The Special Cycle command provides a simple message broadcast mechanism in PCI-X mode the same as in conventional PCI mode. Devices that support Special Cycle commands monitor only for the command code on **C/BE[3::0]#** during the address phase to detect a Special Cycle command. Such devices are permitted also to utilize the PCI-X attribute fields to further define the transaction.

In PCI-X mode, Special Cycle transactions are DWORD write transactions (only one data phase) with no initiator wait states (unlike conventional PCI that allowed initiator wait states). As in conventional PCI, the value on **AD[31::00]** during the address phase is not an address and no device asserts **DEVSEL#** (devices disable comparison to Base Address registers when **C/BE[3::0]#** contain the Special Cycle command value). Master-Abort is the normal termination of Special Cycle transactions and no error is reported for this case of Master-Abort termination. See Figure 2-13.

Special Cycle transactions use the DWORD requester attributes illustrated in Figure 2-2, and all byte enables are asserted, as they are in conventional PCI.

Like conventional PCI, **AD[31::00]** during the data phase contain the message type and an optional data field. The message is encoded on **AD[15::00]** and the optional data field on **AD[31::16]**. Message types for PCI-X are the same as for conventional PCI (see PCI 2.2 Appendix A).

As in conventional PCI, all devices are permitted to initiate Special Cycle transactions. Special Cycle transactions affect only those devices on one bus segment. Bridges do not forward Special Cycles. If an initiator desires to generate a Special Cycle transaction on a specific bus in the hierarchy, it must use a Type 1 Configuration Write transaction to do so. Type 1 Configuration Write transactions transverse PCI-X bridges in both directions for the purpose of generating Special Cycle commands on any bus in the hierarchy.

Type 1 Configuration Write transactions are converted to Special Cycle transactions by PCI-X bridges the same in PCI-X mode as they are in conventional mode. If a Type 1 Configuration Write contains a Device Number of all ones, a Function Number of all ones, and a Register Number of all zeros, the PCI-X bridge that forwards the transaction to the appropriate bus converts the command on **C/BE[3::0]#** from Configuration Write to Special Cycle and drives address bits **AD[31::16]** to zeros (no **IDSEL** is asserted).

The same optional methods for software to generate a Special Cycle transaction are available in PCI-X mode as defined in PCI 2.2 for conventional mode.

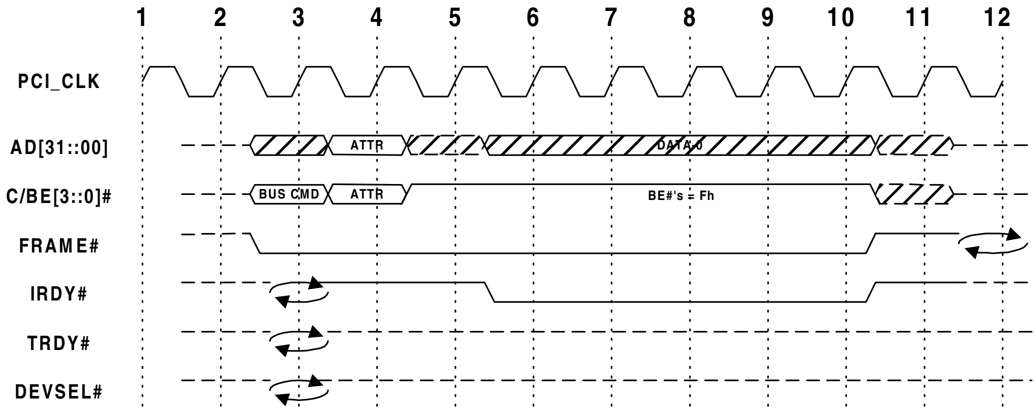


Figure 2-13: Special Cycle

2.7.4. Interrupt Acknowledge Transactions

An Interrupt Acknowledge transaction appears on the bus in PCI-X mode the same as DWORD memory or I/O read transactions, except the command is Interrupt Acknowledge. As in conventional PCI, the Interrupt Acknowledge has no address, so the initiator drives any value on the **AD[31::00]** bus during the address phase. As for all other DWORD transactions, the initiator includes the appropriate byte enables in the Requester Attributes, and **C/BE[3::0]#** are reserved and driven high during the data phase. **DEVSEL#**, wait state, target termination, and Split Transaction requirements are the same as for other DWORD transactions.

2.8. Device Select Timing

PCI-X targets are required to claim transactions by asserting **DEVSEL#** and leaving **TRDY#** and **STOP#** deasserted, using decode A, B, C, or Subtractive as shown in Table 2-8 and Figure 2-14. Conventional **DEVSEL#** timing is shown in the table for reference.

Table 2-8: DEVSEL# Timing

Decode Speed	PCI-X	Conventional PCI (ref)
1 clock after address phase(s)	Not Supported	Fast
2 clocks after address phase(s)	Decode A	Medium
3 clocks after address phase(s)	Decode B	Slow
4 clocks after address phase(s)	Decode C	Subtractive
5 clocks after address phase(s)	N/A	N/A
6 clocks after address phase(s)	Subtractive	N/A

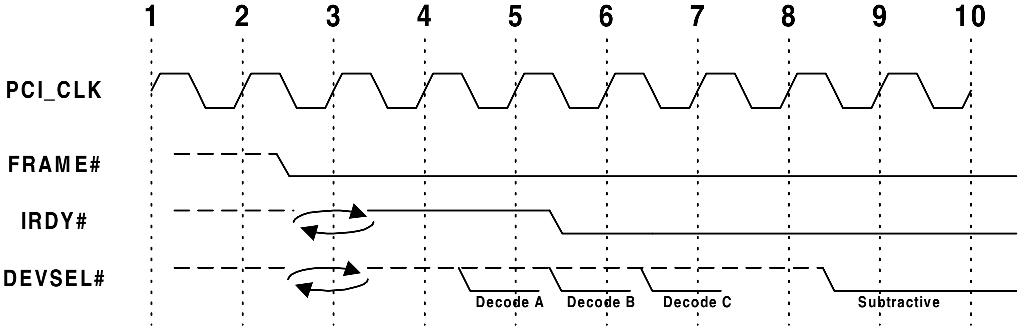


Figure 2-14: DEVSEL# Timing

If the transaction uses a dual address cycle, the decode speeds are measured from the second address phase. If no target asserts **DEVSEL#** within the Subtractive decode time, the initiator ends the transaction as a Master-Abort.

After a target asserts **DEVSEL#,** it must complete the transaction with one or more data phases by signaling one or more of the following: Split Response, Target-Abort, Single Data Phase Disconnect, Wait State, Data Transfer, Retry, or Disconnect at Next ADB.

2.8.1. Writes and Split Completions

The figures in this section illustrate device select timing using burst write transactions with four data phases. Split Completion transaction timing is identical to the burst write transactions shown. Burst transactions of different length and DWORD transactions use the same device select timing.

The figures illustrate that the initiator advances to the second data value of the burst two clocks after the target asserts **DEVSEL#** (see Section 2.9.2 for the effects of wait states on burst write data). For DWORD write transactions, there is only one data value, which remains constant from the second clock after the attribute phase until the end of the transaction.

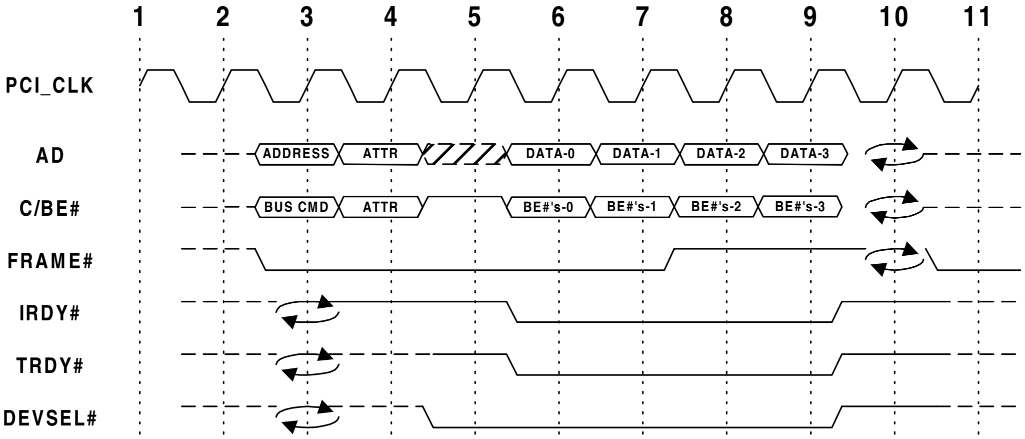


Figure 2-15: Burst Write with DEVSEL# Decode A and No Initial Wait States

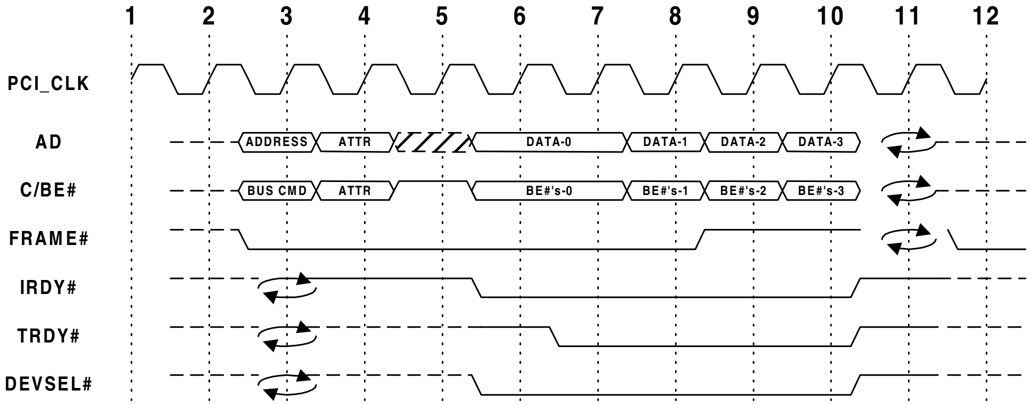


Figure 2-16: Burst Write with DEVSEL# Decode B and No Initial Wait States

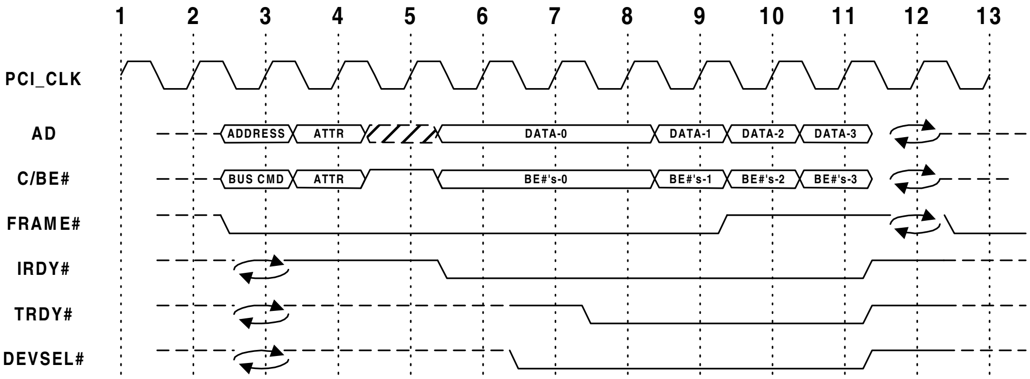


Figure 2-17: Burst Write with DEVSEL# Decode C and No Initial Wait States

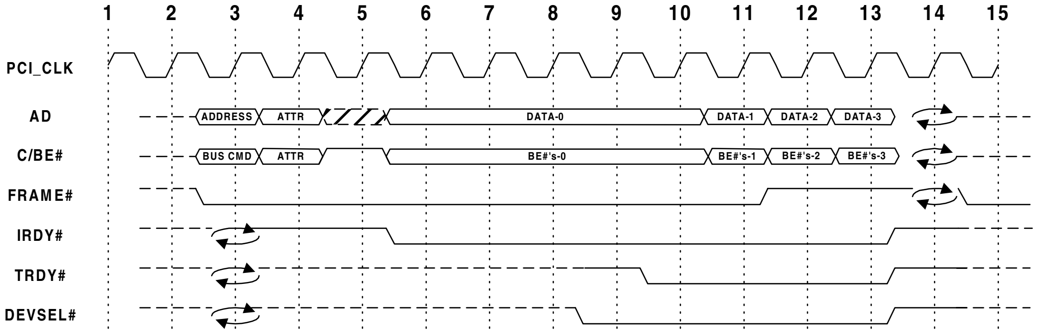


Figure 2-18: Burst Write with Subtractive DEVSEL# Decode and No Initial Wait States

2.8.2. Reads

The figures in this section illustrate device select timing using burst read transactions in which the target signaled Data Transfer for four data phases. Burst transactions of different lengths and DWORD transactions use the same device select timing.

The figures illustrate that for a burst read transaction, the **C/BE#** bus is reserved and driven high after the attribute phase, which is also true for DWORD transactions. The **AD** bus is shown floating during the target response phase after the turn-around clock, but is also permitted to be driven to any value by the target once the target has decoded its address (after the turn-around clock).

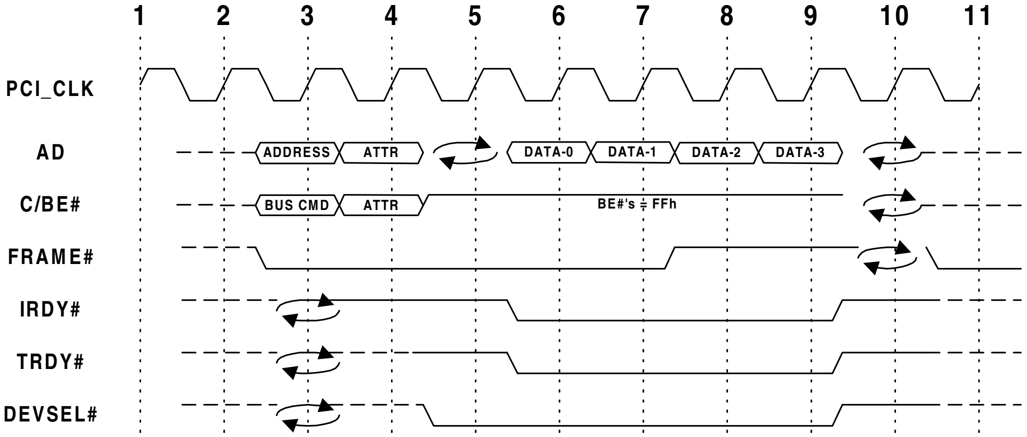


Figure 2-19: Burst Read with DEVSEL# Decode A and No Initial Wait States

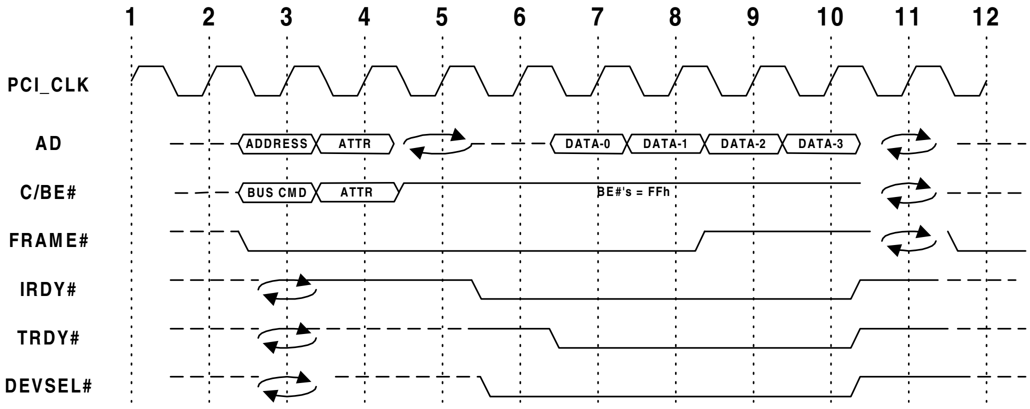


Figure 2-20: Burst Read with DEVSEL# Decode B and No Initial Wait States

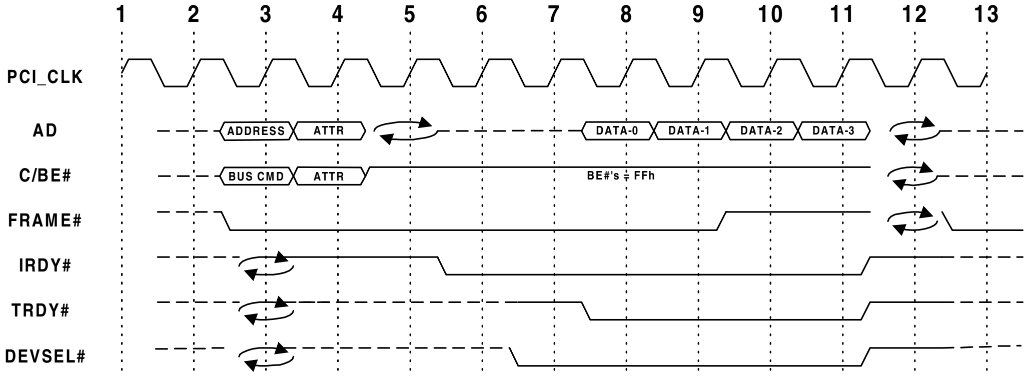


Figure 2-21: Burst Read with DEVSEL# Decode C and No Initial Wait States

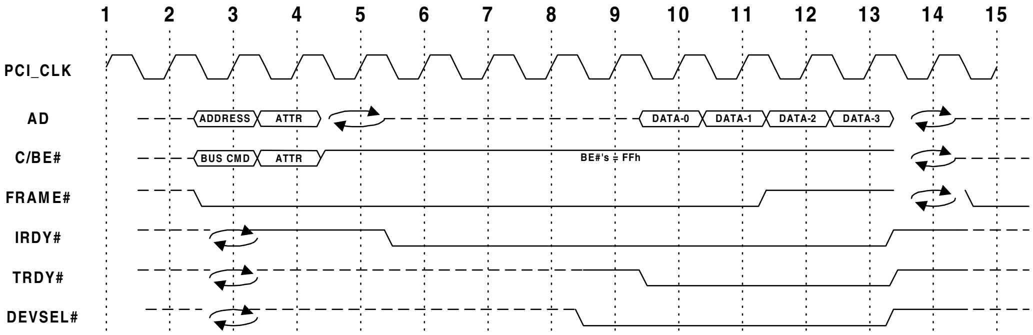


Figure 2-22: Burst Read with Subtractive DEVSEL# Decode and No Initial Wait States

2.9. Wait States

PCI-X initiators are not permitted to insert wait states. Initiators are required to assert **IRDY#** two clocks after the attribute phase. The initiator must drive write data and must be prepared to accept read data when **IRDY#** is asserted. After **IRDY#** is asserted, it must remain asserted until the end of the transaction.

PCI-X targets are permitted to insert wait states only on the initial data phase. If the transaction has more than one data phase, the target must not signal Wait State (see Section 2.11.2) after it has signaled anything else on a data phase. Targets are permitted to insert initial wait states only in pairs of clocks for burst write and Split Completion transactions for which data transfers (the target signals Data Transfer, Disconnect at Next ADB, or Single Data Phase Disconnect). (Pairs of wait states are necessary to allow the initiator to toggle between the first and second data patterns, as described in Section 2.9.2.) For these transactions, the target must signal Wait State an even number of times (zero or more) after asserting **DEVSEL#** (up to the maximum specified below). For all other transactions (including burst write and split completion transactions for which the target signals Retry or Target-Abort), the target is permitted to signal Wait State any number of times (zero or more) after asserting **DEVSEL#** (up to the maximum specified below).

2.9.1. Target Initial Latency

As defined in PCI 2.2, target initial latency is measured from the clock in which the initiator asserts **FRAME#** to the clock in which the target signals something other than Wait State. Table 2-9 shows the target initial latency for all the combinations of **DEVSEL#** timing, numbers of address phases, and numbers of wait states.

Table 2-9: Target Initial Latency

	Single Address Cycle				Dual Address Cycle			
	DEVSEL# Timing				DEVSEL# Timing			
Wait states	A	B	C	Sub	A	B	C	Sub
0	3	4	5	7	4	5	6	8
1	4	5	6	8	5	6	7	9
2	5	6	7	9	6	7	8	10
3	6	7	8	10	7	8	9	11
4	7	8	9	11	8	9	10	12
5	8	9	10	12	9	10	11	13
6	9	10	11	13	10	11	12	14
7	10	11	12	14	11	12	13	15
8	11	12	13	15	12	13	14	16
9	12	13	14	16	13	14	15	N/A
10	13	14	15	N/A	14	15	16	N/A
11	14	15	16	N/A	15	16	N/A	N/A
12	15	16	N/A	N/A	16	N/A	N/A	N/A
13	16	N/A	N/A	N/A	N/A	N/A	N/A	N/A

The maximum number of initial wait states the target is permitted to insert depends upon how the target terminates the transaction. If the target signals Split Response or Retry, the target must do so within eight clocks of the assertion of **FRAME#**. If the target signals Target-Abort in the first data phase for reasons other than an error that prevents transferring the first data, the target must do so within eight clocks of the assertion of **FRAME#**. These cases are outlined with a heavy line in Table 2-9. If the target signals Single Data Phase Disconnect, or signals Data Transfer or Disconnect at Next ADB in the first data phase, the target must do so within 16 clocks of the assertion of **FRAME#**. If the target intends to signal Single Data Phase Disconnect, Data Transfer, or Disconnect at Next ADB but an error condition prevents it from transferring the data for the first data phase, the target is permitted to signal Target-Abort within 16 clocks of the assertion of **FRAME#**. Unlike conventional PCI, all PCI-X targets (including the host bridge) are subject to the same target initial latency limits.

PCI-X devices are exempt from the target initial latency requirements in the following cases:

- The device initialization time after the rising edge of **RST#** (T_{rhfa} specified in Table 9-5) has not elapsed. As in conventional PCI, the device is permitted to ignore such a transaction or to assert **DEVSEL#** and signal Wait State or Retry until the end of the initialization time.
- System power-up initialization software is copying an expansion ROM image from the device into system memory. No upper limit is specified for target initial latency for such transactions.

Implementation Note: Expansion ROM Accesses after a Hot-Insertion Event

As described in PCI HP 1.0, most operating systems do not permit the execution of software in expansion ROMs after a hot insertion event. Expansion ROM software is generally created to execute only at system initialization time. If a device requires access to its expansion ROM after the device is hot-inserted, the device must comply with the target initial latency limits for those transactions.

PCI-X devices have similar options to those defined in PCI 2.2 for meeting the target initial latency for transactions other than Split Completions. See Section 2.13 for additional requirements for Split Completions.

Option 1: The device always transfers data within the target initial latency limits listed in Table 2-9. (Same as PCI 2.2 except the latency limit is lower for some target terminations.)

Option 2: The device normally transfers data within the target initial latency limit listed in Table 2-9, but under some conditions that are guaranteed to resolve quickly, execution of the transaction would take longer. Under these conditions, the device is permitted to signal Retry within the limits listed in Table 2-9. If the initiator repeats the transaction, and the transaction is a memory write or an I/O Write, the device must complete the transaction with something other than Retry within the Maximum Completion Time specified in Section 2.13. (Same as PCI 2.2 except the latency limit is lower.)

Option 3: The device frequently cannot transfer data within the target initial latency limit in Table 2-9. The device must signal Split Response and execute the transaction as a Split Transaction. (Split Transactions in PCI-X replace Delayed Transactions in conventional PCI.)

Implementation Note: Minimizing the Use of Wait States and Retry

It is recommended that devices minimize the target initial latency. Device designers are encouraged to use the minimum device select decode time and never to insert wait states. If wait states cannot be avoided, the number of wait states must be kept to a minimum. If large numbers of wait states are required, executing the transaction as a Split Transaction generally provides more efficient use of the bus. Even in high-frequency systems with few (maybe only one) slots on the bus, Split Transactions allow multi-threaded devices (e.g., multiple devices behind a PCI-X bridge) to issue multiple transactions concurrently.

Devices are encouraged never to signal Retry. If a temporary condition prevents the device from executing the transaction, signaling Retry is acceptable if there is a high probability that the device will be able to execute the transaction within the target initial latency limit if the initiator repeats the transaction later. If the device frequently cannot transfer data within the limits shown in Table 2-9, executing the transaction as a Split Transaction generally provides more efficient use of the bus.

In most cases, there is no guarantee that the initiator will repeat a transaction terminated with Retry. (Completers are always obligated to send data for the full byte count or send a Split Completion Message. Bridges are obligated to forward certain transactions.) Delayed Transactions are not supported in PCI-X.

2.9.2. Wait States on Writes and Split Completions

For write and Split Completion transactions, the initiator must drive data on the **AD** bus two clocks after the attribute phase. If the transaction is a burst with more than one data phase, the initiator advances to the second data value two clocks after the target asserts **DEVSEL#**. If the target also inserts wait states, the initiator must toggle between its first and second data values until the target signals something other than Wait State. See Section 2.12.3 for requirements for a 64-bit initiator to copy data from the upper to the lower bus half to support writing to 32-bit targets. If the target signals Data Transfer, Disconnect at Next ADB, or Single Data Phase Disconnect, it must do so an odd number of clocks after it asserts **DEVSEL#**. If the transaction has a third data value, the initiator advances to it two clocks after the target signals Data Transfer for the first data phase.

The starting address and byte count of some burst transactions is such that the transaction has only a single data phase. If the target inserts wait states on such a burst write transaction, the initiator is permitted to drive any value for the second data pattern.

For DWORD write transactions, the initiator drives the single data value on the **AD[31::00]** bus two clocks after the attribute phase and holds it there until the end of the transaction regardless of **DEVSEL#** timing and target initial wait states.

Figure 2-23 shows a write transaction with four data phases and two wait states. With minimum device select decode timing, the initiator advances immediately from DATA-0 to DATA-1. But since the target did not assert **TRDY#** in clock 6, neither DATA-0 nor DATA-1 were transferred, and the initiator repeats them in clocks 8 and 9. Since the target asserted **TRDY#** in clock 8, the initiator advances to DATA-2 in clock 10.

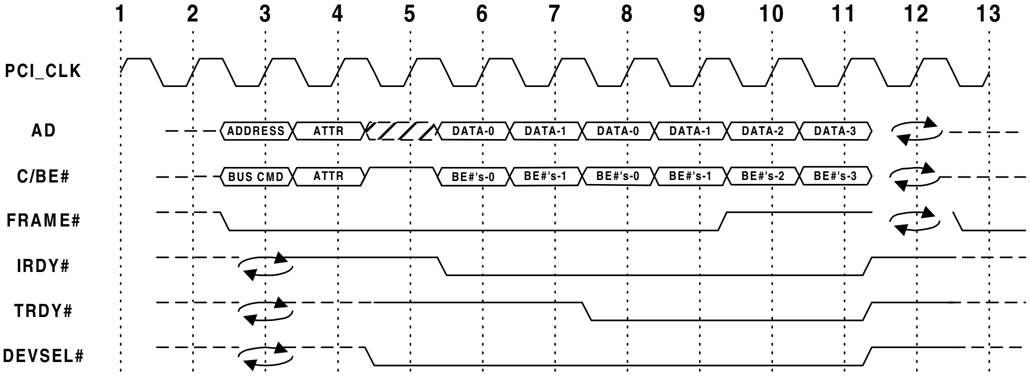


Figure 2-23: Burst Write Transaction with DEVSEL# Decode A and Two Initial Wait States

Figure 2-24 shows a similar transaction with four target initial wait states. The initiator toggles DATA-0 and DATA-1 at clocks 6, 7, 8, and 9, until the target asserts **TRDY#** at clock 10.

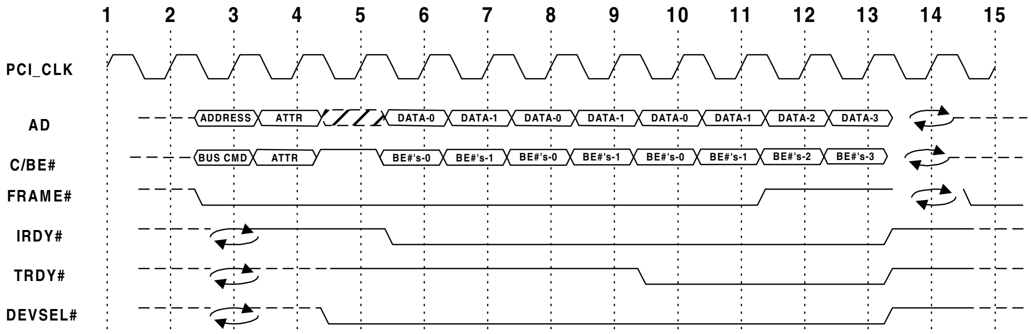


Figure 2-24: Burst Write Transaction with DEVSEL# Decode A and Four Initial Wait States

The following sequence of figures shows the effects of longer device select decode timing and wait states on the initiator of a burst write. Figure 2-25 and Figure 2-26 show device select timing B, and Figure 2-27 and Figure 2-28 shows timing C. In each case, the initiator advances to DATA-1 two clocks after the target asserts **DEVSEL#**, but toggles between DATA-0 and DATA-1 and does not advance to DATA-2 until two clocks after the target asserts **TRDY#**. Since the target asserts **DEVSEL#** later in these cases, the initiator drives DATA-0 for more than one clock.

Implementation Note: Device Select Timing A and B and Wait States on Burst Write Transactions

The target is permitted to insert wait states on burst write and Split Completion transactions only in pairs of clocks (for transactions that successfully transfer data). Therefore, device select decode speed B with no initial wait states is faster than a device select decode of A with the next fewer number (two) of target initial wait states. See Table 2-9.

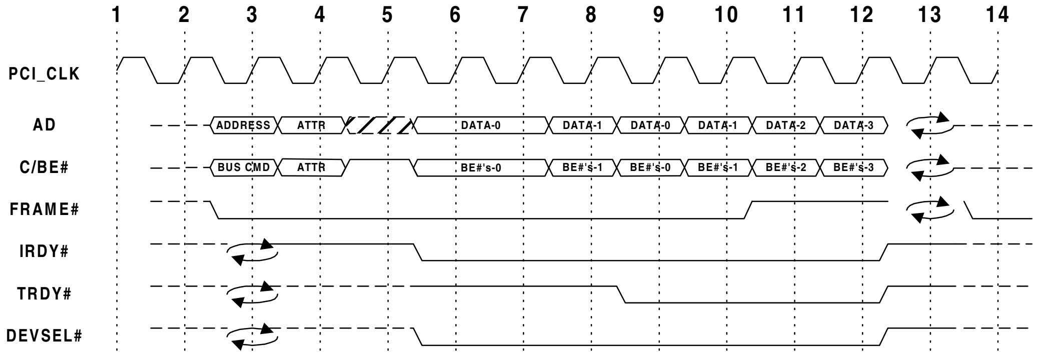


Figure 2-25: Burst Write Transaction with DEVSEL# Decode B and Two Initial Wait States

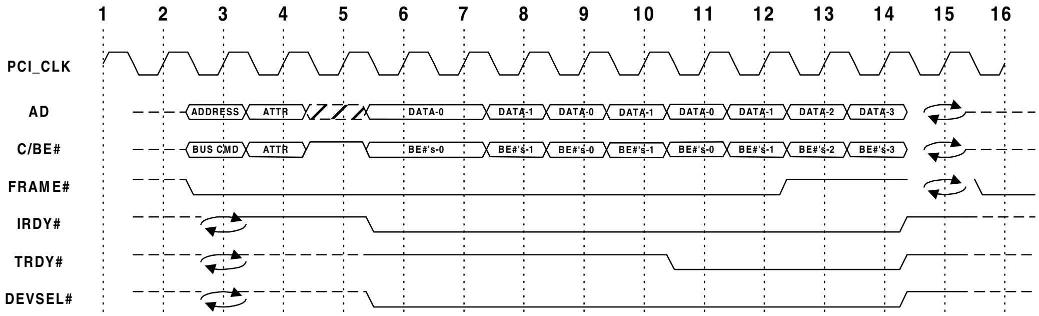


Figure 2-26: Burst Write Transaction with DEVSEL# Decode B and Four Initial Wait States

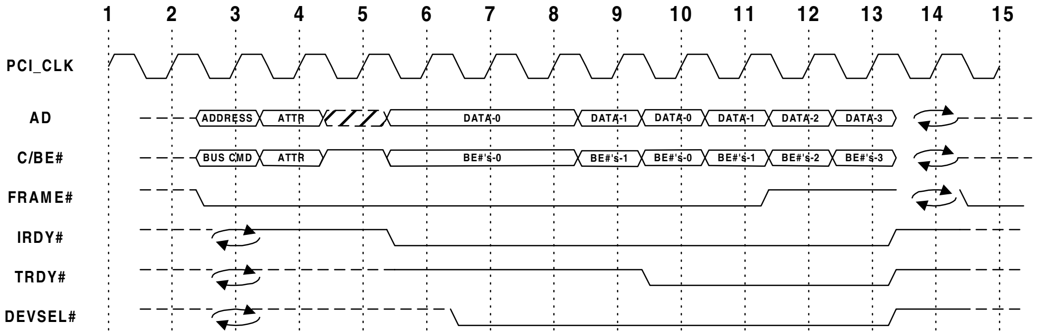


Figure 2-27: Burst Write Transaction with DEVSEL# Decode C and Two Initial Wait States

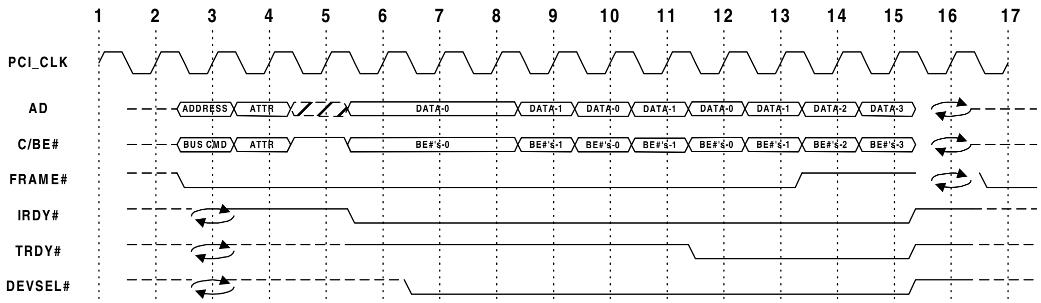


Figure 2-28: Burst Write Transaction with DEVSEL# Decode C and Four Initial Wait States

Figure 2-29 and Figure 2-30 show the effects of device select timing and wait states on DWORD write transactions. In both figures, the initiator drives the single write data value two clocks after the attributes and keeps driving it until the end of the transaction.

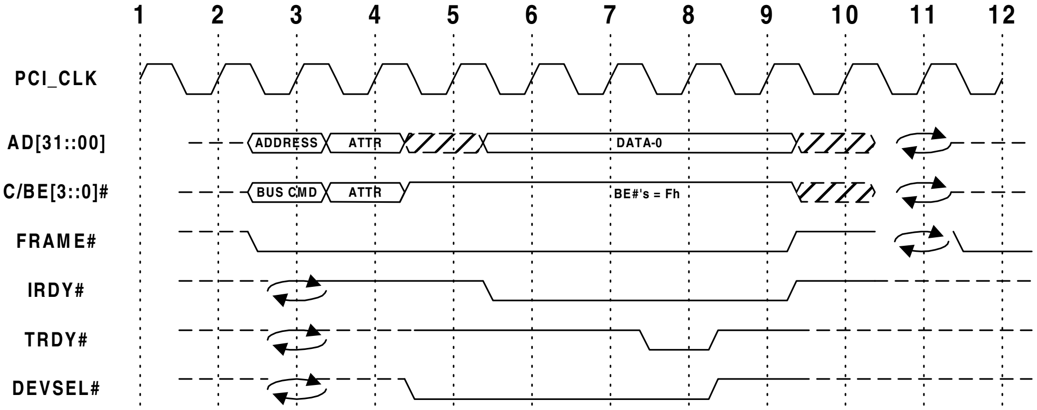


Figure 2-29: DWORD Write Transaction with DEVSEL# Decode A and Two Initial Wait States

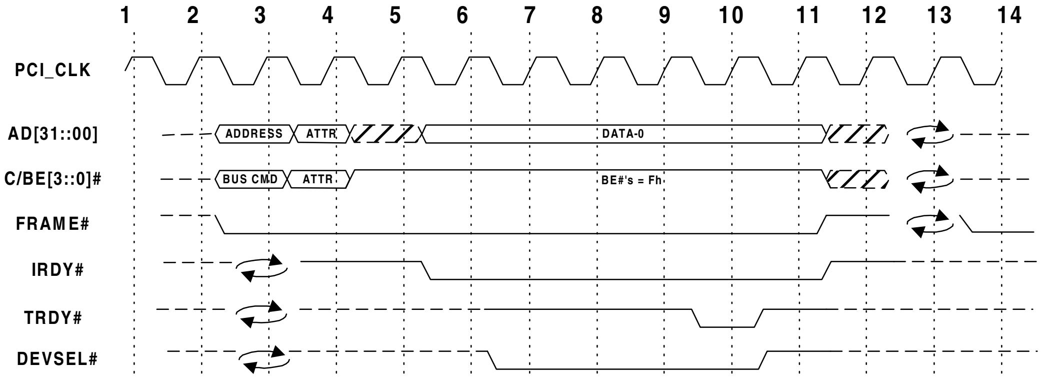


Figure 2-30: DWORD Write Transaction with DEVSEL# Decode C and Two Initial Wait States

2.9.3. Wait States on Reads

On a read transaction, the target is permitted to insert any number of initial wait states up to the maximum specified in Section 2.9.1. (Wait states are not required to be inserted in pairs for read transactions.)

The following figures show some of the possible combinations of **DEVSEL#** timing and wait states to illustrate how they effect the way the target drives the **AD** bus during the data phase. Figure 2-31 through Figure 2-35 show burst read transactions in which data is transferred (Immediate Transactions). Figure 2-36 and Figure 2-37 show DWORD read transactions in which data is transferred (Immediate Transactions). Transactions that the target ends with Split Response, Retry, and Single Data Phase Disconnect begin the same as shown below but end as shown in Section 2.11.2. Figures with **DEVSEL#** decode time other than A show the **AD** bus floating during the target response phase after the turn-around clock. In these cases, the **AD** bus is also permitted to be driven to any value by the target once the target has decoded its address (after the turn-around clock).

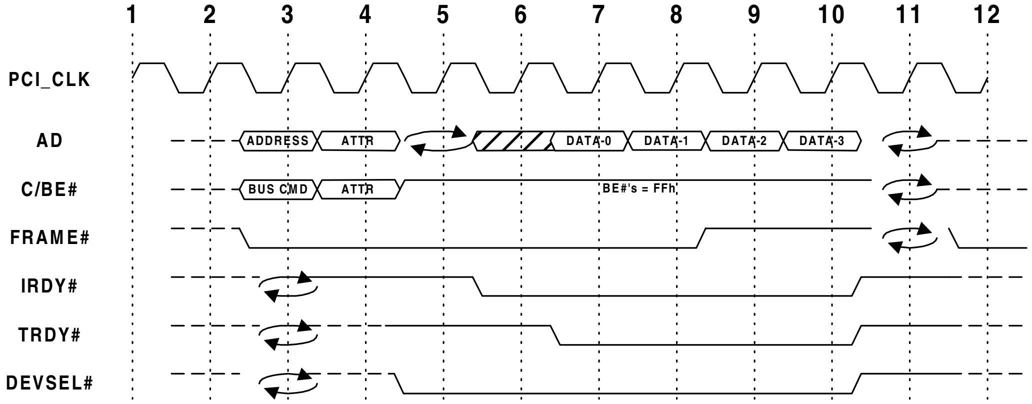


Figure 2-31: Burst Read Transaction with DEVSEL# Decode A and One Initial Wait State

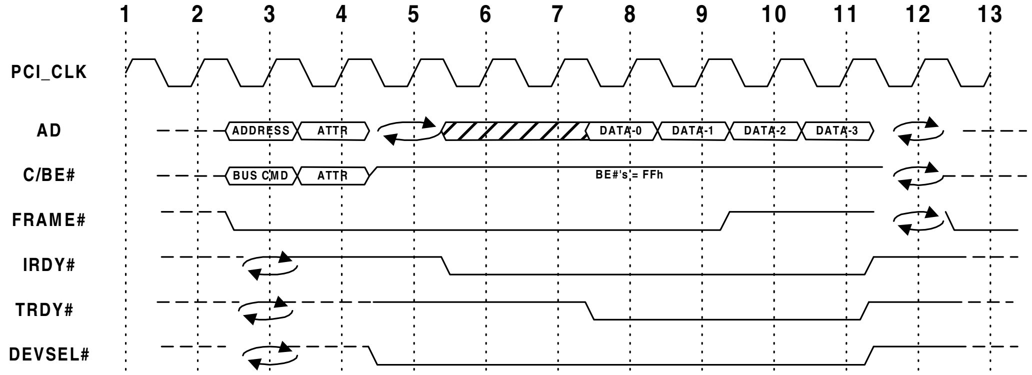


Figure 2-32: Burst Read Transaction with DEVSEL# Decode A and Two Initial Wait States

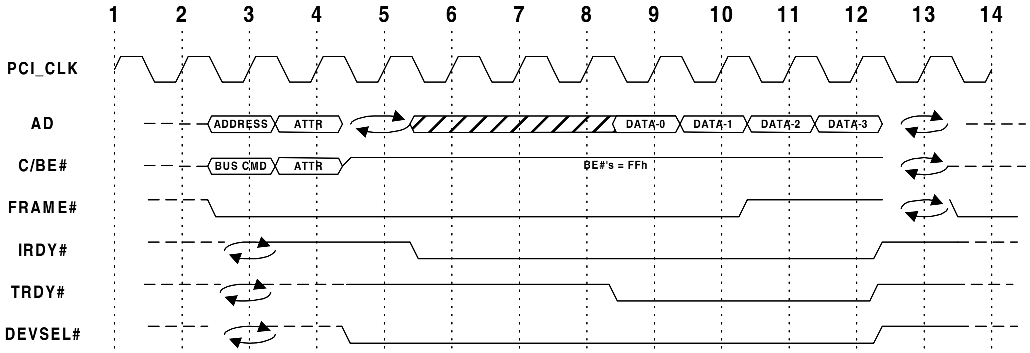


Figure 2-33: Burst Read Transaction with DEVSEL# Decode A and Three Initial Wait States

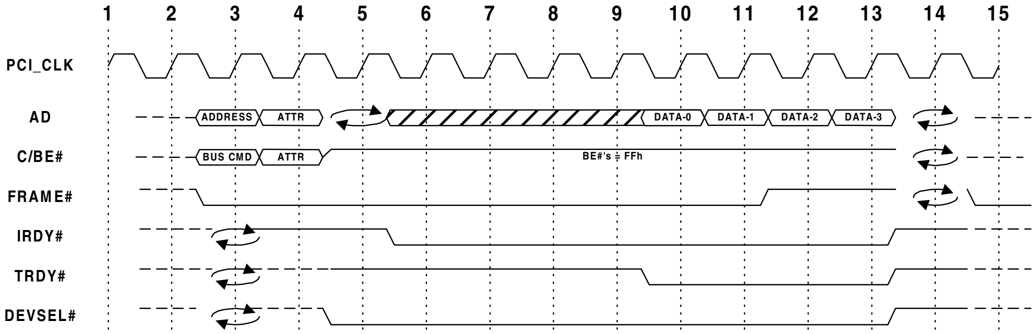


Figure 2-34: Burst Read Transaction with DEVSEL# Decode A and Four Initial Wait States

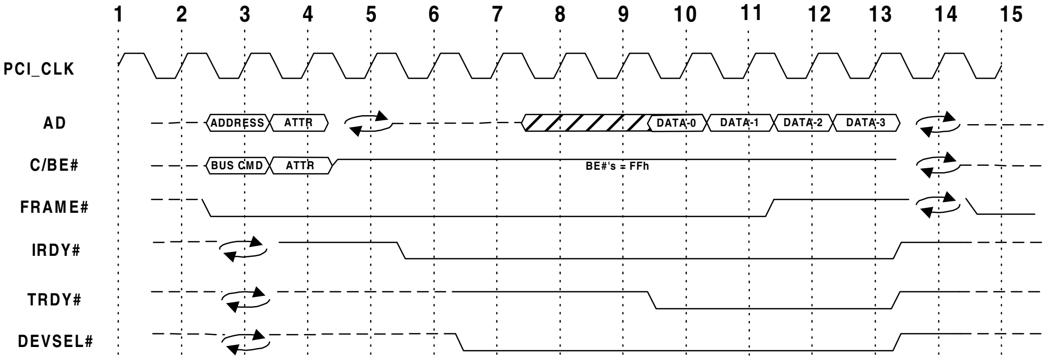


Figure 2-35: Burst Read Transaction with DEVSEL# Decode C and Two Initial Wait States

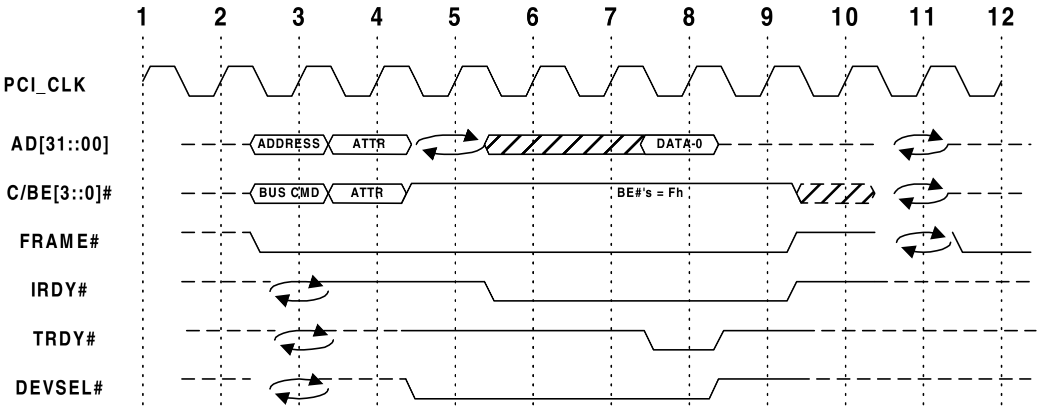


Figure 2-36: DWORD Read Transaction with DEVSEL# Decode A and Two Initial Wait States

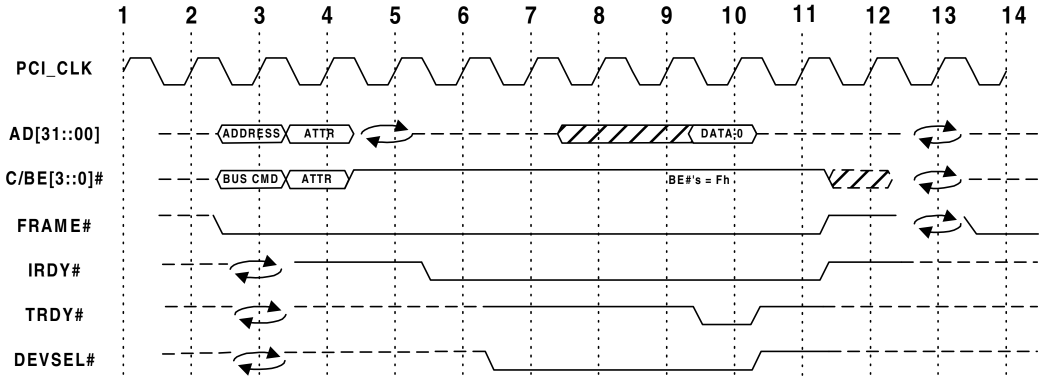


Figure 2-37: DWORD Read Transaction with DEVSEL# Decode C and Two Initial Wait States

2.10. Split Transactions

Split Transactions improve bus efficiency for transactions accessing targets that exhibit long latencies. Split Transactions in PCI-X systems replace Delayed Transactions in conventional PCI systems. Unlike conventional PCI, the target must not assume the initiator will repeat a transaction terminated with Retry. (In some cases, the device is obligated to continue the transaction, but the target must not depend upon the transaction being repeated verbatim. For example, if a completer encountered an error after a Split Completion transaction for a burst read that was terminated with Retry, the completer would be permitted to continue the Sequence with a Split Completion Error rather than repeating the original Split Completion.)

2.10.1. Basic Split Transaction Requirements

A Split Transaction consists of at least two separate bus transactions, a Split Request initiated by the requester, and one or more Split Completions initiated by the completer.

Transactions using any of the following commands are permitted to use Split Transactions:

- Memory Read Block
- Alias to Memory Read Block
- Memory Read DWORD
- Interrupt Acknowledge
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write

The target of such a transaction may optionally complete the transaction as a Split Transaction or may use any other termination method as determined by the rules for those termination methods. All of these termination alternatives are available regardless of whether the transaction was previously terminated with Retry. (See Section 2.11.2.5 for examples of implementations in which the device is unable to respond with Split Response and signals Target-Abort.) Once the target terminates a read transaction with Split Response, the target must transfer the entire requested byte count as a Split Completion (except for error conditions described in Section 2.10.6.2 and Section 8.8).

A Split Transaction begins when the requester initiates a transaction using one of the commands in the list above. The completer optionally signals Split Response as defined in Section 2.11.2.4. (PCI-X bridges are required to signal Split Response in some cases. See Section 8.4.)

Target initial wait states for Split Response termination are allowed up to the limit specified in Section 2.9.1. A transaction terminated with Split Response is called a Split Request.

After signaling Split Response, the completer executes the transaction. If the transaction is a write, the completer updates the bytes specified by the byte enables of the Split Request. If the transaction is a read, the completer prepares all or some of the bytes specified by the byte count (for burst reads) or byte enables (for DWORD reads) of the Split Request. The completer initiates a Split Completion transaction to send the requested read data or a completion message to the requester. Notice that for a Split Completion transaction, the requester and the completer switch roles. The completer becomes the initiator of the Split Completion transaction, and the requester becomes the target.

A device's ability to execute a transaction as a Split Transaction is unaffected by the state of the Bus Master bit in the Command register. A device that is properly addressed by a transaction is permitted to terminate that transaction with Split Response, request the bus, and initiate a Split Completion even if its Bus Master bit in the Command register is cleared.

A Split Transaction is not finished until the requester receives Split Completion transactions for the entire byte count or a Split Completion Message indicating an error occurred as described in Section 2.10.6.2. A completer that executes Split Transactions for multiple Sequences concurrently is permitted to execute the transaction and initiate the Split Completions for different Sequences in any order. (Split Completions for the same Sequence must be initiated in address order.) As in conventional PCI, if a requester requires one non-posted transaction to complete before another, it must not initiate the second transaction until the first one completes.

Implementation Note: Mixing Immediate Response and Split Response

This note uses the following two terms to explain the limitations when Immediate Transactions and Split Transactions are used between a single pair of ADBs:

- Immediate-capable: an area of the device's address space that is capable of responding within the latency requirements defined in Section 2.9.1 such that the device executes read transactions as Immediate Transactions.
- Split-only: an area of the device's address space that cannot respond to read transactions within the latency requirements defined in Section 2.9.1 and thus the device must execute them as Split Transactions.

Device designers should use extreme care in mixing immediate-capable and split-only address spaces. Unless the address map is carefully laid out, the device must either provide data from the immediate-capable range in a Split Completion or must signal Target-Abort to some read transactions that would otherwise be legal.

Although requesters are generally required to understand the range limitations of the devices they address, completers have no ability to regulate the type and length of read commands that they are addressed by. Therefore the completer must respond to *any* read command within its address space. If the device is not designed to provide read data up to the next ADB as an Immediate Transaction, and the device is not designed to deliver all the data as a Split Transaction (as defined in Section 2.11.2.4), the device would have to signal Target-Abort to that read transaction and risk that the system will halt execution (see Section 2.11.2.5).

Case 1: When an immediate-capable address space precedes a split-only address space and both spaces are between a single pair of adjacent ADBs, the device is forced either to support providing the immediate-capable data within a Split Completion, or to signal Target-Abort for reads that cross those internal boundaries.

Example 1: A device with 128 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-3Fh for its immediate-capable register set and offsets 40h-7Fh as a window into some split-only memory. The device receives a memory read transaction for offset 00h with a byte count of 80h. Since the device cannot provide an immediate completion for offsets 40h-7Fh, the device cannot signal Data Transfer or Disconnect at Next ADB. Signaling either of these allows the device to disconnect the transaction no sooner than the next ADB, which is offset 80h. The device also cannot signal Single Data Phase Disconnect. Signaling Single Data Phase Disconnect for any read transaction that includes a split-only location is prohibited. (See Section 2.11.2.1.) The device is permitted to signal Single Data Phase Disconnect *only* if the read request is entirely contained within the offsets 00h-3Fh. If the read transaction includes any portion of the split-only range, the device must either signal Split Response and provide all 128 bytes of valid data in a single Split Completion or signal Target-Abort (and risk that the system will halt execution).

Example 2: A device with 256 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-7Fh for its immediate-capable register set and offsets 80h-FFh as a window into some split-only memory. The device receives a memory read transaction for offset 00h with a byte-count of 100h. Since the boundary between immediate-capable and split-only address spaces is located at an ADB (80h), the device is free to complete the transaction as an Immediate Transaction up to the ADB (signal Data Transfer, or Disconnect at Next ADB, or Single Data Phase Disconnect) and signal Split Response when the initiator continues the Sequence at the ADB.

Case 2: Whenever immediate-capable address space is located directly following split-only address space, the device is forced either to support providing the immediate-capable data within a Split Completion, or to signal Target-Abort to reads that cross those internal boundaries. This restriction applies even if the boundary is located at an ADB.

Example 3: A device with 128 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-3Fh as a window into some split-only memory and offsets 40h-7Fh for its immediate-capable register set. The device receives a memory read transaction for offset 00h with a byte count of 80h. Since the device is not permitted to provide a Split Completion with less than the requested byte count (see Section 2.10.2), it must either signal Split Response and provide all 128 bytes of valid data in a single Split Completion or signal Target-Abort (and risk that the system will halt execution).

Example 4: A device with 256 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-7Fh as a window into some split-only memory and offsets 80h-FFh for its immediate-capable register set. The device receives a memory read transaction for offset 00h with a byte-count of 100h. Although the device could conceivably signal Split Response and then disconnect its Split Completion at the ADB, the device is still required to satisfy the entire byte count (see Section 2.10.2). Therefore, as in Example 3, the device must either signal Split Response and provide all 256 bytes of valid data in a Split Completion or signal Target-Abort.

2.10.2. Split Completion

A Split Completion transaction is a transaction that uses the Split Completion command. As for all burst transactions, Split Completions include the byte count in the attribute phase. The **C/BE#** bus is reserved and driven high during all data phases of a Split Completion.

Split Completion transactions address their targets differently than other burst transactions. The target of a Split Completion is the requester that initiated the Split Request. The completer stores the Requester ID (bus number, device number, and function number of the requester) from the attribute phase of the Split Request. The Requester ID becomes part of the Split Completion address driven on the **AD** bus during the address phase of the Split Completion. (See Section 2.10.3 for a complete description of the Split Completion address.) PCI-X bridges use the Requester ID to determine which transactions to forward. The requester uses the Requester ID to recognize Split Completions that correspond to its Split Requests.

The attributes of a Split Completion differ from the attributes of other burst transactions in that they carry information about the completer rather than the requester. Completer Attributes are specified in Section 2.10.4.

If the Split Request was a burst read and the completer does not encounter an error condition, the Split Completion includes read data and has one or more data phases, up to that required to satisfy the byte count of the Split Request. (See Section 2.10.6 for error conditions and Split Completion Messages.) The completer and intervening bridges are permitted to disconnect the Split Completion transaction on any ADB, following the same protocol as other burst transactions. Each time the completer resumes the Split Completion after an initiator or target disconnection, the address and byte count must be adjusted to the portion remaining in the Sequence. The completer must initiate all Split Completions resulting from a single Split Request (i.e., with the same Sequence ID) in address order. An intervening bridge must maintain the order of Split Completion transactions with the same Sequence ID (that is, it must keep them in address order).

If the completer intends to disconnect the Split Completion on the first ADB (i.e., the next higher ADB from the starting address of the Split Request), the completer is permitted to use a byte count smaller than that of the Split Request (see Section 2.10.4 for the Byte Count Modified bit requirements). (Note that this is the only way the completer can disconnect the transaction on an ADB that is closer than four data phases from the starting address. See Section 2.11.1.1.) The completer must never use a byte count other than the full remaining byte count that would stop the transaction anywhere other than the first ADB of the Sequence. As with all Split Completions, the completer must keep the Split Completion data in address order, even when changing the byte count to disconnect on the first ADB.

The completer is further restricted from using a byte count less than the full remaining byte count of the Sequence if both of the following are true:

- The device is designed to complete as an Immediate Transaction a burst memory read transaction to an address greater than or equal to one particular ADB, ADB_n, and less than the next higher ADB, ADB_{n+1}.

- The starting address of the Sequence being completed as a Split Transaction is ADB_{n+1}.

This limitation exists because in some cases the burst memory read Sequence has crossed a PCI-X bridge and what appears as the starting address to the completer is actually a continuation by the bridge of a larger Sequence. (See Section 8.4.2.2.) Completers that modify the byte count only when the starting address is not equal to an ADB automatically meet this requirement.

If the Split Request was a DWORD transaction, the Lower Address field in the Split Completion address (see Section 2.10.3) is set to zero and the Byte Count field in the Completer Attributes (see Section 2.10.4) is set to four, regardless of which byte enables were asserted in the Split Request. If the Split Request was a read transaction, data is driven on **AD[31::00]** during the data phase of the Split Completion. Only byte lanes corresponding to the enabled bytes in the Split Request contain valid data. The requester must ignore the data in the other byte lanes (except for parity checking). If the Split Request was a write transaction, a Split Completion Message is driven on **AD[31::00]** regardless of the byte enables asserted in the Split Request. See Section 2.10.6 for a complete description of Split Completion Messages.

Implementation Note: Starting Addresses and Byte Count for Split Completions

Split Completion transactions are burst transactions even if the corresponding Split Request was a DWORD transaction. The way the completer generates the starting address and byte count for a Split Completion varies according to the characteristics of the Split Request and Split Completion.

When a completer generates a Split Completion for a burst Split Request, it normally copies the lower seven bits of the starting address and the byte count from the Split Request to the Split Completion. If the completer intends to disconnect the Split Completion on the first ADB, it is permitted to use a byte count other than that of the Split Request and must set the Byte Count Modified bit in the Completer Attributes.

In the following cases, the Split Completion is a single DWORD, and the completer sets the Lower Address field in the Split Completion address to zero, and sets the Byte Count field in the Completer Attributes to four (without setting the Byte Count Modified bit), regardless of the size of the Split Request:

- The Split Request was a DWORD transaction.
- The Split Completion contains a Split Completion Message.

Like all burst transactions, Split Completions are permitted to be initiated as 64- or 32-bit transfers. That is, **REQ64#** is permitted to be either asserted or deasserted. The completer (or an intervening bridge) is permitted to initiate the Split Completion at either transfer width, regardless of the width or type of the Split Request. The width of each transaction is negotiated independent of all previous transactions in the Sequence and independent of the Split Request. (See Section 2.12.3.) For example, a 64-bit PCI-X bridge is permitted to initiate the Split Completion with **REQ64#** asserted even if the requester initiated the Split Request with **REQ64#** deasserted. (In this case, the requester would likely not assert **ACK64#**, so the Split Completion would proceed as a 32-bit transaction.) Furthermore, the completer is permitted to initiate the Split Completion as a 64-bit transfer (**REQ64#** asserted) even if the Split Request was a DWORD transaction. In this case, the completer would drive the DWORD of read data or the Split Completion Message on **AD[31::00]**, since the starting address of the Split Completion (i.e., the Lower Address field of the Split Completion address) is set to 0 for completion of DWORD Split Requests. In other words, the transfer width and byte-lane requirements for a Split Completion are exactly the same as for a memory write of the identical length.

A completer is permitted to accept a single Split Request at a time. Such a device is permitted to terminate subsequent splittable transactions with Retry until the requester accepts the Split Completion. (Overall system performance is generally better if completers accept multiple Split Transactions at the same time.)

2.10.3. Split Completion Address

The Split Completion address is driven on the **AD** bus during the address phase of Split Completion transactions. The completer copies all this information from the address and attribute phases of the Split Request. Figure 2-38 shows the bit assignments for the Split Completion address. The Split Completion command is driven on **C/BE[3::0]#**.

Table 2-10 describes the bit definitions of the Split Completion address fields.

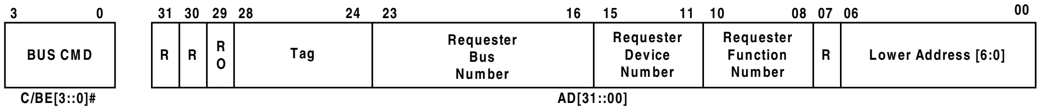


Figure 2-38: Split Completion Address

Table 2-10: Split Completion Address Field Definitions

Attribute	Function
Reserved (R)	Must be set to 0 by the initiator and ignored by the target (except for parity checking). PCI-X bridges forwarding a Split Completion must also set these bits to zero, even if they are set for the Split Completion received by the bridge. Future versions of the PCI-X definition may define these bits for additional features. PCI-X bridges designed to the present revision do not support such additional features and must set the bits to 0.
Relaxed Ordering (RO)	The completer copies this bit from the corresponding bit of the Requester Attributes (see Figure 2-1). Bridges throughout the system optionally use the bit to influence transaction ordering. A PCI-X bridge forwarding the Split Completion to another bus operating in PCI-X mode forwards this bit unmodified with the transaction, even if the bit is not used by the bridge.
Tag	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified.

Attribute	Function
Requester Bus Number	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. A PCI-X bridge uses this field to identify transactions to forward. If this field of a Split Completion on the secondary bus is *not* between the bridge's secondary bus number and subordinate bus number, inclusive, and the primary interface is operating in PCI-X mode, the bridge forwards the transaction upstream. If this field of a Split Completion on the primary bus *is* between the bridge's secondary bus number and subordinate bus number, inclusive, and the primary interface is operating in PCI mode, the bridge forwards the transaction downstream. If the bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified. See Section 8.4.3.1 for the use of this field by PCI-X bridges when one of the interfaces is operating in conventional mode.
Requester Device Number	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified.
Requester Function Number	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified.
Lower Address	The completer copies this field from the least significant seven bits of the address of the Split Request, regardless of the command used by the Split Request, if all of the following are true: • The Split Request for this Sequence was a burst read. • This is the first Split Completion of the Sequence. • The Split Completion is not a Split Completion Message. If the Split Completion is disconnected on an ADB, this field is zero when the Sequence resumes. If the Split Request was a DWORD transaction or the Split Completion is a Split Completion Message, this field is set to zero. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it uses this information to determine where the Split Completion starts relative to an ADB. The bridge leaves this field unmodified.

2.10.4. Completer Attributes

The attribute phase of a Split Completion contains the Completer Attributes. The Completer Attributes are a combination of the Completer ID and information about the Sequence stored from the Split Request. Figure 2-39 shows the bit assignments for the Completer Attributes, and Table 2-11 describes the bit definitions.

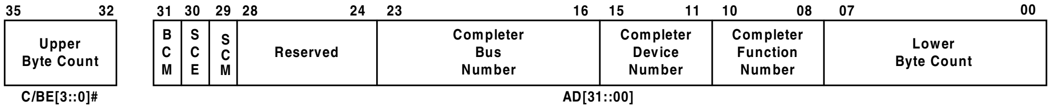


Figure 2-39: Completer Attribute Bit Assignments

Table 2-11: Completer Attribute Field Definitions

Attribute	Function
Byte Count Modified (BCM)	The completer must set this bit to 1 if the Byte Count field for this Split Completion contains a number smaller than the full remaining byte count of the transaction. (The completer is allowed to modify the byte count only to disconnect the transaction on the first ADB of the Sequence. See Section 2.10.2 for additional restrictions.) If the byte count field contains the full remaining byte count of the Split Request, or if the Split Completion is a Split Completion Message, the completer sets this bit to 0. This bit is used only for Split Completions resulting from burst read transactions (Memory Read Block and Alias to Memory Read Block) and is set to 0 for Split Completions resulting from all other commands. This bit is used for diagnostic purposes. Targets (bridges and requesters) are permitted to ignore this bit.
Split Completion Error (SCE)	The completer sets this bit if the transaction is a Split Completion Message that is an error message (i.e., Message Class 1h or 2h). Requesters are permitted to use this information to differentiate between normal and error write completion messages before the actual message is latched and decoded. See the Split Completion Message attribute bit described below for additional requirements.
Split Completion Message (SCM)	The completer sets this bit to 0 if the Split Completion contains read data. It sets this bit to 1 if the Split Completion contains a Split Completion Message. See Section 2.10.6 for a complete discussion of Split Completion Messages. The Split Completion Error and Split Completion Message bits are allowed together as follows: SCE SCM Case 0 0 Normal completion of read (includes read data) 0 1 Normal completion of write (includes message) 1 0 reserved 1 1 Error completion (read or write, includes message)

Attribute	Function
Reserved (R)	Must be set to 0 by the completer and ignored by the requester (except for parity checking). PCI-X bridges forward these bits unmodified.
Completer Bus Number	This 8-bit field identifies the completer's bus number. Completers supply this number from the Bus Number register in the PCI-X Status register. The value FFh is reserved and means the completer's PCI-X Status register has not been initialized. This information is used for diagnostic purposes on the bus. The combination of the Completer Bus Number, Completer Device Number, and Completer Function Number is referred to as the Completer ID.
Completer Device Number	This 5-bit field contains the device number assigned to the completer. Completers supply this number from the Device Number register in the PCI-X Status register. The value 1Fh is reserved and means the completer's PCI-X Status register has not been initialized. The Device Number of the source bridge is always 00h. The combination of the Completer Bus Number, Completer Device Number, and Completer Function Number is referred to as the Completer ID.
Completer Function Number	This 3-bit field contains the function number of the completer within the device. This is the function number in the configuration address to which the function responds. Unlike the Device Number and Bus Number fields in the PCI-X Status register, the value of the Function Number field is assigned to the function by design and needs no initialization. The combination of the Completer Bus Number, Completer Device Number, and Completer Function Number is referred to as the Completer ID.

Attribute	Function
Upper Byte Count, Lower Byte Count	This 12-bit field is divided between the Upper Byte Count in the **C/BE[3::0]#** bus and the Lower Byte Count in the **AD[7::0]** bus. The target (requester or bridge) uses this information to determine the end of the transaction, particularly if the Split Completion has less than four data phases. In some cases, the completer copies this field from the corresponding field in the Requester Attributes. In other cases, the completer generates the value for this field. (See Section 2.10.2 for details). There is no guarantee that the initiator will successfully move the entire byte count in a single transaction. If the Split Completion transaction is disconnected for any reason, the initiator must adjust the contents of the Byte Count field in the subsequent transactions of the same Sequence to be the number of bytes remaining in this Sequence. If the Split Completion is a Split Completion Message, the completer sets the byte count to four (see Section 2.10.6). The Byte Count is specified as a binary number, with 0000 0000 0001b indicating 1 byte, 1111 1111 1111b indicating 4095 bytes, and 0000 0000 0000b indicating 4096 bytes.

Implementation Note: Use of the Byte Count Modified Attribute

The purpose of the Byte Count Modified bit in the Completer Attributes is to provide visibility for devices that monitor but do not participate in the bus protocol (such as a bus analyzer). PCI-X devices and bridges are not required to decode this bit.

For example, suppose a 64-bit requester initiates a Memory Read Block transaction for 280 bytes starting at address 104 (three data phases from the ADB at address 128). The completer (on the same bus) signals Split Response and fetches the data. Further suppose that the completer wants to disconnect the Split Completion transaction at the first ADB (address 128) and, therefore, changes the byte count in the Completer Attributes to 24 bytes and sets the Byte Count Modified bit to 1. A logic analyzer monitoring the bus observes the Byte Count Modified bit set and realizes that the Sequence is not complete, even though the byte count of 24 is satisfied. After the first split completion, the completer regains bus ownership, issues a new Split Completion with a byte count of 256 (the remaining byte count), and sets the Byte Count Modified bit to 0.

2.10.5. Requirements for Accepting Split Completions

The requester is required to accept all Split Completions resulting from its own Split Requests. That is, the requester is required to assert **DEVSEL#** on all Split Completions in which the Sequence ID (Requester ID and Tag) corresponds to a Split Request issued by that device. See Section 5.4.5 for Split Completions that are unexpected or corrupted.

If the requester asserts **DEVSEL#** for a Split Completion, the requester must accept the entire byte count requested without signaling Split Response, Retry, Single Data Phase Disconnect, or Disconnect at Next ADB. If the requester no longer needs the Split Completion data, the requester must accept it and then discard it. In general, a requester must have a buffer ready to receive the entire byte count for all Split Requests it issues.

The requester is permitted to signal Target-Abort for a Split Completion only under error conditions in which the integrity of data in the system cannot be guaranteed. An example of such an error condition is a parity error in the Split Completion address, which includes the Sequence ID. (See Section 5.4.3 for requirements for the target also to assert **SERR#** in this case.) In some cases, signaling Target-Abort for a Split Completion causes another device to assert **SERR#**. (See Sections 5.4.4 and 8.7.1.6.) The requester must assume that a possible consequence of signaling Target-Abort for a Split Completion transaction is that the system will halt execution.

Bridges (PCI-X bridges and application bridges) are permitted to terminate Split Completions with Retry and to disconnect multi-data-phase Split Completions in some cases. See Section 8.4.5 for more details.

If a requester issues more than one Split Request at a time (with different Tags), the requester must accept the Split Completions from the separate requests in any order. (Split Completions with the *same* Tag originate from the same Split Request and always arrive in address order.)

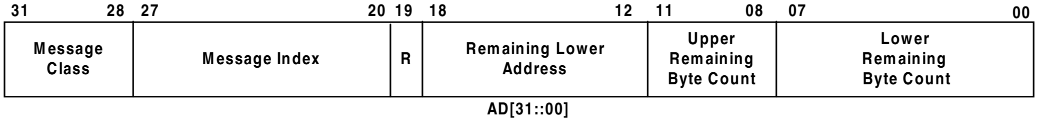
2.10.6. Split Completion Messages

If the SCM bit in the Completer Attributes is set, the transaction includes a message. Split Completion Messages notify the requester when a split write request (I/O or configuration) has completed, and they indicate error conditions in which delivery of data for a read request or execution of a write request is not possible.

A Split Completion Message is a burst transaction (like all Split Completions) but is always a single DWORD in length regardless of the size and type of the Split Request. The Lower Address field in the Split Completion address is set to zero, and the Byte Count field in the Completer Attributes is set to four for all Split Completion Messages. The **C/BE#** bus is reserved and driven high during the data phase of a Split Completion Message as it is for all Split Completions.

A Split Completion Message terminates a Sequence regardless of how many bytes remain to be sent. If the Split Request was a burst read, the Byte Count field in the Split Completion Message indicates the number of bytes that were not sent for this Sequence, and the Remaining Lower Address field indicates the lower seven bits of the starting address of the remainder of the Sequence. (PCI-X bridges use this information to release buffer space that was reserved by the Split Request.)

Figure 2-40 shows the format of the message in the data phase of the Split Completion, and Table 2-12 shows the encoding of those messages.



31 28	27 20	19	18 12	11 08	07 00
Message Class	Message Index	R	Remaining Lower Address	Upper Remaining Byte Count	Lower Remaining Byte Count

AD[31::00]

Figure 2-40: Split Completion Message Format

Table 2-12: Split Completion Message Fields

Field	Function
Message Class	Split Completion Messages are in one of the following classes. All other values are reserved. 0h Write Completion (See Section 2.10.6.1.) 1h PCI-X Bridge Error (See Section 8.8.) 2h Completer Error (See Section 2.10.6.2.)
Message Index	Identifies the type of message within the message class. See Table 2-13 and Table 2-14.
Reserved (R)	Must be set to 0 by the completer (or intervening bridge) and ignored by the requester (or intervening bridge).
Remaining Lower Address	If the Split Request was a burst memory read, this field contains the least significant seven bits of the address of the first byte of read data that has not previously been sent. If the Split Request was a DWORD transaction, the completer sets this field to zero. PCI-X bridges use this number to manage buffer space reserved for Split Completions.
Upper and Lower Remaining Byte Count	If the Split Request was a burst memory read, the completer sets this field to the number of bytes of read data that have not previously been sent. If the Split Request was a DWORD transaction, the completer sets this field to 4. PCI-X bridges use this number to manage buffer space reserved for Split Completions.

2.10.6.1. Write Completion Message Class

The Write Completion class is used for Split Write Completion messages. The Remaining Lower Address field is set to zero and the Upper and Lower Remaining Byte Count field set to four in the data phase of this Split Completion Message (PCI-X bridges reserve a single DWORD for a Split Write Completion). (The Lower Address in the Split Completion address is also zero and the byte count in the Completer Attributes is also four, as it is for all Split Completion Messages.)

Only one message index is defined in this class, as shown in Table 2-13. All other indices are reserved.

Table 2-13: Write Completion Message Index (Class 0)

Index	Message
00h	Normal completion

2.10.6.2. Completer Error Message Class

After signaling Split Response, if the completer encounters an abnormal condition that prevents it from executing a Split Transaction, the completer must notify the requester of the abnormal condition by sending a Split Completion Message with the Completer Error class. Examples of such conditions include the following:

1. The byte count of the request exceeds the range of the completer.
2. Parity errors internal to the completer.

If the byte count exceeds the range of the completer, the completer must initiate Split Completion transactions with read data up to the device boundary and then disconnect the Sequence. The completer then terminates the Sequence by sending the Split Completion Message. In all other cases, the completer is permitted to send a Split Completion

Message of this class in lieu of the first Split Completion, or any continuation in a Sequence (after a disconnection), independent of the actual address of the error in the Sequence. The only inference that can be made as to the actual address of the error is as follows:

1. If the Sequence was previously disconnected on an ADB, the address of the error is greater than the last address of the Split Completion transactions that were previously sent without error for this Sequence.
2. The error address is less than or equal to the ending address of the Sequenœ.

Table 2-14 shows the index values defined for this message class. All other indices are reserved.

Table 2-14: Completer Error Messages Indices (Class 2)

Index	Message
00h	**Byte Count Out of Range.** The completer uses this message if the sum of the address and the byte count of the Split Request exceeds the address range of the completer. The completer must initiate Split Completion transactions with read data up to the device boundary. A normally functioning requester understands the address range of the completer it is attempting to read and does not request data that is out of range. The completer sends this message to indicate to the requester the occurrence of an error condition. (The error could have occurred either in the completer or in the requester). The requester must report this error condition to its device driver.
01h	**Split Write Data Parity Error.** The completer sends this message if it terminated a DWORD write transaction with Split Response and detected a data parity error. (See Section 5.4.4.)
8Xh	**Device-Specific Error.** The completer uses this message if it encounters an error that prevents execution of the Split Request, and the error is not indicated by one of the other error messages. The lower four bits of the index are available for the device to encode device-specific error or diagnostic information. The vendor of the device must provide documentation for this field.

Implementation Note: Reporting Device-Specific Error Messages

One way to report the receipt of a device-specific error Split Completion Message is for the requester to store the lower four bits of the message index in a device-specific location and cause an interrupt to the processor. The device driver servicing the interrupt would read the register.

A common practice in cases such as these is to reserve one error encoding (e.g., 0h) to indicate no error condition. In this case, software would clear the register after the occurrence of each error, so it could tell the difference between new errors and errors it had already recorded.

Alternatively, all 16 codes could be assigned to different errors and an additional device-specific bit assigned to indicate that the register contains a new error condition.

2.11. Transaction Termination

The figures in this section show the methods by which transactions are terminated. Those methods include the following:

- Initiator Termination
  - Initiator Disconnection or Satisfaction of Byte Count
  - Master-Abort Termination
- Target Termination
  - Single Data Phase Disconnection
  - Disconnection at Next ADB
  - Retry Termination
  - Split Response Termination
  - Target-Abort Termination

Most of the figures in this section that illustrate transaction termination show **DEVSEL#** decode A and no target initial wait states. All other **DEVSEL#** decodes and wait state combinations specified in Sections 2.8 and 2.9.1 are also allowed.

When a transaction ends, the initiator deasserts and floats **FRAME#** as described in PCI 2.2 for sustained tri-state signals, that is, the initiator actively deasserts **FRAME#** for one clock and then floats it. For those PCI-X transactions that contain one or two data phases (i.e., when **FRAME#** deasserts at the same time **IRDY#** deasserts), this timing is required to avoid conflicts with the next bus owner (e.g., see Figure 2-43 and Figure 2-44). However, for those transactions that contain three, four, or more data phases (i.e., when **FRAME#** deasserts before **IRDY#** deasserts) the initiator optionally deasserts **FRAME#** for one clock and then floats it (as in the one- and two- data phase cases) or deasserts **FRAME#** for two clocks and then floats it. Most of the figures in this section that show three, four, or more data phases, show **FRAME#** deasserted for two clocks before floating. Figure 2-42 illustrates the other alternative.

Most of the figures in this section that illustrate transaction termination apply both to read and write transactions, and, therefore, do not show the **AD** bus. In all cases the initiator and target begin driving the **AD** and **C/BE#** buses as described in Sections 2.8 and 2.9 for the appropriate **DEVSEL#** timing and number of wait states. The initiator and target float the **AD** and **C/BE#** buses according to the following rules:

1. Initiator:
   a. If the transaction has four or more data phases, the initiator floats the **C/BE#** bus on the clock it deasserts **IRDY#**. If the transaction has less than four data phases, the initiator floats the **C/BE#** bus either on the clock it deasserts **IRDY#** or one clock after that.
   b. If the transaction is a write with four or more data phases, the initiator floats the **AD** bus on the clock it deasserts **IRDY#**. If the transaction is a write with less than four data phases, the initiator floats the **AD** bus either on the clock it deasserts **IRDY#** or one clock after that.
2. Target: If the transaction is a read, the target floats the **AD** bus on the clock after the last data phase, regardless of the number of data phases in the transaction or the type of termination. That is, the target floats the **AD** bus on the clock it deasserts **DEVSEL#**, **STOP#**, and/or **TRDY#** after signaling the last Data Transfer or target termination.

2.11.1. Initiator Termination

2.11.1.1. Initiator Disconnection or Satisfaction of Byte Count

Transactions that are disconnected by the initiator on an ADB before the byte count has been satisfied and those that terminate at the end of the byte count appear the same on the bus. Initiator termination of a transaction with four or more data phases differs from the case in which the transaction has less than four data phases.

Figure 2-41 illustrates initiator termination after four or more data phases. In this case, the initiator signals the end of the transaction by deasserting **FRAME#** one clock before the last data phase. It deasserts **IRDY#** on the clock after the last data phase.

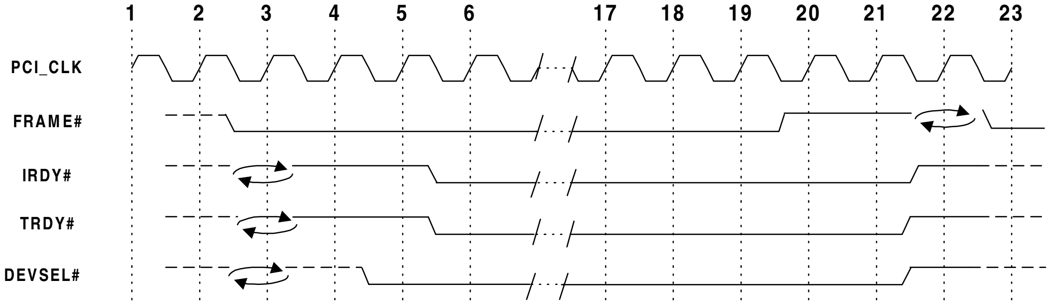


Figure 2-41: Initiator Termination of a Burst Transaction with Four or More Data Phases

Initiator termination in less than four data phases occurs only if the starting address, byte count, and width of the bus are such that the transaction has less than four data phases. If the initiator intends to disconnect a transaction on the first ADB, and the width of the bus is such that the starting address is less than four data phases from the ADB, the initiator must adjust the byte count to terminate the transaction on the ADB.

Table 2-15 shows the number of data phases for transactions up to 12 DWORDs long. (For this table, the length of the transaction is measured from the starting address rounded down to the next DWORD address.) As the table shows, the number of data phases is equal to the number of DWORDs if the transaction width is 32 bits. If the transaction width is 64 bits, the number of data phases depends on whether the transaction starts on an even or odd DWORD.

Table 2-15: Data Phases Dependence on Starting Address and Bus Width

	Data Phases		
		64-bit Transfer	
Transaction Length (DWORDs)	32-bit Transfers	Starting on Even DWORD	Starting on Odd DWORD
1	1	1	1
2	2	1	2
3	3	2	2
4	4	2	3
5	5	3	3
6	6	3	4
7	7	4	4
8	8	4	5
9	9	5	5
10	10	5	6
11	11	6	6
12	12	6	7

Figure 2-42 through Figure 2-44 illustrate initiator termination after three, two, and one data phases, respectively. In each of these cases, the initiator deasserts **FRAME#** two clocks after the target asserts **TRDY#**. The initiator deasserts **IRDY#** one clock after the *last* data phase but never less than two clocks after the *first* data phase (the clock in which **FRAME#** is deasserted). The target deasserts **TRDY#** and **DEVSEL#** on the first clock after the byte count provided by the initiator is satisfied.

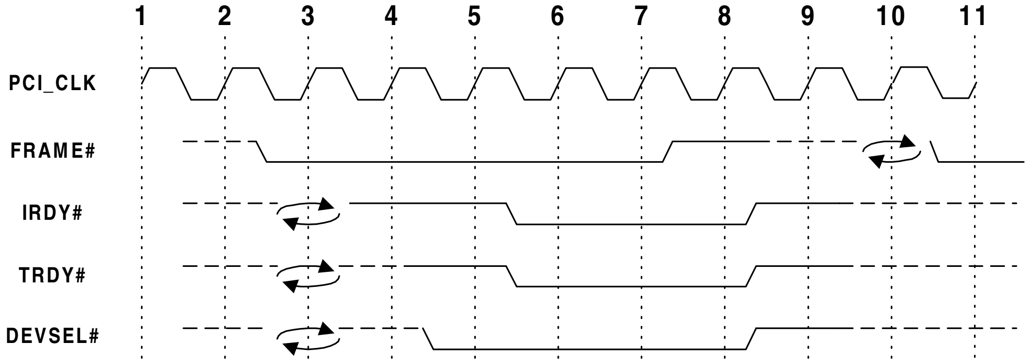


Figure 2-42: Initiator Termination of a Burst Transaction with Three Data Phases

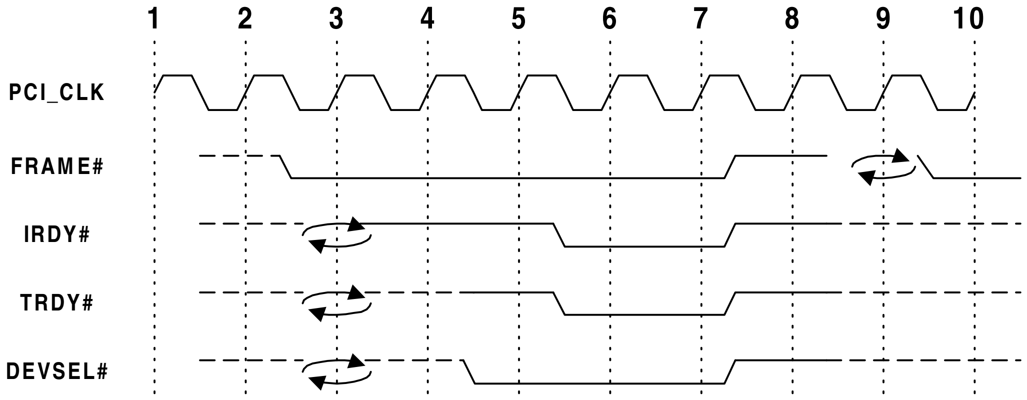


Figure 2-43: Initiator Termination of a Burst Transaction with Two Data Phases

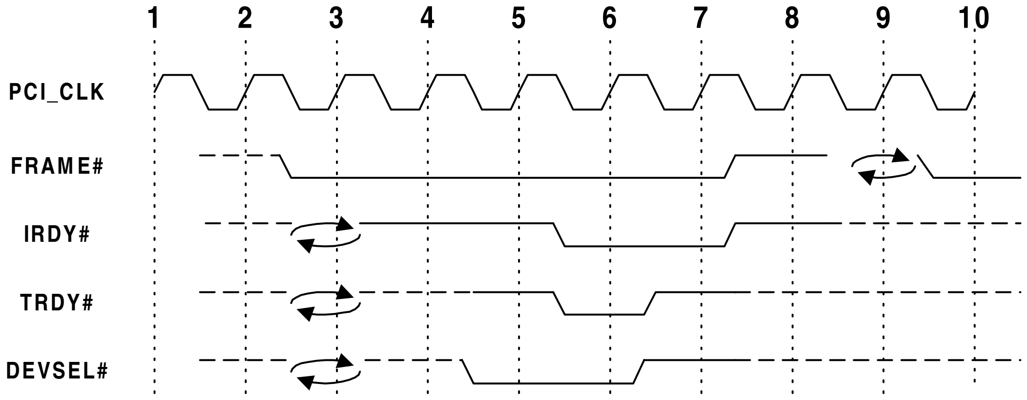


Figure 2-44: Initiator Termination of a Burst Transaction with One Data Phase

2.11.1.2. Master-Abort Termination

If no target asserts **DEVSEL#** within six clocks after the address phase (the second address phase for dual address cycles), the initiator deasserts **FRAME#** and **IRDY#** eight clocks after the address phase(s) and floats the bus one clock later. The initiator sets bits in its Status register the same as for conventional PCI.

Figure 2-45 shows a Master-Abort termination of a transaction.

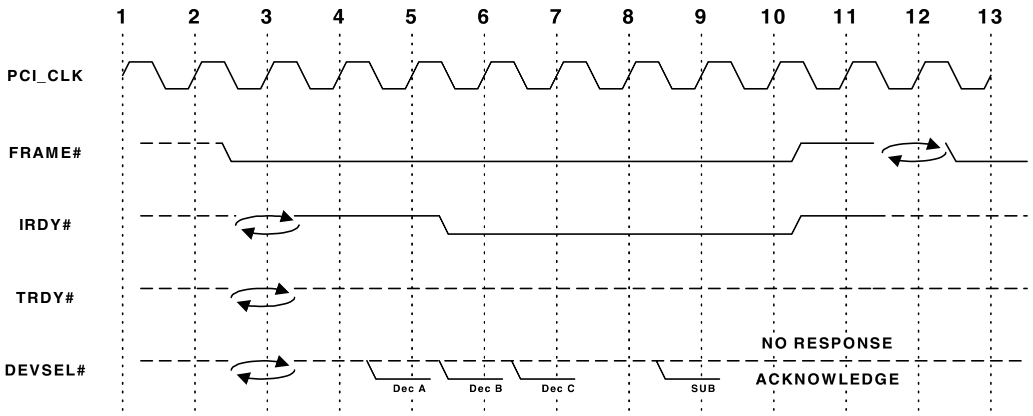


Figure 2-45: Master-Abort Termination

2.11.2. Target Termination and Data Phase Signaling

After a target asserts **DEVSEL#** in the target response phase, it must complete the transaction with one or more data phases. The target signals its intention on each clock after the target response phase with a combination of the target control signals, **DEVSEL#**, **STOP#**, and **TRDY#**. Table 2-16 shows all of the alternatives for target data phase signaling.

Table 2-16: Target Data Phase Signaling

Target Data Phase Signaling	DEVSEL#	STOP#	TRDY#	Target Initial Latency Section 2.9.1	Data Transfer	Transaction Terminates or Continues
Master-Abort (Note 1)	Deassert	Deassert	Deassert	na	na	na
Split Response	Deassert	Deassert	Assert	8	(Note 2)	Terminates
Target-Abort	Deassert	Assert	Deassert	8	No	Terminates
Single Data Phase Disconnect	Deassert	Assert	Assert	16	Yes	Terminates
Wait State	Assert	Deassert	Deassert	na	No	Continues
Data Transfer	Assert	Deassert	Assert	16	Yes	Continues
Retry	Assert	Assert	Deassert	8	No	Terminates
Disconnect at Next ADB	Assert	Assert	Assert	16	Yes	(Note 3)

Notes:

1. Shown for reference only. Not allowed after **DEVSEL#** is asserted. No target drives **DEVSEL#**, **STOP#**, and **TRDY#** for a transaction terminated with Master-Abort. The signals are deasserted by their respective pull-up resistors.
2. No data transfers on a Split Response for a read transaction. The target latches data on a Split Response for a write transaction. However, in both cases, the transaction is not complete until the requester receives the Split Completion.
3. If the target signals Disconnect at Next ADB, the transaction continues to an ADB. (See Section 2.11.2.2 for details.)

A data phase ends each time the target signals anything other than Wait State (which is permitted only on the first data phase). See Section 2.9.1 for a discussion of the number of wait states permitted and the target initial latency.

If the target signals Data Transfer on one data phase, the transaction continues until the byte count is satisfied or the initiator terminates the transaction. The target is limited to signaling Data Transfer, Disconnect on Next ADB, or Target-Abort on subsequent data phases.

If the target signals Split Response, Target-Abort, Single Data Phase Disconnect, or Retry, the transaction terminates immediately. The transaction terminates on an ADB if the target signals Disconnect at Next ADB (see Section 2.11.2.2 for details).

When the transaction terminates (either by initiator or target termination), the target deasserts **DEVSEL#**, **STOP#**, and **TRDY#** one clock after the last data phase (if they are not already deasserted) and floats them one clock after that.

Targets must not store any information about a transaction after it is terminated either by the initiator or the target in any method other than Split Response. (Storing of transaction information for diagnostic purposes is permitted, if such information does not affect the device's response to transactions on the bus. Target-Abort termination and some error conditions require the target to set bits in the Status register.) Delayed Transactions are not permitted. For example, if a target collects data up to the byte count of a read transaction, delivers some of that data by signaling Data Transfer (i.e., executes it as an Immediate Transaction), and the transaction is disconnected (either by the target or the initiator), the target must discard the remainder of the data, unless the target guarantees that the buffered data will not become stale. The target must not assume that the initiator will continue any read operation after a transaction is terminated by the initiator or the target.

2.11.2.1. Single Data Phase Disconnection

The target signals its intention to complete a single data phase and then disconnect the transaction by signaling Single Data Phase Disconnect. The target signals Single Data Phase Disconnect by asserting **TRDY#** and **STOP#** and deasserting **DEVSEL#** on the first data phase of the transaction (with or without preceding wait states up to the maximum specified in Section 2.9.1). It is permitted both on burst transactions (even if the byte count is small enough to limit the transaction to a single data phase) and DWORD transactions (which are always a single data phase). If the target signals Single Data Phase Disconnect, the transaction contains only a single data phase, and the initiator deasserts **FRAME#** and **IRDY#** two clocks after the data phase.

Targets must be designed never to signal both Single Data Phase Disconnect and Data Transfer for memory write transactions that begin four or less data phases before any single ADB, unless the target verifies that the byte count is small enough not to include that ADB. That is, if a target is designed to signal Single Data Phase Disconnect for a memory write transaction with an address four or less data phases before an ADB, that target must be designed never to signal Data Transfer for a memory write transaction that begins four or less data phases from that same ADB and has a bye count large enough to include the ADB.

Targets are permitted to signal Single Data Phase Disconnect for a memory read transaction only if they are prepared to complete all transactions that are continuations of that Sequence, up to the next ADB, as Immediate Transactions. That is, the device must *not* signal Single Data Phase Disconnect for any individual read transaction of a Sequence if all of the following are true:

- The transaction is a burst read.
- The byte count is such that the transaction addresses at least one location to which the device will respond with Split Response when the Sequence is continued by the initiator.
- There is *not* an ADB between the first address of the transaction and the location to which the device will respond with Split Response when the Sequence is continued.

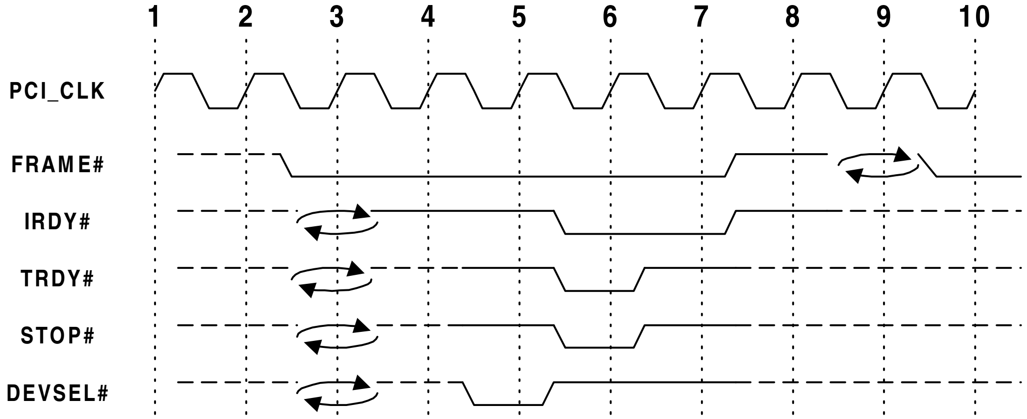


Figure 2-46: Single Data Phase Disconnection

Implementation Note: Use of Single Data Phase Disconnection

Single data phase disconnection is intended for address spaces such as control registers that generally are not accessed using burst transactions. Although it is permitted for both read and write transactions, its most common application is for writes. In some cases, memory write transactions that are intended to be separate transactions are combined into a single transaction by a host bridge or a conventional PCI bridge. The target avoids having to accept a burst up to the next ADB by signaling Single Data Phase Disconnect on each data phase. If the target signals Single Data Phase Disconnect for a location that is frequently addressed with multiple-data-phase burst transactions, the device's performance is severely reduced.

Implementation Note: Single Data Phase Disconnection and Memory Write Transactions

If a target signals Single Data Phase Disconnect for a memory write transaction that starts close to an ADB and signals Data Transfer for the continuation of that memory write, a PCI-X bridge will be unable to forward the memory write transaction in the following case:

1. A requester initiates a long memory write transaction (e.g., a host bridge combines many small memory write transactions) addressing a completer on the other side of a PCI-X bridge.
2. The bridge does not have buffer space available to hold the entire byte count of the memory write. However, it does have space for several ADQs of memory write data, so it responds to the transaction with Data Transfer and begins accepting data.
3. As the last bridge buffer fills, the bridge signals Disconnect at Next ADB, and the requester disconnects the transaction at the next ADB.
4. The bridge forwards this first memory write transaction of the Sequence to the destination bus.
5. The completer responds to the memory write transaction with Single Data Phase Disconnect and continues to do so for each continuation of the Sequence until the bridge holds less than four data phases of data in its buffers.

If the completer were to respond to the next continuation of the memory write Sequence with Data Transfer, the bridge would not be able to disconnect the transaction at the next ADB, because the continuation began less than four data phases from an ADB. The bridge could not continue beyond the ADB, because the requester has not yet written that data on the originating bus.

Implementation Note: Single Data Phase Disconnection and Burst Memory Read Transactions

The use of Single Data Phase Disconnect for burst memory read transactions is allowed, but rarely occurs in actual applications. In normal use, a device that is designed to respond with Single Data Phase Disconnect is never addressed by burst memory read transactions. Therefore, completers are limited in the way they respond to burst read transactions if they mix Single Data Phase Disconnect and Split Response between a single pair of adjacent ADBs.

For example, a device with 256 bytes of memory space assigned through a Base Address register is designed to respond with Split Response if address offset A0h is read. If the device is addressed by a read transaction starting at offset 00h with a length of 256 bytes, the device would be permitted to signal Single Data Phase Disconnect. In this case there is an ADB (offset 80h) between the first address of the read transaction and the Split Response address. However, as the initiator continues reading from the disconnection point, the starting address eventually advances to the ADB (offset 80h) with a length of 128 bytes. In this case the device would *not* be permitted to signal Single Data Phase Disconnect because there is no ADB between the starting address and the address to which the device will respond with Split Response (A0h).

If the same device is addressed by a different Sequence starting at address 80h with a byte count of 32 bytes (that is, an ending address of 9Fh), the device is permitted to signal Single Data Phase Disconnect because the Sequence does not include any locations to which the device will respond with Split Response when the initiator resumes after the disconnection.

See Section 2.10.1 for additional design considerations when locations of this type are mixed between the same two ADBs.

This restriction on the use of Single Data Phase Disconnect and Split Response simplifies the design of PCI-X bridges forwarding a burst read transaction. Without this restriction a PCI-X bridge forwarding a burst read request would have to deal with the possibility that the completer could signal Single Data Phase Disconnect at the beginning of the transaction and then change to Split Response midway between two ADBs. A PCI-X bridge would not generally be able to create a Split Completion for the transactions if it only held a portion of the data that ended midway between two ADBs. The completer is permitted to change from an immediate completion to a Split Response only at an ADB.

2.11.2.2. Disconnection at Next ADB

The target signals its intention to disconnect the transaction at the next ADB by signaling Disconnect at Next ADB. The target signals Disconnect at Next ADB by asserting **TRDY#**, **DEVSEL#**, and **STOP#** on any data phase of the transaction. The target is permitted to signal Disconnect at Next ADB regardless of the starting address or length of the transaction, or whether the transaction is a burst or DWORD. Some restrictions apply to the use of Disconnect at Next ADB by bridges (see Section 8.4.6). Once the target has signaled Disconnect at Next ADB, it is limited to signaling Disconnect at Next ADB or Target-Abort on all subsequent data phases until the end of the transaction. (The transaction ends immediately after the target signals Target-Abort. See Section 2.11.2.5.)

If the length of a transaction is such that it does not cross the next ADB (i.e., if it is a DWORD transaction or the byte count of a burst is satisfied before reaching the next ADB), Disconnect at Next ADB is treated by the initiator the same as Data Transfer. If the transaction is a burst that would otherwise cross the next ADB and the target signals

Disconnect at Next ADB on the first data phase of the transaction, the transaction ends at the first ADB. If the target signals Disconnect at Next ADB after the first data phase and four or more data phases before an ADB, the initiator disconnects the transaction on that ADB. If the target signals Disconnect at Next ADB after the first data phase and less than four data phases before an ADB, the transaction crosses that ADB and continues to the next ADB (unless the byte count is satisfied before that).

The following figures illustrate Disconnect at Next ADB. Figure 2-47 illustrates Disconnect at Next ADB after the first data phase and four data phases from an ADB. The target signals Disconnect at Next ADB by asserting **STOP#** while **TRDY#** and **DEVSEL#** are asserted four data phases before the ADB.

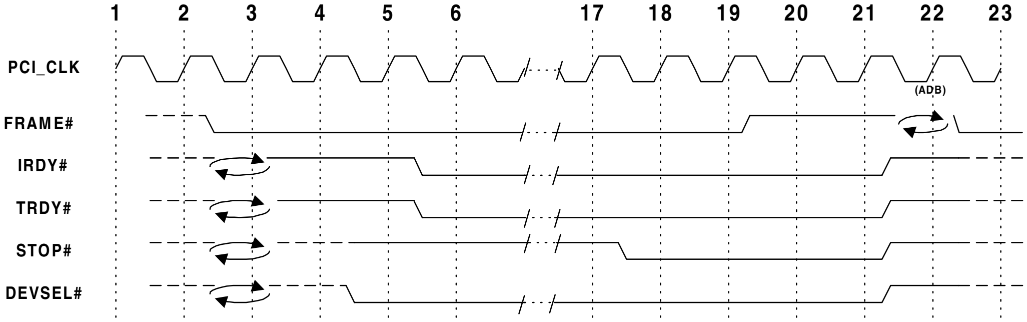


Figure 2-47: Disconnect at Next ADB Four Data Phases from an ADB

Figure 2-48 illustrates the case in which the target signals Disconnect at Next ADB on various data phases relative to an ADB. If the target signals Disconnect at Next ADB after the first data phase and less than four data phases from an ADB (clocks 7, 8, or 9 in the figure), the transaction crosses that ADB and disconnects on the next one. If the target signals Disconnect at Next ADB four or more data phases before an ADB (clocks 11 through 22 in the figure), the transaction disconnects on the ADB.

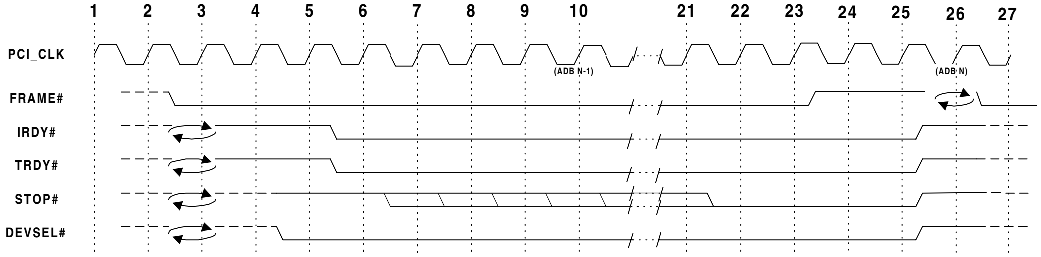


Figure 2-48: Disconnect at Next ADB on ADB N

Figure 2-49 through Figure 2-51 illustrate the target signaling Disconnect at Next ADB for transactions whose starting address is three, two, and one data phases from the ADB, respectively. In these figures, the target signals Disconnect at Next ADB on the first data phase. The initiator responds by deasserting **FRAME#** two clocks after the *first* data phase. The initiator deasserts **IRDY#** one clock after the *last* data phase but never less than two clocks after the *first* data phase (the clock in which **FRAME#** is deasserted).

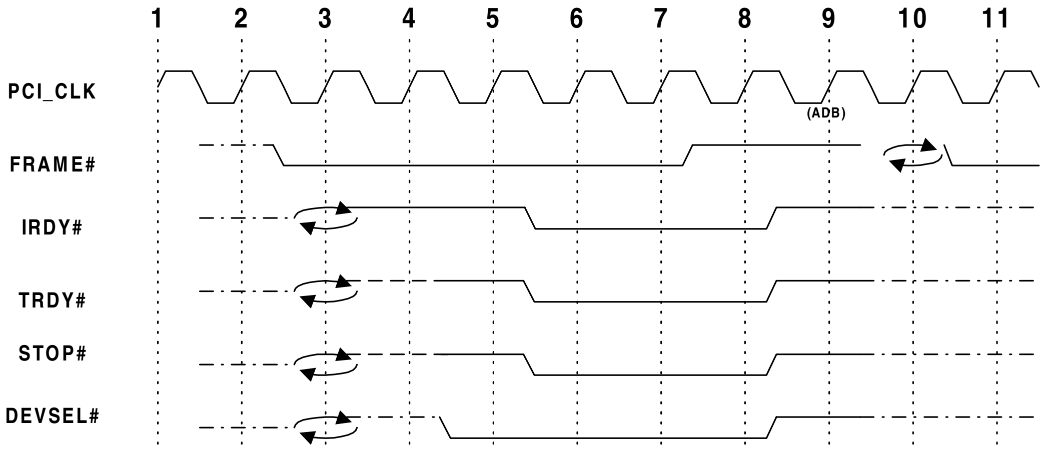


Figure 2-49: Disconnect at Next ADB with Starting Address Three Data Phases from an ADB

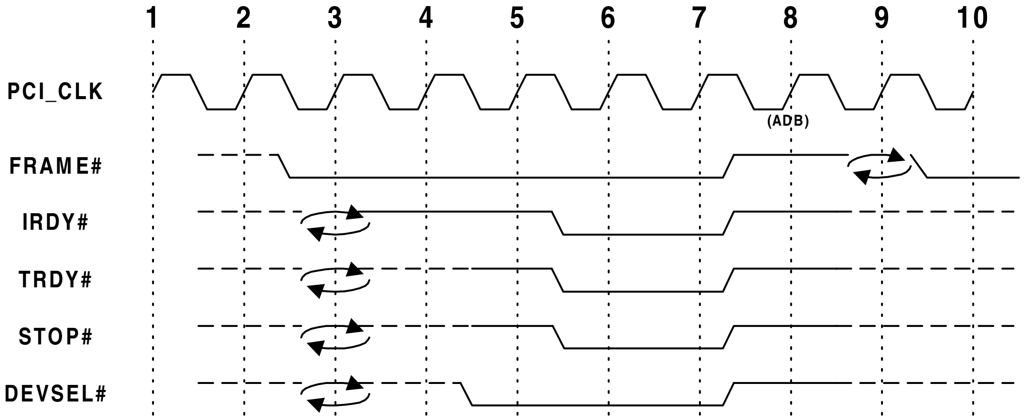


Figure 2-50: Disconnect at Next ADB with Starting Address Two Data Phases from an ADB

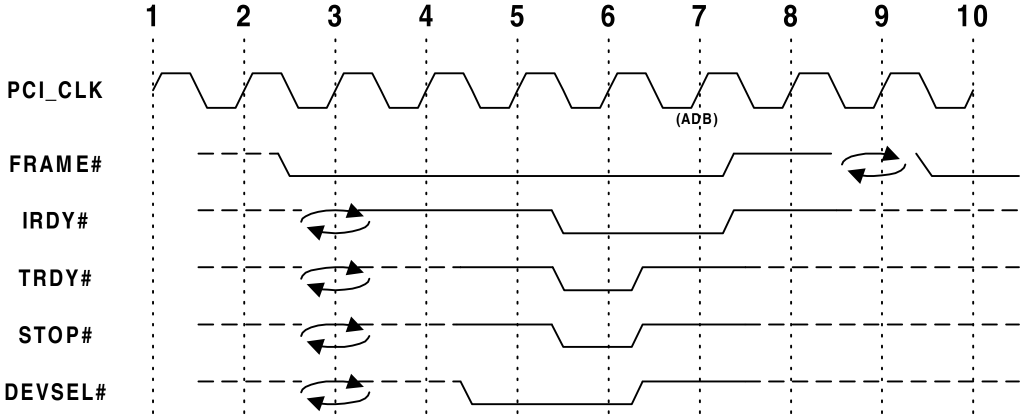


Figure 2-51: Disconnect at Next ADB with Starting Address One Data Phase from an ADB

2.11.2.3. Retry Termination

The target indicates that it is temporarily unable to complete the transaction by signaling Retry. The target signals Retry by asserting **STOP#** and **DEVSEL#** and keeping **TRDY#** deasserted on the first data phase of the transaction (with or without preceding wait states up to the maximum specified in Section 2.9.1). The target is permitted to terminate the transaction with Retry only under the following conditions:

- The device initialization time after the rising edge of **RST#** (T_{rhfa} specified in Table 9-5) has not elapsed.
- The device normally transfers data within the target initial latency limit listed in Table 2-9, but under some conditions that are guaranteed to resolve quickly, execution of the transaction would take longer. See Section 2.9.1 for additional limitations.
- The transaction is a memory write and all of the buffers for accepting memory write transactions are currently full with previous memory write transactions. See Section 2.13 for additional limitations.
- The transaction is not a memory write, it would require longer than the target initial latency to execute, and the target's Split Request queue is full.
- The transaction is a Split Completion, the target is a bridge as defined in Section 8.2, and the buffers for accepting Split Completions are currently full. See Section 8.4.5 for more details.

Unlike conventional PCI, a PCI-X target must not assume the initiator will repeat a transaction terminated with Retry. For transactions other than memory writes, the target must discard all state information related to a transaction for which it signals Retry. Delayed Transactions as defined in PCI 2.2 are not allowed. For memory write transactions, the target must not change its internal state in any way if it signals Retry on the first data phase of the Sequence. (The requester must deliver the full byte count of the Sequence after the first data phase is accepted. See Section 2.1.)

The target signals Retry by asserting **DEVSEL#** and **STOP#** and keeping **TRDY#** deasserted on the first data phase as shown in Figure 2-52.

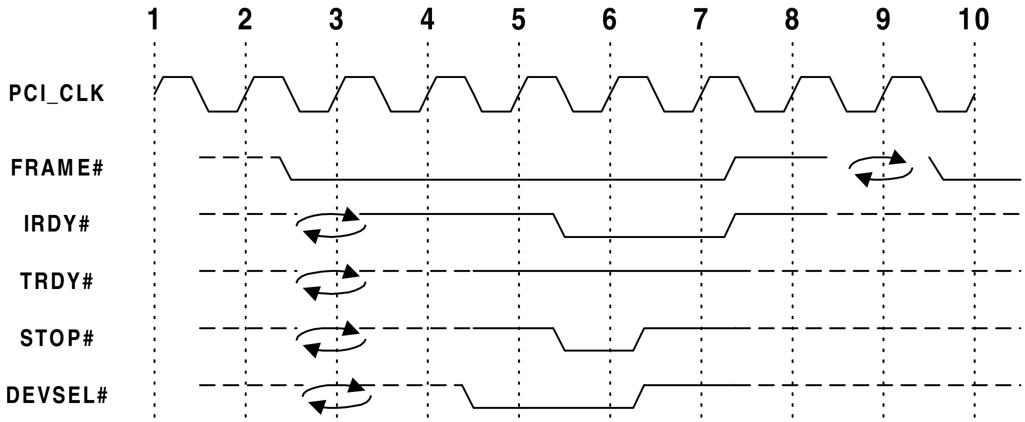


Figure 2-52: Retry Termination

2.11.2.4. Split Response Termination

The target signals that it has enqueued the transaction as a Split Request by signaling Split Response. The target signals Split Response by asserting **TRDY#**, deasserting **DEVSEL#**, and keeping **STOP#** deasserted on the first data phase of the transaction (with or without preceding wait states up to the maximum specified in Section 2.9.1). Figure 2-53 shows Split Response for a read transaction (either burst or DWORD). The target drives all bits of the **AD** bus high during the clock in which it signals Split Response of a read transaction. Figure 2-54 shows Split Response for a write transaction.

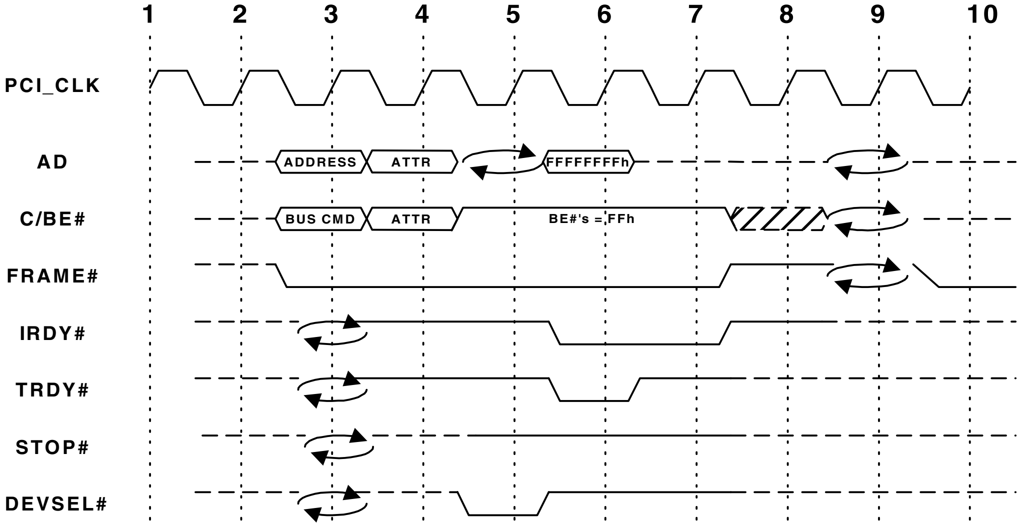


Figure 2-53: Split Response Termination for a Read Transaction

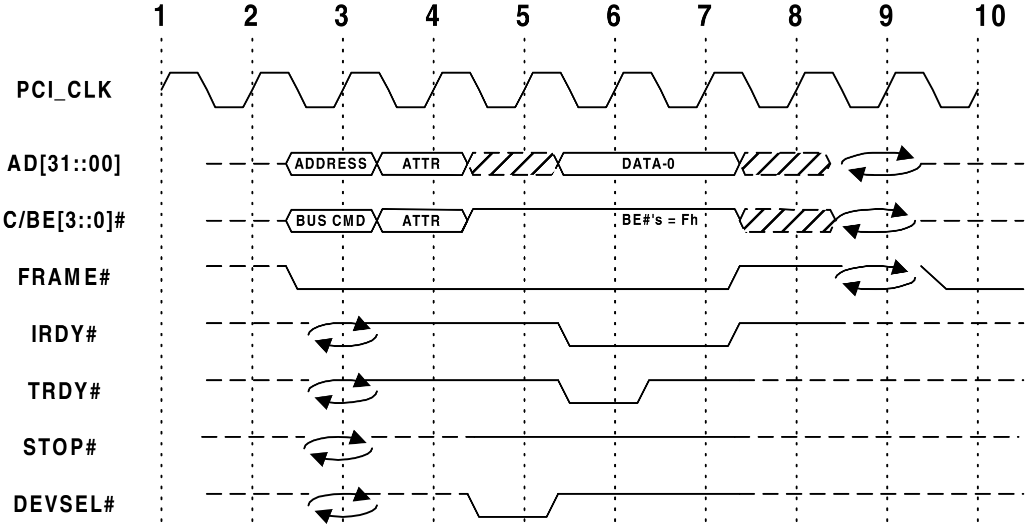


Figure 2-54: Split Response Termination for a DWORD Write Transaction

2.11.2.5. Target-Abort Termination

As in conventional PCI, the target signals Target-Abort to end the transaction and to notify the initiator not to repeat it. As in conventional PCI, PCI-X targets are permitted to limit the size and type of read transactions that they execute and to terminate all others with Target-Abort. For example, if a PCI-X device supports only DWORD read transactions in a certain address range, and if the device receives a read request for more than a DWORD, the device is permitted to signal Target-Abort. See Section 2.10.1 for examples of the use of Target-Abort for read transactions that address both immediate-capable and split-only regions. (In some cases independent memory write Sequences are combined by host or conventional PCI bridges, so targets are not permitted to use Target-Abort to limit the size of memory write transactions they execute.) The use of Target-Abort for Split Completion transactions is restricted. (See Section 2.10.5.)

It should be understood that signaling Target-Abort typically has deleterious effects on the system, possibly including halting execution of the system software, and device designers should avoid these circumstances whenever possible.

The target signals Target-Abort by asserting **STOP#** and deasserting **DEVSEL#** and **TRDY#**. The target is permitted to signal Target-Abort on any data phase. The transaction and the Sequence end on the clock in which the target signals Target-Abort regardless of its relationship to an ADB or the number of bytes remaining to be sent in the Sequence. The initiator deasserts **FRAME#** and **IRDY#** two clocks after the target signals Target-Abort, unless one or both of these signals deasserts sooner because the transaction was already about to end (e.g., byte count satisfied, initiator or target disconnection on an ADB).

Figure 2-55 illustrates a Target-Abort in the first data phase of a transaction. Figure 2-56 illustrates a Target-Abort after the target has signaled Data Transfer for several data phases of a burst transaction.

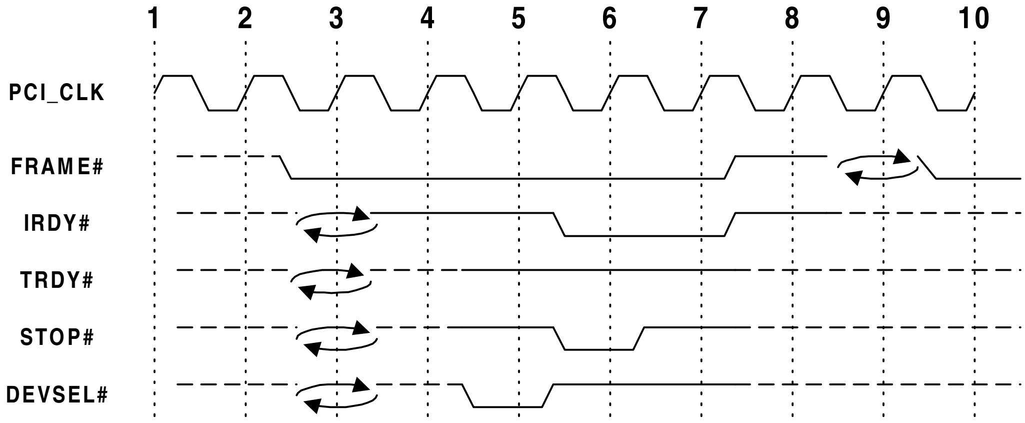


Figure 2-55: Target-Abort on First Data Phase

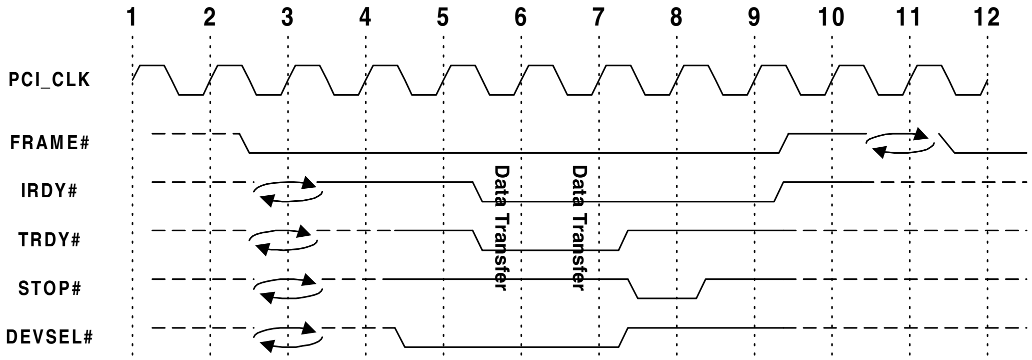


Figure 2-56: Target-Abort after Data Transfer

2.12. Bus Width

As in conventional PCI, PCI-X devices are permitted to implement either a 64-bit or a 32-bit version of the interface. The width of the address is independent of the width of the device or of the data transfer. Addresses are driven in one clock for non-memory transactions and one or two clocks for memory transactions depending on whether the address is below the first 4 GB boundary (see Section 2.12.1). Attributes are always driven in a single clock for both 64- and 32-bit devices. The width of a PCI-X data transfer is negotiated between the initiator and target on each transaction using **REQ64#** and **ACK64#** in a manner similar to conventional PCI (see Section 2.12.3).

Devices discover the width of the bus to which they are attached by the state of **REQ64#** at the rising edge of **RST#** as specified in PCI 2.2.

At various places throughout the discussion of bus widths, a bus or a portion of a bus is described as "reserved." Unless otherwise noted, the state of a "reserved" bus is not specified and is ignored by the device receiving the bus.

2.12.1. Address Width

PCI-X support for varying the width of the address minimizes the changes from the corresponding support in conventional PCI. The following requirements for PCI-X are the same as for conventional PCI:

1. Addresses in I/O and Configuration Spaces are always 32-bit. Interrupt Acknowledge, Special Cycle, and Split Completion transactions always have a 32-bit address field, even though they use it for other purposes. The upper **AD** bus is reserved during the address phase of these transactions. Addresses in Memory Space are permitted up to 64-bits.
2. If the address of a transaction is less than 4 GB, the following are all true:
    a. The transaction uses a single address cycle.
    b. During the address phase, a 64-bit initiator drives the address on **AD[31::00]**, and **AD[63::32]** are reserved. A 32-bit initiator drives the address on **AD[31::00]**.
    c. During the address phase, a 64-bit initiator drives the command on **C/BE[3::0]#**, and **C/BE[7::4]#** are reserved. A 32-bit initiator drives the command on **C/BE[3::0]#**.

3. If the address of a transaction is greater than or equal to 4 GB, the transaction uses a dual address cycle.

    a. If the initiator is 64-bits wide, the following are all true:

        i) In the first address phase, **AD[63::32]** contain the upper half of the address, and **AD[31::00]** contain the lower half of the address. In the second address phase, **AD[63::32]** and **AD[31::00]** contain duplicate copies of the upper half of the address.

        ii) In the first address phase, **C/BE[3::0]#** contain the Dual Address Cycle command and **C/BE[7::4]#** contain the transaction command. In the second address phase, **C/BE[3::0]#** and **C/BE[7::4]#** contain duplicate copies of the transaction command.

    b. If the initiator is 32-bits wide, the following are all true:

        i) In the first address phase, **AD[31::00]** contain the lower half of the address. In the second address phase **AD[31::00]** contain the upper half of the address.

        ii) In the first address phase, **C/BE[3::0]#** contain the Dual Address Cycle command. In the second address phase, **C/BE[3::0]#** contain the actual transaction command.

4. **DEVSEL#** timing designations measure from the second address phase of a transaction with a dual address cycle. Note that it is possible for a 64-bit target to decode its address from a 64-bit initiator after only the first address phase of a dual address cycle and be ready to assert **DEVSEL#** sooner than a 32-bit target. (However, in PCI-X mode no device is permitted to assert **DEVSEL#** sooner than the first clock after the attribute phase.)

5. The rest of the transaction proceeds identically after either a single address cycle or a dual address cycle.

The following requirements for PCI-X are different from conventional PCI:

1. All PCI-X devices that initiate or respond to memory transaction must support 64-bit memory addressing. This includes the following:

    a. All devices that initiate memory transactions must be capable of generating addresses greater than 4 GB.

    b. All targets that include memory Base Address Registers (except Expansion ROM Base Address registers) must implement the 64-bit versions using the method defined in PCI 2.2. (PCI-X devices set the Prefetchable bit in all memory Base Address registers unless the range includes addresses with read side effects or addresses in which the device does nottolerate write merging. See Section 7.1.)

    c. All prefetchable memory range registers in PCI-X bridges must support the 64-bit versions of those registers as defined in Bridge 1.1.

2. Split Completions always have a single address phase both for 64-bit and 32-bit initiators. See Section 2.10.3.

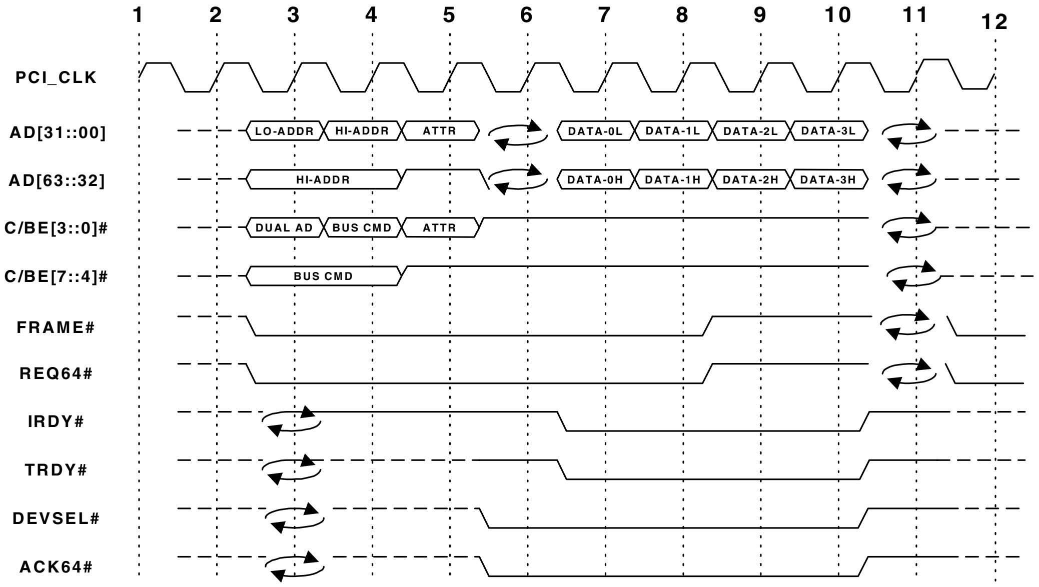


Figure 2-57: Dual Address Cycle 64-bit Memory Read Burst Transaction

Figure 2-57 illustrates a 64-bit initiator executing a transaction with a dual address cycle for a 64-bit burst read transaction in which the target signals Data Transfer until the end (initiator disconnection or byte count satisfied). The initiator drives the entire address (lower address on **AD[31::00]** and upper address on **AD[63::32]**) and both commands (Dual Address Cycle on **C/BE[3::0]#** and the actual transaction command on **C/BE[7::4]#**) during the initial address phase at clock 3. On the second clock of the address phase, the initiator drives the upper address on **AD[31::00]** (and **AD[63::32]**) and the transaction command on **C/BE[3::0]#** (and **C/BE[7::4]#**). The one-clock attribute phase in clock 5 immediately follows the second address phase. The figure shows a 64-bit target responding with device select timing A by asserting **DEVSEL#** in clock 6. **DEVSEL#** is never asserted earlier than the clock after the attribute phase (device timing A).

2.12.2. Attribute Width

Attributes are always driven in a single attribute phase both for 64-bit and 32-bit initiators. **AD[63::32]** and **C/BE[7::4]#** are driven high during the attribute phase of transactions from a 64-bit initiator.

2.12.3. Data Transfer Width

PCI-X support for varying the width of data transfers minimizes the changes from the corresponding support in conventional PCI. The following requirements for PCI-X are the same as for conventional PCI:

1. Only memory transactions use 64-bit data transfers. All other transactions use 32-bit data transfers.
2. 64-bit addressing is independent of the width of the data transfers.
3. A device with a 64-bit bus is permitted to initiate and respond to transactions either as a 64-bit device or as a 32-bit device.

4. A 64-bit initiator asserts **REQ64#** with the same timing as **FRAME#** to request a 64-bit data transfer. It deasserts **REQ64#** with the same timing as **FRAME#** at the end of the transaction.
5. If a 64-bit initiator addresses a device that responds as a 32-bit target, the initiator is permitted either to drive to a valid but unspecified state or to float the **C/BE[7::4]#** bus after the first data phase. If the transaction is a write, the initiator is permitted to do the same with the **AD[63::32]** bus and **PAR64**.

The following requirements for PCI-X are different from conventional PCI:

1. All PCI-X devices support a status bit indicating whether they are a 64- or 32-bit device. See Section 7.2.4.
2. Only burst transactions use 64-bit transfers. DWORD transactions use 32-bit transfers.
3. Allowable disconnect boundaries are unaffected by the width of the data transfer. A 32-bit transfer has twice as many data phases between two ADBs.
4. **AD[2]** is either 0 or 1, depending on the starting byte address of the transaction. (Conventional PCI requires **AD[2]** to be 0 for 64-bit data transfers because the byte enables indicate the actual starting address.)
5. The following rules apply to memory write and Split Completion transactions from a 64-bit initiator. (Split Completion transactions have only a partial starting address, as described in Section 2.10.3.)

    a. If **AD[2]** of the starting byte address is 1 (that is, the starting address of the transaction is in the upper 32-bits of the bus), the 64-bit initiator must drive the data both on **AD[63::32]** and **AD[31::00]**, and the byte enables both on **C/BE[7::4]#** and **C/BE[3::0]#** of the first data phase.

    b. If the target asserts **ACK64#** when it asserts **DEVSEL#** (indicating it is a 64-bit target), and the target inserts wait states, the initiator must toggle between the first and second QWORD data phases on the **AD[63::00]** and byte enables on **C/BE[7::0]#**. If the transaction starts on an odd DWORD, that DWORD and its byte enables must be copied down to the lower half of the bus each time the first data phase is repeated.

    c. If the target does not assert **ACK64#** when it asserts **DEVSEL#** (indicating it is responding as a 32-bit target) and the target inserts wait states, the initiator must toggle between the first and second DWORD data phases of the transaction on **AD[31::00]** and byte enables on **C/BE[3::0]#** (as it would if it were a 32-bit initiator on a 32-bit bus).

Implementation Note: Deassertion of ACK64# for Single Data Phase Disconnect

As in conventional PCI, the width of the transaction is established by the state of **ACK64#** on the first clock that **DEVSEL#** is asserted, and **ACK64#** always deasserts when **DEVSEL#** deasserts. If a 64-bit PCI-X target asserts **ACK64#** with **DEVSEL#** and then signals Single Data Phase Disconnect (see Section 2.11.2.1), the target deasserts **DEVSEL#** and **ACK64#** on the last clock of the data phase (the clock in which data transfers). This data phase is 64-bits wide, even though **ACK64#** is deasserted during the data phase.

Figure 2-58 through Figure 2-65 illustrate the cases in which a device initiates a 64-bit transaction and a 32-bit target responds. Split Completion transactions behave the same as write transactions except that the **C/BE#** bus is driven high by the initiator. In each case, the data bus shows the low and high DWORDs from the point of view of a 64-bit initiator.

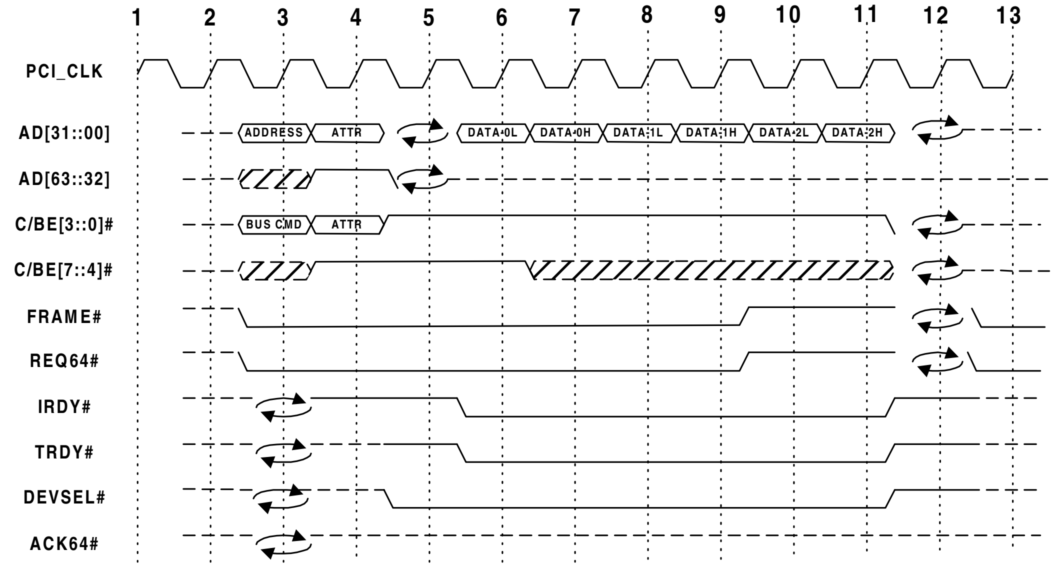


Figure 2-58: 64-bit Initiator Reading from 32-bit Target Starting on Even DWORD

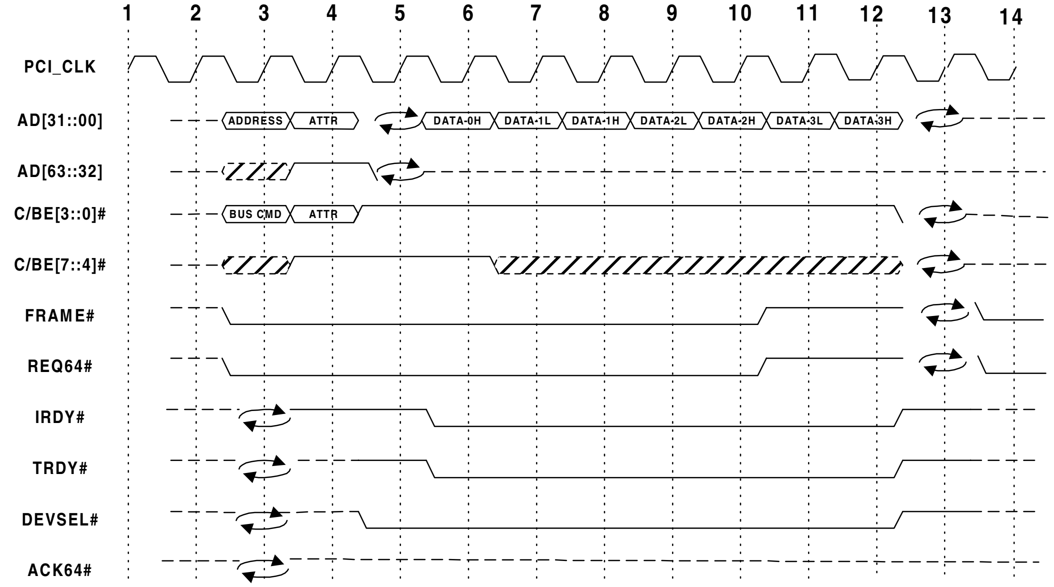


Figure 2-59: 64-bit Initiator Reading from 32-bit Target Starting on Odd DWORD

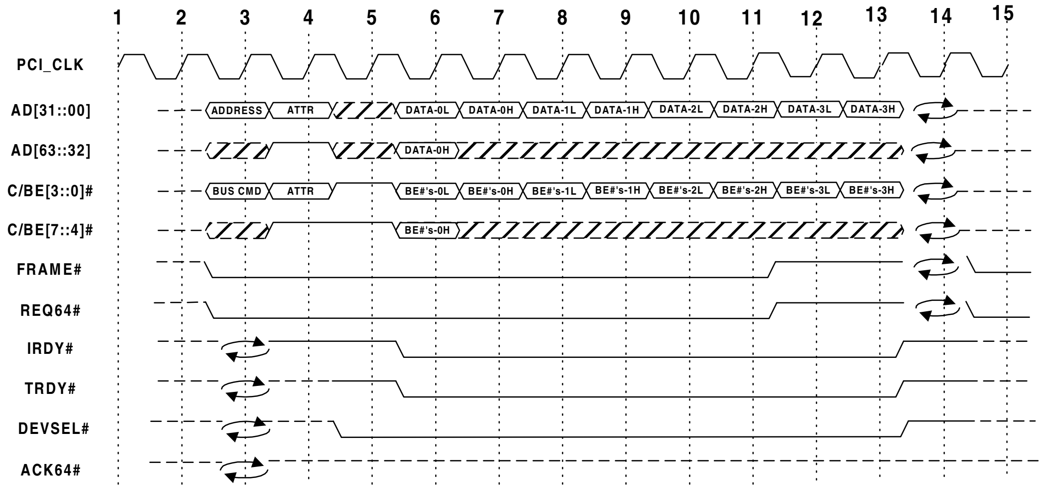


Figure 2-60: 64-bit Initiator Writing to 32-bit Target Starting on Even DWORD

Figure 2-61 illustrates the case of a device initiating a 64-bit write that begins on an odd DWORD. Split Completion timing would be the same, except the **C/BE#** bus is reserved in the data phases. In this case, the initiator must duplicate the first DWORD of data and byte enables on both the upper and lower bus halves. Notice that in this case, one or more byte enables in **C/BE[3::0]#** are asserted even though the transaction starts on an odd DWORD and no byte enable before the starting address of a write transaction is allowed to be asserted. If a 32-bit target responds by asserting **DEVSEL#** without asserting **ACK64#** (as is shown in Figure 2-61), the 32-bit target captures the first data and byte enables from the lower half of the bus. When the initiator observes **DEVSEL#** asserted with **ACK64#** deasserted, it continues the transaction on the lower bus half as a 32-bit initiator would. Notice that Figure 2-61 illustrates initiator termination after an even DWORD, which can only occur if the byte count is satisfied in that DWORD. (An ADB would always occur after an odd DWORD.)

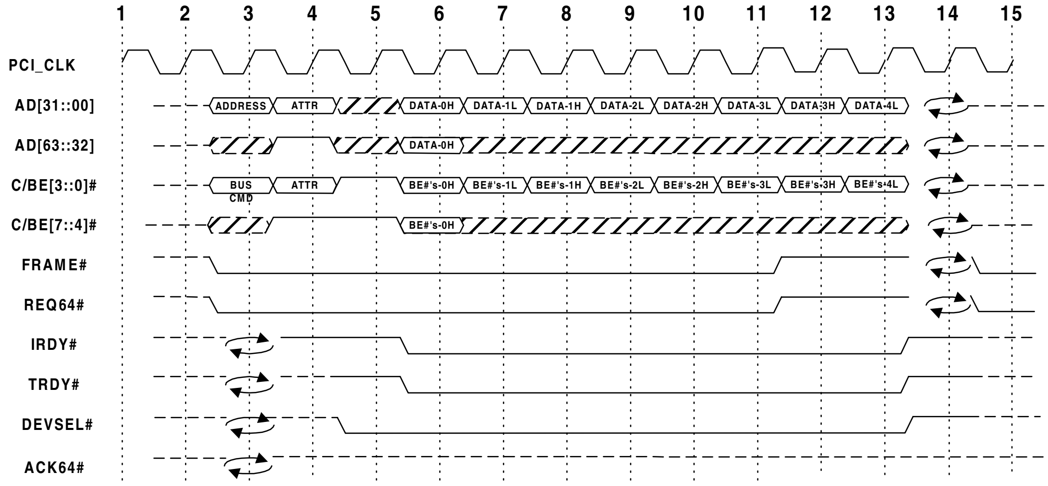


Figure 2-61: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD

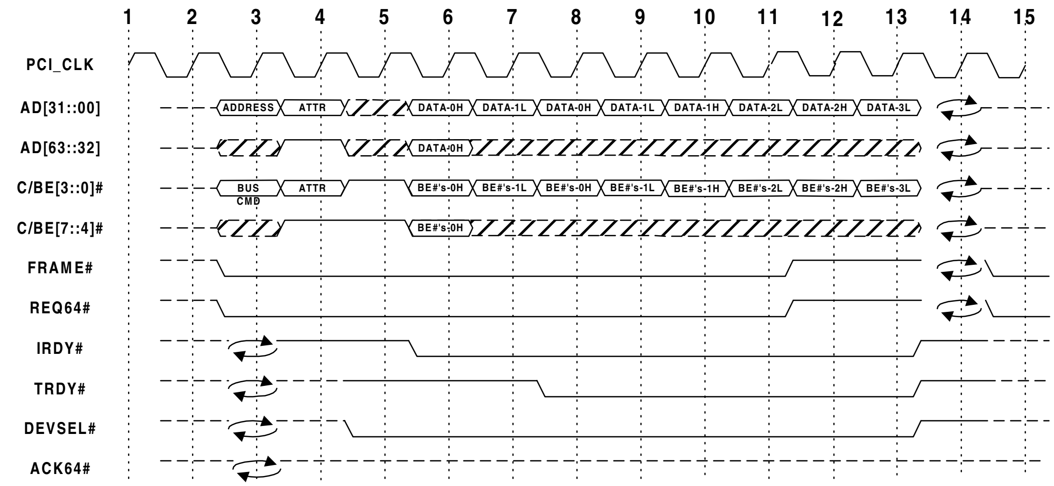


Figure 2-62: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, with DEVSEL# Decode A and Two Initial Wait States

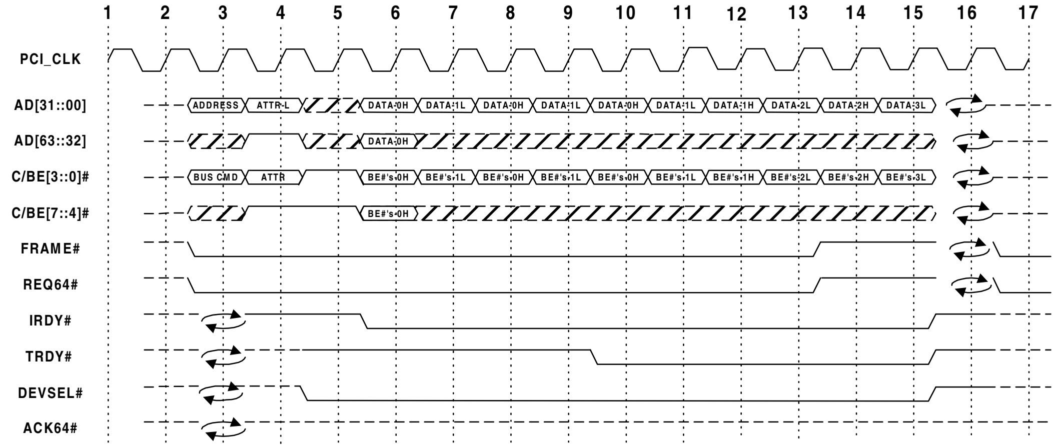


Figure 2-63: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, with DEVSEL# Decode A and Four Initial Wait States

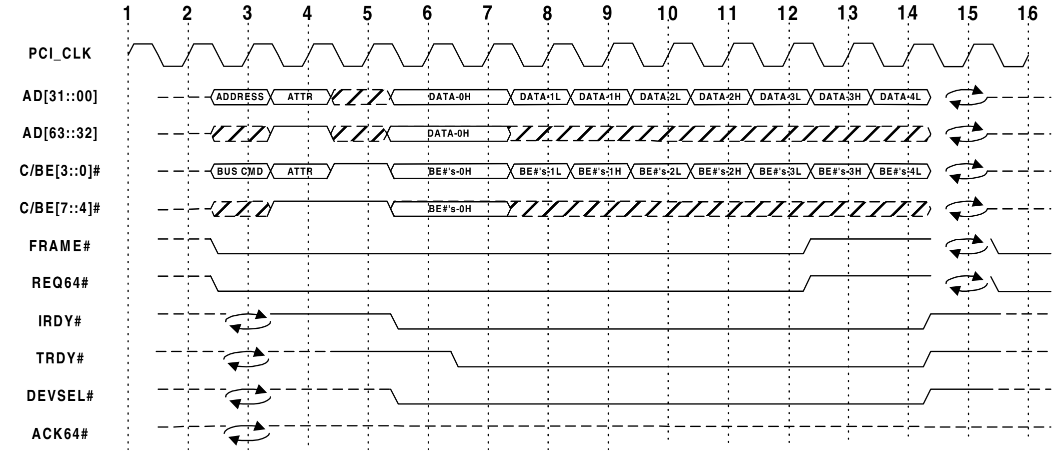


Figure 2-64: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, with DEVSEL# Decode B

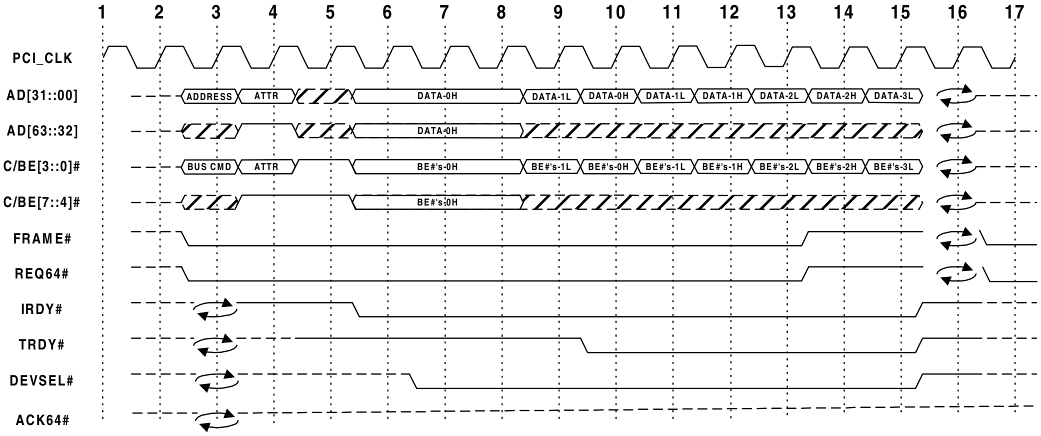


Figure 2-65: 64-bit Initiator Writing to 32-bit Target Starting on Odd DWORD, with DEVSEL# Decode C and Two Initial Wait States

2.13. Required Acceptance and Completion Rules for Simple Devices

Using the terminology of PCI 2.2, a "simple device" is one that does *not* implement internal posting of memory write transactions that must be initiated by the device on the PCI-X interface.

As in conventional PCI, a simple PCI -X device is never allowed (with the exception of Split Completions described below) to make the acceptance of a transaction as a target contingent upon the prior completion of another transaction as an initiator. Furthermore, a simple PCI-X device is never allowed to make the completion of a Sequence for which it is the completer contingent upon another device completing a Sequence for which the simple device is the requester. That is, a simple PCI-X device that has terminated a transaction with Split Response is required to request the bus to initiate the Split Completion for that Sequence independent of other Sequences the simple device initiates. (This is analogous to the requirement in PCI 2.2 for simple conventional devices not to make the completion of any transaction as a target contingent upon the prior completion of any other transaction as an initiator.) (See Section 8.4.4 for the corresponding rule for bridges to allow a Split Completion to pass a Split Request.)

A simple PCI-X device is permitted to terminate a memory write transaction with Retry only for temporary conditions that are guaranteed to resolve over time. After terminating a memory write transaction with Retry, a PCI-X device must be able to accept a memory write transaction within 267 clocks on buses initialized to 133 MHz mode, 200 clocks on buses initialized to 100 MHz mode, and 133 clocks on buses initialized to 66 MHz PCI-X mode. (See Section 6.2 for a description of mode and frequency initialization.) This corresponds to 2 µs in systems running at the maximum frequency of each mode. Devices are permitted to limit their completion time to 2 µs independent of the frequency of the clock. PCI 2.2 calls this the Maximum Completion Time and defines how the number is to be measured (and also specifies a limit of 10 µs for conventional PCI devices). This requirement applies to all devices in their normal mode of operation with their device drivers. In its normal mode of operation, the device driver must not initiate a memory write to the device unless the device is able to accept it within the specified limit. This requirement does not apply to diagnostic modes or device-specific cases that are not intended for normal use in a system with other PCI-X devices.

To provide backward compatibility with PCI-to-PCI bridges designed to revision 1.0 of the *PCI-to-PCI Bridge Architecture Specification*, all PCI-X devices are required to accept memory write transactions even while executing a previous Split Transaction (that is, after signaling Split Response and prior to initiating the Split Completion). (This is analogous to the requirement in PCI 2.2 for conventional devices to accept memory write transactions even while executing a Delayed Transaction.)

A simple PCI-X device that is executing a Split Transaction (as a completer) is permitted to terminate a non-posted request with Retry until it finishes its Split Completion as an initiator. Completers execute a finite number of Split Transactions at one time. However, in the normal mode of operation with its device driver, a device is required to terminate an I/O write transaction with something other than Retry within the same Maximum Completion Time limit as specified above for memory write transactions. If the device executes the I/O write transaction as a Split Transaction, the device must also request the bus to execute the Split Completion within the Maximum Completion Time limit. In its normal mode of operation, the device driver must not initiate an I/O write to the device unless the device is able to complete it within the specified limit. This requirement does not apply to diagnostic modes or device-specific cases that are not intended for normal use in a system with other PCI-X devices.

A simple device is permitted to execute more than one Split Transaction (as a completer) at the same time. In this case, the device is permitted to initiate the Split Completions for different Sequences in any order. (Split Completions for the same Sequence must be initiated in address order.) See Section 2.10.1 for additional details.

Implementation Note: Completers Executing Multiple Split Transactions

Devices such as host bridges that are routinely addressed by multiple other devices are encouraged to complete multiple Split Transactions concurrently. In other applications, devices are rarely or never addressed by a second Split Transaction before the previous Split Transaction completes. A device for such an application benefits little from completing multiple Split Transactions concurrently and is permitted to execute a single Split Transaction at a time and terminate all other non-posted transactions with Retry until it finishes its Split Completion.

If an application benefits from completing multiple transactions of one type concurrently but not others, the device might continue to accept and execute some non-posted transactions and terminate others with Retry. For example, if a device is designed to complete multiple Memory Read DWORD transactions concurrently, but only a single Configuration Read transaction, the device would signal Split Response to the first Memory Read DWORD and Configuration Read DWORD transactions. The device would also signal Split Response to a subsequent Memory Read DWORD transaction that was received before the device executed the Split Completion for the first one. However, the device would signal Retry if it received a subsequent Configuration Read before the device executed the Split Completion for the first one.

The device driver should understand the number of each kind of transaction its device executes concurrently. The device driver is discouraged from issuing more transactions than the device is able to execute. If a device driver issues more requests than a device is able to execute, the excess requests back up in bridges in the system and potentially degrade system performance.

A simple PCI-X device is required to accept all Split Completion transactions that correspond to the device's outstanding Split Requests. The simple device is not permitted to terminate a Split Completion transaction with Retry or Disconnect at Next ADB. See Section 2.10.5 for more details.

2.14. Quiescing Device Operation

From time to time, a device's operation must be stopped by the software so that the device's state can be changed. In all cases, the device must accept Split Completions corresponding to that device's Split Requests. If a transaction has been terminated with Split Response, the requester must accept all the data requested (or a Split Completion Exception message that indicates no more data is coming). If the change of state of an otherwise normally functioning device jeopardizes that device's ability to accept its outstanding Split Completions, the state change must be delayed until all outstanding transactions finish. (A device that has ceased to function normally must be reset regardless of the state of its outstanding Split Transactions. All devices must return to their initial states when **RST#** is asserted.)

Examples of some situations in which the device's state change must be delayed until all outstanding transactions finish include the following:

- PCI hot-removal operation. The PCI HP 1.0 requires the orderly shut-down of a device before it can be removed.
- Changing a function's PCI Power Management state to $D1$, $D2$, or $D3_{hot}$. (See Section 3.3.)
- Software-initiated reset of the card.

How the software device driver determines that no transactions remain outstanding is not controlled by this specification. Some example methods include the following:

- The device driver stops giving new work for the device and uses normal operational status indicators to determine when all the old work is complete.
- The device driver sets a device-specific control bit whose function is to quiesce device operation. The device hardware stops issuing new transactions and sets a status indication when all outstanding transactions complete. The device driver waits for the status indicator to be set.

2.15. Snooping PCI-X Transactions

A device is said to snoop a transaction if it monitors a transaction for which it is neither the initiator nor the target. Snooping is most often done by diagnostic tools like bus analyzers or system management devices. As in conventional PCI, snooping of PCI -X transactions is allowed only if the snooping device is on the path between the requester and the completer.

Implementation Note: Snooping PCI-X Transactions

Snooping of Immediate Transactions in PCI-X is very similar to conventional PCI. If data is transferred, the snooping device latches the data when the target signals Data Transfer, Single Data Phase Disconnect, or Disconnect at Next ADB.

If a write transaction is terminated with Split Response, the snooping device latches the data during the Split Response. If the snooping agent tracks error conditions or completion order, it must also wait for the corresponding Split Completion.

If a read transaction is terminated with Split Response, the snooping agent must capture the command, address, and attributes during the Split Request and capture the data during the Split Completion.

3. Device Requirements

3.1. Source Sampling

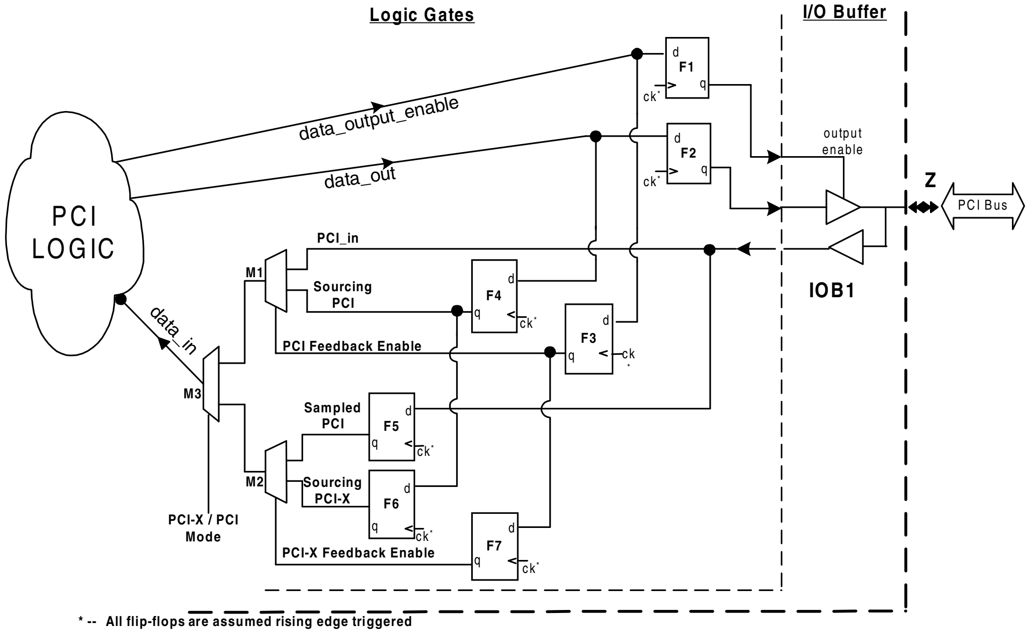
Like conventional PCI, PCI-X devices are not permitted to drive and receive a signal at the same time. The electrical design of the bus does not guarantee that the signal meets the setup time specified in Section 9.4.2 at a pin that is driving the bus. If the state of an input/output signal is used by logic inside a device during a clock cycle that the device is driving the signal, an internal version of that signal must be used.

Implementation Note: Source Sampling

One approach to satisfying the requirement not to drive and receive a bus signal at the same time, is to implement a multiplexer in the input path for any signal that the device monitors while the device is driving the signal. PCI-X devices would receive on one input the registered input signal from the I/O pad and on the other an internal equivalent of the signal being driven onto the bus with the proper registered delay. The multiplexer control would be a registered delayed version of the output enable (or equivalent) that automatically switches the multiplexer to use the internal signal.

Figure 3-1 illustrates an implementation of the logic a PCI-X device needs when monitoring its own signals on the bus. Notice that flip-flops F3 and F4 provide the same output register delay as flip-flops F1 and F2, with F3 output controlling multiplexer M1 to provide the conventional PCI source sampling requirement. In addition, for PCI-X source sampling requirements, flip-flops F6 and F7 provide that same input register delay as flip-flop F5, with F7 output controlling multiplexer M2. Switching between conventional PCI and PCI-X mode is multiplexer M3, which is controlled by the PCI-X/PCI mode enable signal set at the rising edge of **RST#**.

Note: Figure 3-1 is only a design aid. Designers are free to choose an equivalent implementation that helps them meet conventional PCI setup to output delay requirements. Details such as **RST#** and JTAG connection have been omitted to simplify the diagram.



* -- All flip-flops are assumed rising edge triggered

Figure 3-1: A Logic Block Diagram for Bypassing Source Sampling

3.2. Message-Signaled Interrupts

Support of message-signaled interrupts is optional for systems and system software.

PCI-X devices that generate interrupts are required to support message-signaled interrupts and must support a 64-bit message address. Implementation of these features is specified in PCI 2.2. Devices that require interrupts in systems that do not support message-signaled interrupts must also implement interrupt pins.

System software must not assume that a message-capable device has an interrupt pin. Devices that rely on polling for device service in systems that do not support message-signaled interrupts are permitted to implement messages to increase performance in systems that do support it.

The requester of a message-signaled interrupt transaction must set the No Snoop and Relaxed Ordering bit in the Requester Attributes to 0.

3.3. PCI Power Management

PCI-X devices intended for use on add-in cards are required to support PCI power management, as defined in the PCI PM 1.1. Host bridges and other devices intended for use only on the system board are exempt from this requirement. This requirement applies only to device hardware. Operating system requirements determine whether the device driver software supports PCI power management. Refer to Section 12 for additional information on implementing power management in devices.

The system is required not to change the frequency of the clock input to a device beyond the limits stated in Section 9.4.1, even if all functions in the device are in $D3_{hot}$ state.

If the function is in $D3_{hot}$ state but **RST#** remains deasserted, the function must maintain its frequency and mode information (from the PCI-X initialization pattern). (In other words, the "soft reset" that the function performs when changing from $D3_{hot}$ to *D0* must not affect the mode and frequency information that was captured by the function on the last rising edge of **RST#**.)

PCI-X functions in *D1*, *D2*, and $D3_{hot}$ are permitted to signal Split Response to a configuration transaction only and initiate the corresponding Split Completion transaction. (PCI PM1.1 requires functions in *D1*, *D2*, or $D3_{hot}$ to respond only to configuration transactions and not initiate other transactions. This implies the function must be quiesced before being placed in any of these states (see Section 2.14). However, if a function in one of these states executes configuration transactions as Split Transactions, it must initiate Split Completions.)

4. Arbitration

This section presents requirements for bus arbitration that affect initiators and the central bus arbiter.

4.1. Arbitration Signaling Protocol

The following PCI-X arbiter characteristics remain the same as for conventional PCI:

- No arbitration algorithm is specified. The arbiter is permitted to assign priorities using any method that grants each initiator fair access to the bus.
- The arbiter uses the same **REQ#** and **GNT#** signals defined in PCI 2.2.
- Initiators that require access to the bus are allowed to assert their **REQ#** signals on any clock.
- An initiator is permitted to issue any number of transactions as long as its **GNT#** remains asserted. If **GNT#** is deasserted, the initiator must not start a new transaction. (In PCI-X mode, the **GNT#** input is registered. When comparing PCI-X transactions to conventional transactions, the relevant clock for **GNT#** is one clock earlier in PCI-X mode than in conventional mode.)
- While a transaction from one initiator is in progress on the bus, the arbiter is permitted to deassert **GNT#** to the current initiator and to assert and deassert **GNT#** to other initiators (with some restrictions listed below). The next initiator cannot start a transaction until the current transaction completes.
- Each initiator includes a Latency Timer that is loaded with a preset value each time the initiator starts a new transaction and counts down the number of clocks that **FRAME#** is asserted. If **GNT#** is deasserted when the Latency Timer expires, the initiator disconnects the current transaction as soon as possible (in most cases, the next ADB).

Implementation Note: Fair Arbitration

A fair algorithm is one in which all devices that request the bus are eventually granted the bus, independent of other device's requests for the bus. A fair algorithm is not required to give equal access to every requester. The algorithm is permitted to allow some requesters greater access to the bus than others. One example of a fair algorithm is a multi-level round-robin algorithm in which the arbiter grants the bus to the source bridge for every even transaction and rotates among the other requesters on the odd transactions.

A fixed-priority algorithm in which one device is always granted the bus and blocks another device indefinitely is not fair.

The following PCI-X arbiter characteristics are different from conventional PCI:

- In PCI-X mode, all **REQ#** and **GNT#** signals are registered by the arbiter as well as by all initiators. That is, they are clocked directly into and out of flip-flops at the device interface.
- All fast back-to-back transactions as defined in PCI 2.2 are not permitted in PCI-X mode.

Implementation Note: Meeting REQ# and GNT# Timing Requirements

The system must satisfy setup and hold time requirements for **REQ#** and **GNT#** regardless of whether the bus is operating in PCI-X mode or conventional mode. One alternative is for the arbiter of a bus that is capable of operating in PCI-X mode to clock all **REQ#** signals directly into registers and clock all **GNT#** signals directly from registers, regardless of whether the bus is operating in PCI-X mode or conventional mode. Another alternative is to register **REQ#** and **GNT#** only if the bus is operating in PCI-X mode.

4.1.1. Device Requirements

In most cases, an initiator starts a transaction by driving the **AD** and **C/BE#** buses and asserting **FRAME#** on the same clock. However, if an initiator is starting a configuration transaction, the initiator drives the **AD** and **C/BE#** buses for four clocks and then asserts **FRAME#** (see Section 2.7.2.1). The following discussion uses the phrase “start a transaction” to indicate the first clock in which the device drives the **AD** and **C/BE#** buses for the pending transaction. The initiator of a configuration transaction has additional restrictions before asserting **FRAME#**, which are described in Section 2.7.2.1.

An initiator is permitted to assert and deassert **REQ#** on any clock. Unlike conventional PCI, there is no requirement to deassert **REQ#** after a target termination (**STOP#** asserted). (The arbiter is assumed to monitor bus transactions to determine when a transaction has been target terminated if the arbiter uses this information in its arbitration algorithm.) An initiator is permitted to deassert **REQ#** on any clock independent of whether **GNT#** is asserted. An initiator is permitted to deassert **REQ#** without initiating a transaction after **GNT#** is asserted. However, if **GNT#** is asserted and the bus is idle in clock N, and **GNT#** remains asserted, the initiator must either assert **FRAME#** or deassert **REQ#** on or before clock N+6.

In PCI-X mode, the **GNT#** input is registered in the initiator. When comparing PCI-X transactions to conventional transactions, the relevant state of **GNT#** is one clock earlier in PCI-X mode than in conventional mode. In general, **GNT#** must be asserted two clocks prior to the start of a transaction. This also means that the initiator is permitted to start a transaction one clock after **GNT#** deasserts.

An initiator acquiring the bus is permitted to start a transaction in any clock N in which the initiator’s **GNT#** was asserted on clock N-2 and either of the following conditions is true:

Case 1. The bus was idle (**FRAME#** and **IRDY#** are both deasserted) on clock N-2.

Case 2. **FRAME#** was deasserted and **IRDY#** was asserted on clock N-3.

In the first case, there is a minimum of two idle clocks between transactions from different initiators. In the second case, the minimum number of idle clocks is reduced to one following initiator terminated transactions. By monitoring the transaction of the preceding bus owner and observing when it deasserts **FRAME#**, the new bus owner starts a transaction with only one idle clock.

If the above conditions are met, the initiator is permitted to start a new transaction on clock N even if **GNT#** is deasserted on clock N-1.

If an initiator has more than one transaction to execute, and **GNT#** is asserted on the last clock of the preceding transaction (that is, one clock before the idle clock and two clocks before the start of the next transaction), the initiator is permitted to start the next transaction with a single idle clock between the two transactions.

All fast back-to-back transactions as defined in PCI 2.2 are not permitted in PCI-X mode.

Some devices include multiple sources of initiator activity. Examples of this include the following:

- A multifunction device.
- A single-function device with multiple sources of initiator traffic, like a UART or LAN controller with separate receiver and transmitter logic.
- A device that signals Split Response when addressed as a target and also initiates its own requests. The Split Completion is a separate source of initiator activity.

If a device includes multiple sources of initiator activity, each of these sources must share a single **REQ#** and **GNT#** signal pair. An arbiter internal to the device must determine which source uses the bus when **GNT#** is asserted. This internal arbitration algorithm is not specified but is recommended to be fair to all internal sources. If the device initiates Split Completion transactions, they must have fair access to the bus.

4.1.2. Arbiter Requirements

If no **GNT#** signals are asserted, the arbiter is permitted to assert any single **GNT#** on any clock.

The arbiter must provide each device a fair opportunity to initiate configuration transactions. As described in Section 2.7.2.1, **GNT#** must be asserted for five clocks while the bus is idle for the device to initiate a configuration transaction. **GNT#** is permitted to be asserted for fewer than five clocks at other times, for example, if the arbiter intends to grant the bus to a higher priority device before the bus is in the idle state. The arbiter is permitted to assume that a device is broken, deassert **GNT#** to that device, and keep it deasserted if all of the following are true:

- **GNT#** is asserted and the bus is idle on clock N.
- **GNT#** remains asserted through clock N+5.
- The device keeps **FRAME#** deasserted and **REQ#** asserted through clock N+6.

If the arbiter deasserts **GNT#** to one initiator, it must not assert another **GNT#** until the next clock. (The first initiator is permitted to sample its **GNT#** on the last clock it was asserted and assert **FRAME#** one clock after **GNT#** deasserts. In this case, **GNT#** to the next initiator asserts on the same clock as **FRAME#** from the current initiator.)

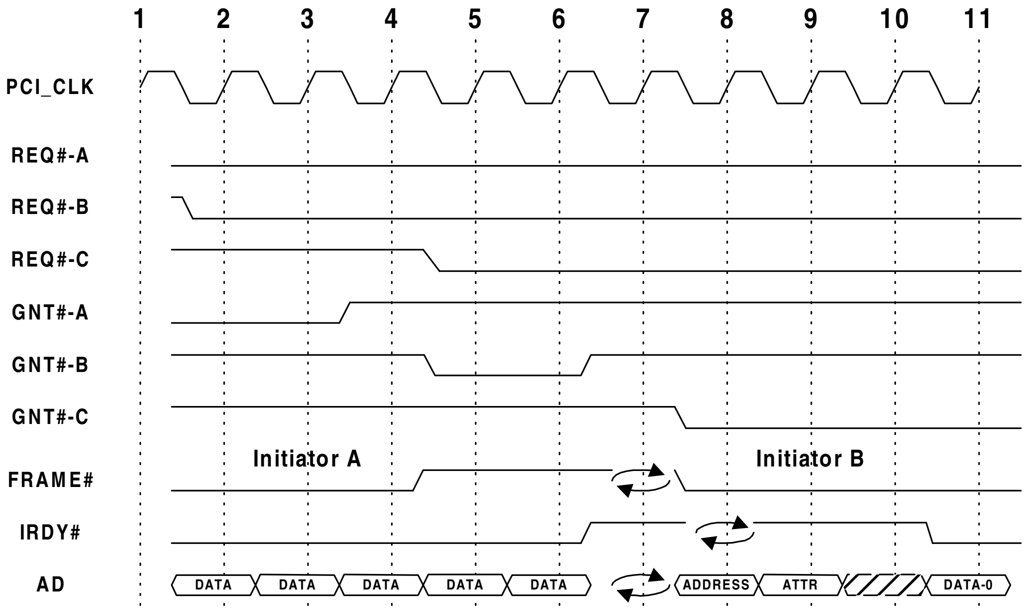


Figure 4-1: Arbitration Example

Figure 4-1 illustrates an arbitration example switching between several initiators. The figure shows that after the arbiter asserts **GNT#**, an initiator (B in the figure) must wait for the current bus transaction to end before it can start its transaction. The figure also shows that an initiator can start that transaction as late as one clock after its **GNT#** is deasserted.

Figure 4-1 shows three initiators, A, B, and C, in increasing order of arbitration priority. The example starts with initiator A owning the bus and wanting to keep ownership by keeping its **REQ#-A** asserted. At clock 2, initiator B requests bus ownership and the arbiter starts preemption of initiator A at clock 4 by deasserting **GNT#-A** and asserting **GNT#-B** one clock later. At clock 5, initiator A deasserts **FRAME#** because the byte count was satisfied or an ADB was reached. Also in clock 5, initiator C requests bus ownership by asserting its **REQ#-C**. The arbiter deasserts the **GNT#** to initiator B to grant ownership of the bus to initiator C. Initiator B observes that the previous owner (initiator A) deasserted **FRAME#** on clock 5, and **GNT#-B** is asserted on clock 6. Initiator B asserts **FRAME#** to start its transaction three clocks after **FRAME#** deasserted on clock 8 allowing one idle clock for the change of bus ownership. Initiator B starts a transaction in clock 8, even though **GNT#-B** is deasserted in clock 7.

If only one device requests the bus, it is recommended that the arbiter keep **GNT#** asserted to that device.

Implementation Note: Cascaded Arbiters

Cascaded arbiters (that is, arbiters provided in different components and connected together externally) are permitted only if all components of the cascade are specifically designed to support such an arrangement.

4.2. Arbitration Parking

As in conventional PCI, if no initiators request the bus, the arbiter is permitted to park the bus at any initiator to prevent the bus signals from floating. The arbiter parks the bus by asserting **GNT#** to an initiator even though its **REQ#** is not asserted.

If **GNT#** is asserted and the bus is idle for four consecutive clocks, the device must actively drive the bus (**AD[31::0]** and **C/BE[3::0]#**) no later than the sixth clock and **PAR** one clock later. (Note: Conventional PCI requires the device to drive the bus after eight clocks and recommends driving after only two to three clocks.) The device must stop driving the bus two clocks after **GNT#** is deasserted.

As in conventional PCI, if the parked initiator intends to execute a transaction, the initiator is not required to assert **REQ#**. The parked initiator must assert **REQ#** if it intends to execute more than a single transaction. Otherwise, it could lose the bus after only a single transaction. A parked PCI-X initiator is permitted to start a transaction up to two clocks after any clock in which its **GNT#** is asserted, regardless of the state of **REQ#** (the bus is idle since it is parked).

The same PCI-X rules apply for deasserting **GNT#** after a bus-parked condition that apply for other times. The arbiter cannot assert **GNT#** to another initiator until one clock after it deasserts **GNT#** to the parked initiator. There is only one clock reserved for bus turn-around when the bus transitions from a parked initiator to an active initiator, as shown in Figure 4-2.

Given the above, the minimum arbitration latency (that is, the delay from **REQ#** asserted to **GNT#** asserted) achievable from a PCI-X arbiter on an idle PCI bus is as follows:

1. Parked: zero clocks for parked agents, three clocks for others
2. Not Parked: two clocks for every agent

The arbiter must park the bus on a device that is capable of being an initiator. Target-only devices that do not use Split Transactions are not required to implement the **GNT#** pin or to be able to drive all the bus signals. The arbiter is permitted to assume that the device is capable of being an initiator if the device ever asserts its **REQ#** pin.

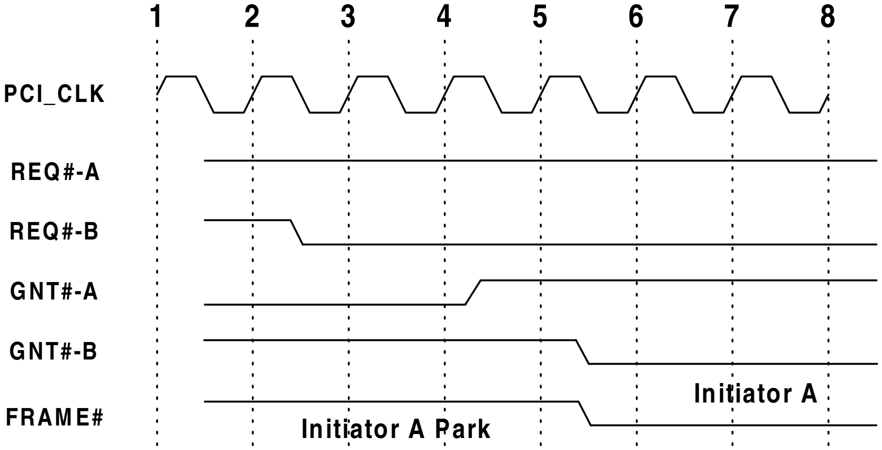


Figure 4-2: Initiating a Transaction While the Bus Is Parked

4.3. Arbiter Coordination with the PCI Hot-Plug Controller

PCI HP 1.0 requires the Hot-Plug Controller to protect other devices and transactions when a device is being hot-inserted or hot-removed. One of the things that most Hot-Plug Controllers do to provide this protection is to prevent other transactions from running on the bus during hot-plug operations.

The Hot-Plug Controller in a PCI-X system is not permitted to execute any transactions on the bus at the rising edge of **RST#** after a device has been hot-inserted. (The PCI-X initialization pattern requires the bus to be idle. See Section 6.2.) The arbiter in a PCI-X system that supports PCI hot-plug must coordinate with the Hot-Plug Controller to allow it to keep the bus in the idle state and to drive the PCI -X initialization pattern as required during hot-plug operations.

4.4. Latency Timer

The Latency Timer operation in PCI-X mode differs from its operation in conventional PCI mode in two areas, the default value in PCI-X mode and the restrictions on ending the current transaction.

The default value loaded into the Latency Timer register in the Type 00h and Type 01h Configuration Space header (offset 0Dh) in PCI-X mode is 64 (rather than 0 in conventional mode). If the PCI-X initialization pattern indicates that the device is to enter PCI-X mode at the rising edge of **RST#**, the Latency Timer register is initialized to its PCI-X value. Otherwise, it is initialized to its conventional PCI value. The PCI-X value permits PCI-X initiators to transfer data between multiple ADBs under heavy traffic conditions and low target initial latencies. This value provides good bus efficiency, effective sharing of the bus, and reasonable arbitration latencies in most cases. Configuration software is discouraged from changing the Latency Timer from its default value in PCI-X mode without a good understanding of the needs of each device in the system and the effects such a change has on all devices.

If **GNT#** is deasserted when the Latency Timer expires, the device is required to disconnect the current transaction as soon as possible. In most cases, this means the initiator disconnects on the next ADB. However, if the Latency Timer expires during a burst transaction less than four data phases from the ADB, the initiator does not have enough time to deassert **FRAME#** for this ADB. In such cases, the initiator continues past this ADB and disconnects on the next one.

5. Error Functions

The following PCI-X error detection and response functions are the same as conventional PCI. (Some new PCI-X requirements that are closely related to these conventional PCI requirements are noted in parentheses.)

- All devices generate parity. Parity checking is required except for system-board-only devices and devices that never contain any data that represents permanent or residual system or application state (e.g., audio or video output devices).
- Even parity is used. There are an even number of ones in **AD[63::32]**, **C/BE[7::4]#**, and **PAR64** for 64-bit addresses and data transfers and in **AD[31::00]**, **C/BE[3::0]#**, and **PAR**.
- The device driving the **AD** bus also drives **PAR64** (for 64-bit addresses and data transfers) and **PAR**.
- Parity is generated for each of the following:
  - Address phases (single or dual). (PCI-X devices also generate parity for the attribute phase.)
  - All clocks of all data phases of write transactions (i.e., including target initial wait states). (PCI-X devices also drive parity on Split Completions the same as write transactions.)
  - All data-transfer clocks of all data phases of read transactions (i.e., excluding target initial wait states).
- The parity bit lags the **AD** bus by one clock. During a write transaction, the initiator drives **PAR64** (if initiating as a 64-bit device) and **PAR** on clock N+1 for the write data and the byte enables it drives on clock N. (In PCI-X read data phases, the appropriate clock for the **C/BE#** bus is one clock earlier than conventional PCI.)
- Address parity errors cause **SERR#** (if enabled).
- Initiators and targets set bits in the Status register when a parity error occurs.
- **SERR#** is an open-drain signal. Whenever it is asserted, it is actively driven low synchronously with the **CLK** for one clock period. It is pulled up by a resistor supplied by the system, so its rise time is permitted to span more than one clock period.
- A device that asserts **SERR#** also sets the Signaled System Error bit in the Status register.

The following items are different for PCI-X:

- Attribute parity errors cause **SERR#** (if enabled).
- No parity is generated or checked for the target response phase.
- During a read transaction, the target drives parity on clock N+1 for the read data it drove on clock N and the byte enables driven by the initiator on clock N-1.
- The target of a write or Split Completion transaction must not check data parity while it is inserting target initial wait states. It must check parity only on the data-transfer clock.
- **PERR#** is asserted after a data parity error one clock later than conventional PCI.

- The requester sets the Master Data Parity Error bit to record a data parity error. For Split Completions, this is the target rather than the initiator. See Section 5.4.1 and Section 5.4.6.
- The Data Parity Error Recover Enable bit in the PCI-X Command register enables the system to recover from some data parity errors. See Sections 5.4.1.1 and 5.4.1.2.

The following sections provide additional details.

5.1. Parity Generation

The requirements for generation of parity in PCI-X devices are the same as for conventional devices, except for timing, which is described below.

5.2. Parity Checking

Targets check parity for address and attribute phases. The device receiving the data checks parity in data phases. No device checks for parity errors on the **AD** and **C/BE#** buses in the clock following the attribute phase since **PAR64** and **PAR** are not valid.

PCI-X devices treat parity errors in address and attribute phases the same as conventional PCI devices treat address parity errors. The device asserts **SERR#** and sets the Detected Parity Error bit in the Status register, as specified in PCI 2.2 for address parity errors.

Data parity error checking and signaling for PCI-X devices are enabled by the same Control register bits as conventional devices. For those devices that check parity, the following participants in a PCI-X transaction are considered to receive data and, therefore, check data parity:

- The initiator of a read transaction that is completed immediately or is terminated with Split Response.
- The target of a write that is completed immediately.
- The target (completer or PCI-X bridge) of a write that is terminated with Split Response.
- The target (requester or PCI-X bridge) of a Split Completion. The target checks parity on the read data or Split Completion Message.

5.3. Parity Timing

On any given address and attribute phase, **PAR64** (for 64-bit devices) and **PAR** are driven by the initiator. On any given data phase, **PAR64** (for 64-bit transfers) and **PAR** are driven by the device that drives the data. In all cases, the parity bits lag the corresponding address or data by one clock.

Parity checking occurs in the clock after **PAR64** and **PAR** are valid. If a parity error is detected in a data phase, **PERR#** is asserted (if enabled) two clocks after **PAR64** and **PAR** are valid.

Figure 5-1 illustrates parity generation and checking on a write or Split Completion transaction, and Figure 5-2 illustrates parity generation and checking on a read transaction. For the write or Split Completion transaction in Figure 5-1, the initiator drives **PAR64** (for a 64-bit device) and **PAR** for the address phase on clock 4 and for the attribute phase on clock 5. For these transactions, the response phase in clock 5 carries no parity and, therefore, must not be checked by the target device. The data phases follow with the parity for each data transfer lagging by one clock as shown in clocks 7, 8, 9, and 10. If the target detects a data parity error, it asserts **PERR#** two clocks after the **PAR64** and **PAR** are valid as shown in clocks 9, 10, 11, and 12.

The initiator generates parity for write or Split Completion data phases even if the target inserts initial wait states, which requires the initiator to toggle between two data patterns (see Section 2.9.2). The target of a write or Split Completion transaction must not check data parity while it is inserting target initial wait states. It must check parity only on the data-transfer clock. (If the burst is only a single data phase long, the toggling data is not part of the transaction.)

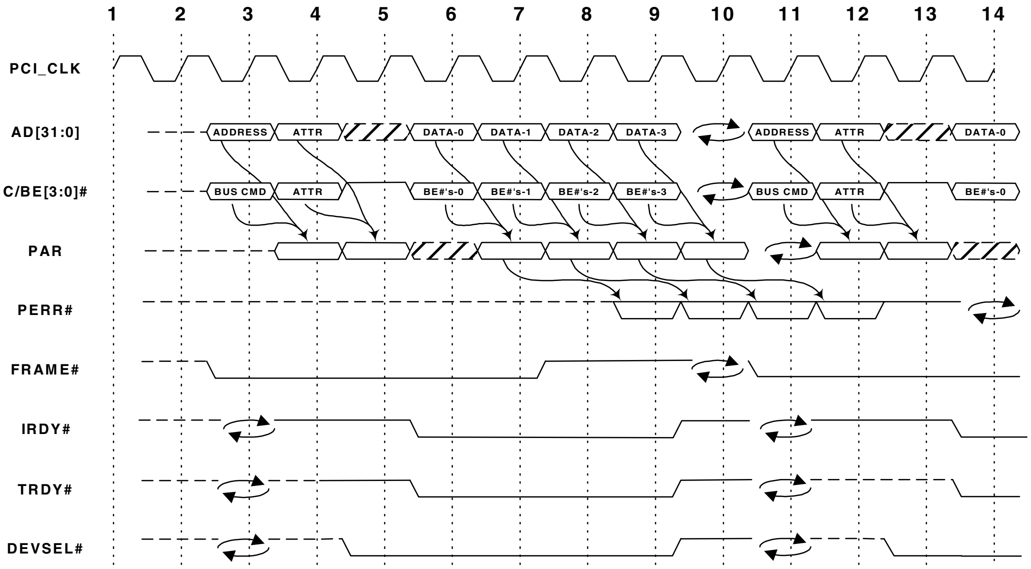


Figure 5-1: Burst Write or Split Completion Transaction Parity Operation

The read transaction illustrated in Figure 5-2 begins identically to the write transaction with the initiator driving **PAR64** (if a 64-bit device) and **PAR** for the address phase on clock 4 and for the attribute phase on clock 5. As in the write and Split Completion transaction, no parity is generated for the response phase, which is also the turn-around cycle for the read transaction. Parity generation for the data phase, however, is different for read transactions. During a read transaction, the target drives **PAR64** (if a 64-bit transfer) and **PAR** on clock N+1 for the read data it drove on clock N and the byte enables driven by the initiator on clock N-1. The **C/BE#** bus for burst reads is included in the parity calculation for consistency with conventional PCI, even though the bus is reserved and driven high by the initiator after the attribute phase. This is illustrated with the parity at clock 7 using the **AD** bus driven by the target on clock 6 (DATA-0) and the byte enables driven by the initiator on clock 5. Notice that the byte enables driven by the initiator on clock 9 are not used in the transaction and are not protected by any parity.

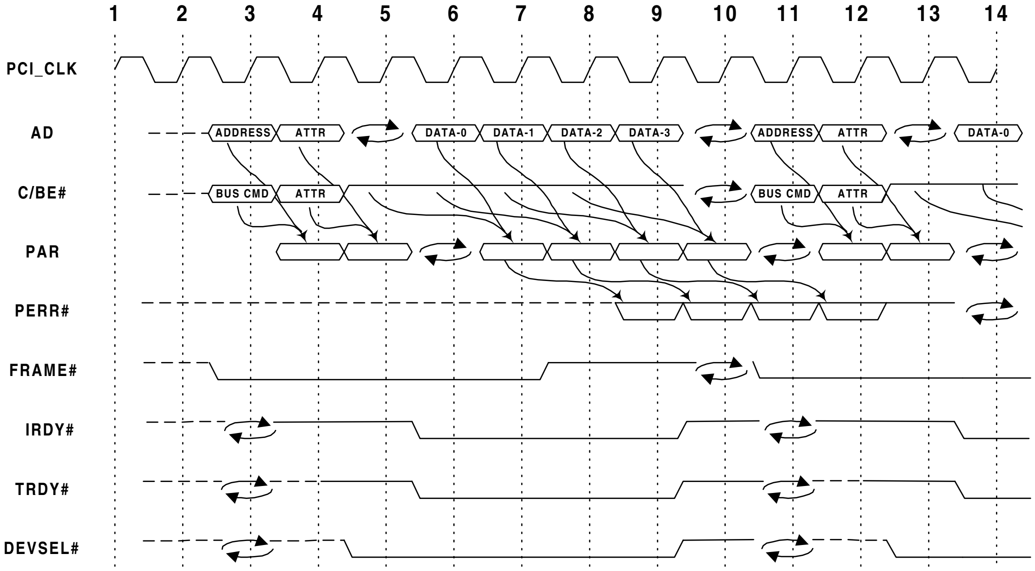


Figure 5-2: Burst Read Transaction Parity Operation

Figure 5-3 and Figure 5-4 illustrate how the **C/BE[3::0]#** bus is included in parity checking for DWORD read transactions with no wait states and **DEVSEL#** decode speed A and B respectively. The **C/BE[3::0]#** bus for DWORD reads is included in the parity calculation for consistency with conventional PCI, even though the bus is reserved and driven high by the initiator after the attribute phase.

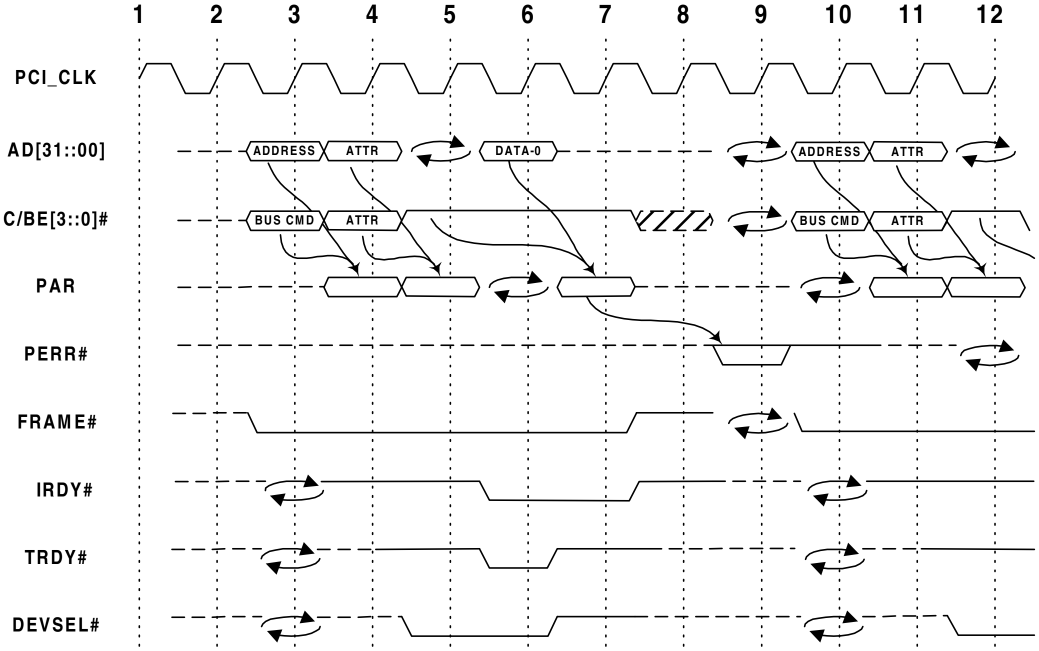


Figure 5-3: DWORD Read Parity Operation with DEVSEL# Decode A and No Initial Wait States

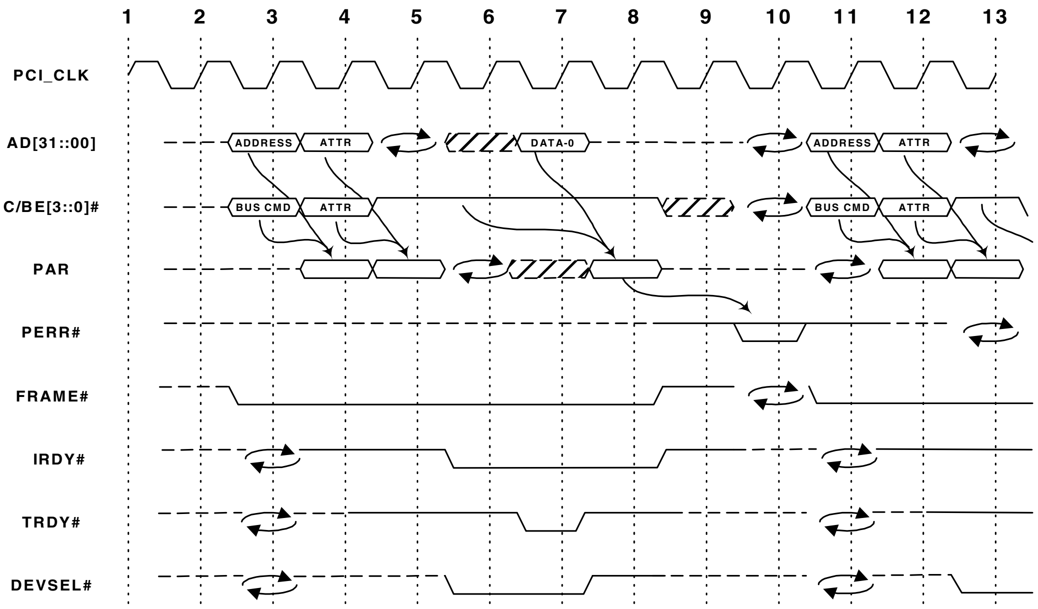


Figure 5-4: DWORD Read Parity Operation with DEVSEL# Decode B and No Initial Wait States

Figure 5-5 and Figure 5-6 illustrate parity generation and checking on a DWORD write transaction with **DEVSEL#** decode speed A and B respectively. As in a burst write, the initiator drives **PAR** for the address phase on clock 4 and for the attribute phase on clock 5. No parity is generated for the response phase in clock 6, so none is checked. Data-phase parity includes the **C/BE#** bus, even though that bus is driven high throughout the data phase. Data parity must be generated for each clock that the data is required to be stable, beginning with clock 7. If the data is required to be stable for additional clocks because of slower **DEVSEL#** timing as in Figure 5-6, or because of target initial wait states, **PAR** is also required to be stable for the same number of additional clocks. The target checks **PAR** only one clock after the data-transfer clock and asserts **PERR#** two clocks after that, if an error is detected.

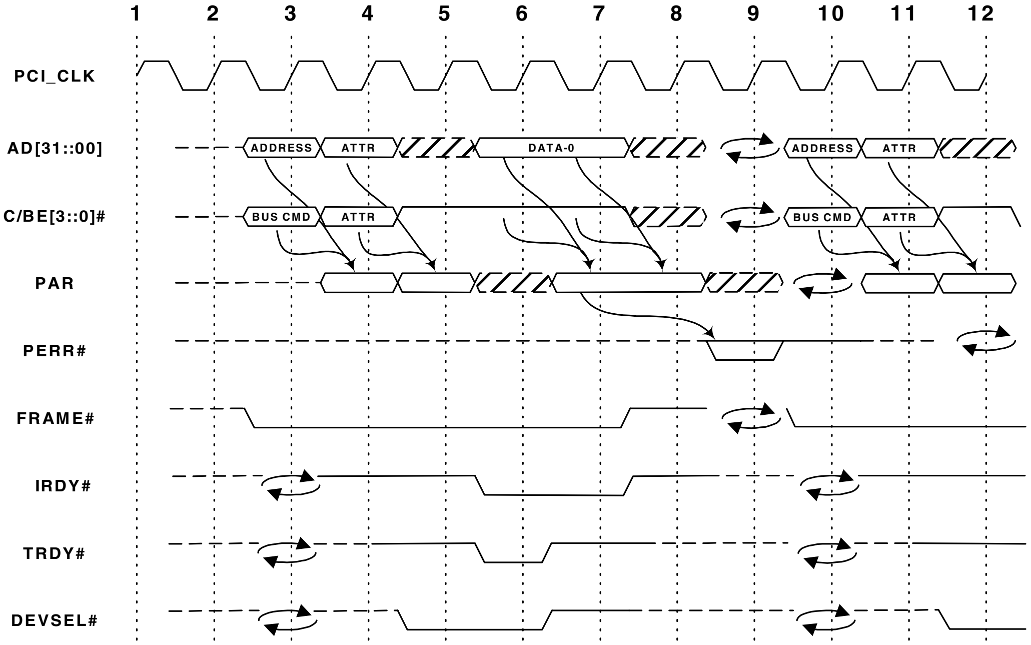


Figure 5-5: DWORD Write Parity Operation with DEVSEL# Decode A and No Initial Wait States

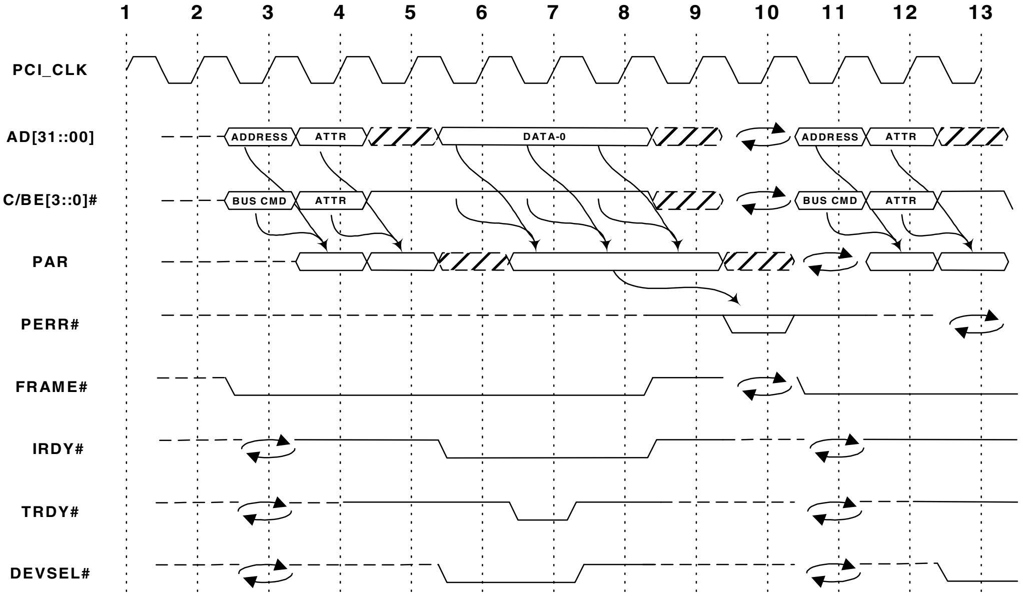


Figure 5-6: DWORD Write Parity Operation with DEVSEL# Decode B and No Initial Wait States

5.4. Error Handling and Fault Tolerance

This section describes the requirements for PCI-X devices and their device drivers when an error occurs.

5.4.1. Data Parity Errors

If parity checking is enabled and a device receiving data detects a data parity error, it must assert **PERR#** on the second clock after **PAR64** and **PAR** are driven (one clock later than conventional PCI) as illustrated in Figure 5-1 and Figure 5-2.

The Master Data Parity Error (bit 8) in the conventional Status register in a PCI-X device is set under slightly different conditions than conventional PCI devices. It is subject to the same enabling bits in the Control registers as conventional PCI but is set in either the initiator or target under the following conditions:

- The initiator (requester or PCI-X bridge) of a read transaction that is completed immediately calculates a data parity error.
- The initiator (requester or PCI-X bridge) of a read transaction that is terminated with Split Response calculates a data parity error in the Split Response.
- The initiator (requester or PCI-X bridge) of a write that is completed immediately observes **PERR#** asserted three clocks after one or more of its data phases.
- The initiator (requester or PCI-X bridge) of a write that is terminated with Split Response observes **PERR#** asserted three clocks after the data phase.
- The target (requester or PCI-X bridge) of a Split Completion calculates a data parity error in either read data or a Split Completion Message.
- The target (requester or PCI-X bridge) receives a Split Completion Message that indicates a data parity error occurred on one of this device's non-posted write transactions (see Section 5.4.6).

The Detected Parity Error (bit 15) bits in the Status register is set by the device whose parity checking logic calculated the data parity error, the same as for conventional PCI.

PCI-X error handling builds on conventional PCI error functions to enable recovery from a broader range of errors. All PCI-X devices in combination with system software and their device drivers are required either to recover from a data parity error or to assert **SERR#**. By requiring the device and/or software to recover from the error or to assert **SERR#**, the system is freed from the assumption that data parity error conditions are always catastrophic to the system.

Devices are allowed to attempt to recover from a data parity error only under control of the software. Only the device driver has the necessary information to determine what is appropriate and necessary to repeat and what is not. For example, a read transaction from a location that has side effects cannot be repeated by itself and get the same results. The following sections list the requirements for devices and software drivers in PCI-X mode that are designed to support data parity error recovery and for devices and software drivers that are not (and assert **SERR#**).

The requirements for a PCI-X device that is operating in conventional PCI mode are governed by PCI 2.2.

5.4.1.1. Devices and Software Drivers that Support Data Parity Error Recovery

When **RST#** is asserted, all PCI-X devices clear the Data Parity Error Recovery Enable bit in the PCI-X Command register. If the operating system loads a software device driver that supports data parity error recovery, either the device driver or system software is required to set the bit as specified by the operating system vendor.

The device that originated the Sequence that experienced a data parity error (that is, the device that sets the Master Data Parity Error bit as described in Section 5.4.1) is required to notify its device controlling software. The device is allowed to attempt to recover from the error only under control of the software.

The operating system vendor must specify the requirements of the controlling software after a data parity error. The operating system commonly provides an API for the device driver to report such errors to the operating system. In some cases, the operating system vendor specifies that the device driver must perform actions to recover from the error. For example, the following non-exhaustive list illustrates what the operating system vendor might require the device driver to do:

- Reschedule the failing transaction
- Notify the user of the failing transaction
- Reinitialize the card and continue
- Take the card off-line
- Shut down the operating system

The operating system vendor must also specify how the **PERR#** status bits, Master Data Parity Error and Data Parity Error Detected, in device and bridge Status registers are to be treated by system software after recovery from a data parity error. If these bits are to be cleared, the operating system vendor must specify what software is responsible for clearing them.

5.4.1.2. Devices or Software Drivers That Do Not Support Data Parity Error Recovery

When **RST#** is asserted, all PCI-X devices clear the Data Parity Error Recovery Enable bit in the PCI-X Command register. System software that does not attempt to recover from data parity errors leaves this bit in its default state.

In addition to the requirements specified in Section 5.4.1, a PCI-X device asserts **SERR#** (if enabled) after a data parity error if both of the following are true:

- The Data Parity Error Recovery Enable bit in the PCI-X Command register is cleared.
- A data parity error occurred that caused the device to set the Master Data Parity Error bit (see Section 5.4.1).

As in conventional PCI, a system-specific service routine is activated as a result of the **SERR#** assertion. The **SERR#** service routine is allowed to treat an **SERR#** assertion as a catastrophic exception that will ultimately result in a system halt.

Implementation Note: Alternative Platform-Specific Recovery Routines for Data Parity Errors

Some systems provide platform-specific routines to recover from data parity errors from a selected list of devices. For these PCI-X systems, the following recommendation is provided.

Systems that support **PERR#** recovery using platform-specific recovery routines provide system-specific ROM software that at power-up sets the Data Parity Error Recovery Enable bit in the PCI-X Command register for the selected set of PCI-X devices (assumed to be supported on all PCI-X bus segments). PCI-X devices with this bit set do not assert **SERR#** as a result of a data parity error but assert **PERR#** as conventional PCI devices do. The platform-specific service routines execute the platform-specific data parity error recovery routines before returning control to the operating system.

5.4.1.3. Data Parity Errors in Split Response for Read Transactions

If an initiator (requester or bridge) calculates a data parity error when the target (completer or bridge) signals Split Response for a read transaction, the initiator must record the error as described in Section 5.4.1. Furthermore, if the initiator is enabled to assert **PERR#** and does not support recovery from data parity errors (i.e., Data Parity Error Recover Enable bit in the PCI-X Command register is cleared), the device must also assert **SERR#** (if enabled).

If the device supports recovery from data parity errors and the Data Parity Error Recovery bit is set, the device is permitted to report and recover from errors in the Split Response differently from errors in the corresponding Split Completion.

Implementation Note: Recovering from Data Parity Errors in Split Response

A data parity error in a Split Response indicates the existence of a serious problem in the system, but does not imply that the read data in the subsequent Split Completion is erroneous. It is possible that the read data in the subsequent Split Completion will arrive without error. However, most causes of parity errors affect more than a single transaction, so implementation of significant hardware and software to recover from this special case is probably not justified.

Devices may optionally report the occurrence of a data parity error in a Split Response by setting a unique status bit in a device-specific register. If the device is enabled by its driver and system software to recover from data parity errors, the device might recover from errors in a Split Response differently from errors in the Split Completion. For example, an error in a Split Response alone might not require the transaction to be repeated, whereas an error in a Split Completion would require one or more transactions to be repeated.

If a device issues an interrupt as a result of a data parity error on a Split Response, it is recommended that the interrupt not be issued until the Sequence is complete. Otherwise, it would theoretically be possible for the interrupt service routine to execute before the last data of the Split Completion arrives at the requester. Although this is highly unlikely because Split Transactions normally complete in much less time than is required for the CPU to acknowledge an interrupt, the potential problem is avoided if the interrupt is delayed until the completion of the Sequence.

5.4.2. Target-Abort and Master-Abort Exceptions

The initiator of a transaction other than a Split Completion is required to notify its device driver via interrupt or other suitable means whenever a Target-Abort or Master-Abort occurs (except those cases defined in PCI 2.2 in which a Master-Abort does not indicate an error condition, like configuration or Special Cycle transactions). If notification of the device driver is not possible, the initiator must assert **SERR#** (if enabled) and update its Status register. See Section 5.4.4 for errors that occur on Split Completion transactions.

5.4.3. Address and Attribute Parity Errors

A PCI-X target that detects a parity error in the address or attribute phase of any transaction must assert **SERR#** and set status bits as defined for address parity errors in PCI 2.2. If the device's address decode logic indicates that the device is selected, the device has the same options for ignoring (Master-Abort) or executing the transaction as conventional PCI devices with address parity errors. As described in PCI 2.2, if the device asserts **DEVSEL#** prior to detecting a parity error in the address or attribute phase, the device has the option either to complete the transaction as if no error occurred or to signal Target-Abort (even if the transaction is a Split Completion). If the error occurred in the attribute phase and the device terminates the transaction with Split Response, the device must discard the transaction and assert **SERR#** (if enabled). (An error in the attribute phase means it is unlikely that a Split Completion could be routed properly back to the requester.)

5.4.4. Split Transaction Errors

Errors are possible in three different phases of a Split Transaction: during the Split Request, during the execution of the request by the completer, and during the Split Completion.

The completer is permitted to signal Split Response to a DWORD write either before or after it checks for data parity errors. (See Section 8.7.1.2 for the requirements for a bridge.) If the completer detects the data parity error on a DWORD write transaction and signals Data Transfer (an Immediate Transaction), the completer asserts **PERR#**. If the completer detects a data parity error on a DWORD write transaction and signals Split Response, the completer asserts **PERR#** and generates the appropriate Split Completion Message (see Section 2.10.6.2). All other error conditions for Split Requests are handled by the initiator as they would be for an Immediate Transaction.

Abnormal conditions are also possible in the second phase of a Split Transaction after the completer has terminated a transaction with Split Response termination. If such a condition occurs, the completer is required to notify the requester of the abnormal condition by sending a Split Completion Message as described in Section 2.10.6.

A variety of abnormal conditions are possible during the third phase of the Split Transaction, which is the Split Completion transaction. If the Split Completion transaction completes with either Master-Abort or Target-Abort, the requester (or intervening bridge) is indicating a failure condition that prevents it from accepting the completion it requested. In this case if the Split Request is a write or if it addresses a location that has no read side effects, the completer must decide whether the error causes a risk to the integrity of the system. If the completer decides that the error does not cause a serious risk to the integrity of the system, the completer discards the Split Completion and takes no further action (not set the Split Completion Discarded bit and not assert **SERR#**, however, the completer is permitted to record the condition for diagnostic purposes using device-specific means). If the completer decides that the error causes a serious risk to the integrity of the system, the completer must discard the Split Completion, set the Split Completion Discarded bit in the PCI-X Status register, and assert **SERR#** (if enabled). If the Split Request is a read and the location has read side effects, the completer must discard the Split Completion, set the Split Completion Discarded bit in the PCI-X Status register, and assert **SERR#** (if enabled). In none of the above cases does the completer set the Received Master-Abort or Received Target-Abort bits in the Status register, since the completer is not the original initiator of the Sequence. The completer behaves as an initiator for setting all other Status register bits.

Implementation Note: Abnormal Termination of Split Completion Transactions

A Split Completion normally terminates with Data Transfer. A properly functioning requester in a properly functioning system takes all the data indicated by the byte count of the original Split Request without signaling Target-Abort or allowing a Master-Abort.

For completeness, the completer's response to the abnormal terminations, Master-Abort and Target-Abort, is specified. The transaction would terminate with Master-Abort if the requester did not recognize the Sequence ID of the Split Completion. This can occur only under error conditions. The Split Completion would terminate with Target-Abort only if the requester encountered an internal error that prevented it from guaranteeing the integrity of data in the system. (See Section 2.10.5.)

If the requester detects a data parity error during a Split Completion, it asserts **PERR#** and sets bit 15 (Detected Parity Error) in the Status register. The requester also sets bit 8 (Master Data Parity Error) in the Status register, because it was the original initiator of the Sequence (even though the requester is the target of the Split Completion).

5.4.5. Corrupted or Unexpected Split Completions

Several scenarios are possible if a Split Completion becomes corrupted. For example, if the Requester ID of a Split Completion becomes corrupted, it is possible that it matches that of another device in the system. Furthermore, if the Requester ID of a Split Completion is accurate but the Tag becomes corrupted, the requester is unable to match the Split Completion to its Split Request. Also, if the byte count of a Split Completion becomes corrupted, the requester either receives more data than the original request, or not enough.

A device may optionally assert **DEVSEL#** if the Requester ID matches that of the device, but the Tag does not match any outstanding requests from this device. That is, a device is permitted to ignore the Tag when deciding whether to assert **DEVSEL#**. Alternatively, the device may choose to ignore the corrupted transaction (not assert **DEVSEL#**) if the Tag does not match any outstanding request from this device. For example, a device that never initiates transactions, or that has no transactions outstanding at the moment, might choose this option.

If a requester asserts **DEVSEL#** for a Split Completion, but the Tag does not match any that the requester currently has outstanding, the transaction is identified as an unexpected Split Completion. The requester is required to accept the Split Completion transaction in its entirety and discard the data. In addition to discarding the data, the device must set the Unexpected Split Completion status bit in the PCI-X Status register.

Valid values for the Lower Address field in the Split Completion Address and byte count in the Completer Attributes are specified in Sections 2.10.2, 2.10.3, and 2.10.4. If the Sequence ID of a Split Completion matches that of an outstanding request, but the Lower

Address field or the byte count is not valid, the requester is required to accept the Split Completion transaction in its entirety (as determined by the invalid address and byte count). A corrupted address or byte count in a Split Completion does not justify signaling Target-Abort, even if the invalid byte count is larger than the byte count of the original request. The requester is not required to detect this case. However, if it does, it discards the entire Split Completion and sets the Unexpected Split Completion status bit in the PCI-X Status register.

Other than setting the Unexpected Split Completion status bit, the method by which a device reports a corrupted or unexpected Split Completion to its device driver is not specified.

5.4.6. Reporting Split Completion Error Messages

A device (other than a bridge) that receives a Split Completion Message with the Split Completion Error attribute bit set must set the Received Split Completion Error Message bit in its PCI-X Status register.

A device (requester or bridge) that receives a Split Completion Message that reports an error condition that corresponds to non-posted write data parity errors, Master-Abort conditions, and Target-Abort conditions must set the corresponding bit in the conventional Status register (or Secondary Status register in a bridge). Other Split Completion error messages are reported via device-specific means that are beyond the scope of the PCI-X definition. Table 5-1shows what bits in the Status register or Secondary Status register must be set when a requester or bridge receives these messages.

Table 5-1: Reporting the Receipt of Split Completion Error Messages

Message Class	Message Index	Message Description	Bits Set in Status Register or Secondary Status Register
1	00h	Master-Abort	Received Master-Abort
1	01h	Target-Abort	Received Target-Abort
1	02h	Write Data Parity Error	Master Data Parity Error
2	00h	Byte Count Out of Range	none
2	01h	Split Write Data Parity Error	Master Data Parity Error
2	8Xh	Device-Specific	none

Furthermore, if a bridge forwards upstream a Split Completion Message indicating the occurrence of a Target-Abort (Class 1, Index 01h), it must set the Signaled Target-Abort bit in the Status register. If a bridge forwards downstream such a transaction, it must set the Signaled Target-Abort bit in the Secondary Status register.

Implementation Note: Setting Status Bits when Forwarding Split Completion Error Messages

Split Completion Messages that report the occurrence of errors that correspond to equivalent errors in conventional PCI mode have the same effect on a device's Status register as if the error had occurred on an Immediate Transaction. Furthermore, these messages have the same effect on a bridge's Status register and Secondary Status register as a similar error would if the bridge interfaces were operating in conventional mode. This means, for example, that a requester that receives a Split Completion Message indicating the detection of a data parity error on a non-posted write transaction must set the Master Data Parity Error bit as if the error occurred on the immediate completion of the transaction and was indicated by the assertion of **PERR#** by the target. Also, a bridge that forwards this message must set its Master Data Parity Error bit on the interface that receives the message. This corresponds to the requirement in Bridge 1.1 for a device that forwards a non-posted write as a Delayed Transaction. If such a bridge detects a data parity error on the destination bus, the bridge must set the Master Data Parity Error bit for that bus and must return the error to the requester by asserting **PERR#** on the requester-side interface when the requester repeats the transaction. (If the requester-side bus were in conventional PCI mode the assertion of **PERR#** would cause the requester to set its Master Data Parity Error bit.)

By setting the error status bits the same way regardless of whether the buses are in PCI-X mode and use Split Transactions or the buses are in conventional mode and use Delayed Transactions, the error analysis software can implement a single algorithm to identify the devices involved with a data parity error on a non-posted write transaction.

6. System Interoperability and Initialization

6.1. Interoperability

6.1.1. Device and Add-In Card Interoperability Requirements

PCI-X devices must meet the 33 MHz protocol and timing requirements of PCI 2.2 when operating in that mode. PCI-X devices may optionally meet the 66 MHz timing requirements of PCI 2.2.

The protocol and electrical requirements for PCI-X 133 devices and PCI-X 66 devices is identical except for the maximum operating frequency. The operating frequency range for PCI-X 133 devices is a superset of the range for PCI -X 66 devices (see Section 9.4.1).

Implementation Note: Minimum Bandwidth for Application

PCI-X devices are required to operate when installed in any PCI-X or conventional PCI bus down to conventional 33 MHz mode, 32-bits wide. However, some applications require a minimum bus bandwidth to perform their intended function. In some applications, it may be clear that some of the slower PCI modes or frequencies do not provide this bandwidth. At the middle frequencies, the application may have sufficient bandwidth only if there is no conflict with other devices.

If the slower PCI modes or frequencies clearly do not provide the minimum bandwidth required by the application, the device driver should detect these cases by determining in what mode the device initialized and report any obvious problems to the user. If the device requires restrictions on other devices when used in some PCI or PCI-X modes and frequencies, the device vendor should provide guidelines to the user as to what other devices should be installed on the same bus.

PCI-X devices use 3.3V I/O signaling levels defined in Section 9.1 (compatible with that defined in PCI 2.2) when operating in PCI-X mode. PCI-X devices optionally also support the 5V I/O signaling levels defined in PCI 2.2 when operating in 33 MHz conventional PCI mode. PCI-X add-in cards must be keyed either for 3.3V I/O or Universal I/O as defined in PCI 2.2.

6.1.2. System Interoperability Requirement

If a bus includes at least one 33 MHz conventional device, the bus must operate in conventional 33 MHz mode. If only conventional 66 MHz devices are present in slots on the bus, a PCI bus optionally operates either in conventional 66 MHz mode or conventional 33 MHz mode. (PCI 2.2 permits 33-MHz-mode buses to operate at any frequency below 33 MHz and 66-MHz-mode buses to operate at frequencies between 33 and 66 MHz, if required because of bus loading.)

If a bus includes only PCI-X devices, the bus operates in PCI-X mode. If the bus includes at least one PCI-X 66 device, the maximum clock frequency is 66 MHz. If the bus contains only PCI-X 133 devices, the maximum clock frequency is 133 MHz. PCI-X systems are permitted to limit bus frequency to a value lower than the nominal down to the minimum specified in Section 9.4.1. This is generally done to support higher loading on the bus. For example, a bus with two expansion slots would typically operate at 100 MHz.

6.1.3. Interoperability Matrix

Figure 6-1 shows the interoperability matrix for variations of system and add-in card operation mode and frequency capability.

Systems		Conventional PCI Cards			PCI-X Cards[1]	
		33 MHz (5V I/O)	33 MHz (3.3V I/O or Universal)	66 MHz (3.3V I/O or Universal)	66 MHz (3.3V I/O or Universal)	133 MHz (3.3V I/O or Universal)
Conventional System	33 MHz (5V I/O) 10 ns[2,3]	33 (5V I/O)	33 (5V I/O)	33 (5V I/O)	33 (5V I/O)	33 (5V I/O)
	33 MHz 10 ns[2]		33	33	33	33
	66 MHz 5 ns[2]		33	66	a) 33[5] b) 66	a) 33[5] b) 66
PCI-X System	66 MHz 9 ns[2]		33	a) 33[4,6] b) 66	66	66
	100 MHz 4.5 ns[2]		33	a) 33[4] b) 66	66	100
	133 MHz 2 ns[2]		33	a) 33[4] b) 66	66	133

Figure 6-1: Interoperability Matrix for Frequency and I/O Voltage

Legend:

xx	Conventional PCI system or add-in card operating in conventional mode xx = nominal clock frequency in MHz
xx	PCI-X system and add-in card operating in PCI-X mode xx = nominal clock frequency in MHz
xx	Most popular cases

Notes:

1. Unless otherwise specified, all cases use an I/O signaling voltage of 3.3 V.
2. Time indicates maximum value of T_{prop} for the system shown. 33 MHz and 66 MHz values are taken from PCI 2.2. PCI-X values come from Table 9-11.
3. Most systems shipped prior to the development of the PCI-X definition fall into this row. 5V I/O and Universal I/O cards work here but not 3.3V I/O cards.
4. PCI-X systems optionally operate in 33 MHz mode or 66 MHz mode when only 66 MHz conventional PCI cards are installed.
5. PCI-X devices must support conventional 33 MHz timing and may optionally support conventional 66 MHz timing.
6. A bus designed for PCI-X 66 MHz operation generally has too many loads to support conventional 66 MHz timing.

6.2. Initialization Requirements

PCI-X systems inform devices of the width of the bus by the state of **REQ64#** at the rising edge of **RST#** as specified in PCI 2.2.

Add-in cards indicate whether they support PCI-X, and if so which frequency, by the way they connect one pin called **PCIXCAP**. If the card's maximum frequency is 133 MHz, it leaves this pin unconnected (except for a decoupling capacitor specified in Section 9.10). If the card's maximum frequency is 66 MHz, it connects **PCIXCAP** to ground through a resistor (and decoupling capacitor) specified in Section 9.10. Conventional cards connect this pin to ground. See Section 14 for recommendations for circuits to detect the types of cards connected to **PCIXCAP**.

An add-in card indicates its capability with one of the combinations of the **M66EN** and **PCIXCAP** pins listed in Table 6-1.

Table 6-1: M66EN and PCIXCAP Encoding

M66EN	PCIXCAP	Conventional Device Frequency Capability	PCI-X Device Frequency Capability
Ground	Ground	33 MHz	Not capable
Not connected	Ground	66 MHz	Not capable
Ground	Pull-down	33 MHz	PCI-X 66 MHz
Not connected	Pull-down	66 MHz	PCI-X 66 MHz
Ground	Not connected	33 MHz	PCI-X 133 MHz
Not connected	Not connected	66 MHz	PCI-X 133 MHz

The source bridge places all devices on that segment in PCI-X mode or conventional mode by driving a particular combination of control signals on the bus at the rising edge of **RST#**. If **FRAME#** is deasserted and **IRDY#** is deasserted (i.e., the bus is idle) and one or more of **DEVSEL#**, **STOP#**, and **TRDY#** are asserted at the rising edge of **RST#**, the device enters PCI-X mode (see Table 6-2). Otherwise, the device enters conventional PCI mode at the rising edge of **RST#**. The combination of **FRAME#**, **IRDY#**, **DEVSEL#**, **STOP#**, and **TRDY#** at the rising edge of **RST#** is called the PCI-X initialization pattern.

Table 6-2: PCI-X Initialization Pattern

DEVSEL#	STOP#	TRDY#	Mode	Max Clock Period (ns)	Min Clock Period (ns)	Min Clock Freq (MHz) (ref)	Max Clock Freq (MHz) (ref)
Deasserted	Deasserted	Deasserted	Conventional 33	∞	30	0	33
			Conventional 66	30	15	33	66
Deasserted	Deasserted	Asserted	PCI-X	20	15	50	66
Deasserted	Asserted	Deasserted	PCI-X	15	10	66	100
Deasserted	Asserted	Asserted	PCI-X	10	7.5	100	133
Asserted	Deasserted	Deasserted	PCI-X	Reserved	Reserved	Reserved	Reserved
Asserted	Deasserted	Asserted	PCI-X	Reserved	Reserved	Reserved	Reserved
Asserted	Asserted	Deasserted	PCI-X	Reserved	Reserved	Reserved	Reserved
Asserted	Asserted	Asserted	PCI-X	Reserved	Reserved	Reserved	Reserved

The PCI-X initialization pattern also informs the devices of the frequency range of the clock as shown in Table 6-2. Systems with a nominal clock frequency of 66 MHz indicate the 50-66 MHz range, and systems with a nominal clock frequency of 100 MHz indicate the 66-100 MHz range.

The system must not generate reserved PCI-X initialization patterns. Behavior of devices is not specified if they are initialized with a reserved PCI-X initialization pattern.

The remainder of this section explains the system initialization requirements both for devices and the system and explains interoperability requirements between PCI-X and conventional PCI devices.

6.2.1. Device and Add-in Card Initialization Requirements

PCI-X devices enter conventional or PCI-X mode based on the PCI-X initialization pattern, as defined in Table 6-2, at the rising edge of **RST#**. The device must select all state-machines, PLL lock ranges, and electrical differences between conventional and PCI-X based on this pattern at the rising edge of **RST#**. When the system powers up, **FRAME#**, **IRDY#**, **DEVSEL#**, **STOP#**, and **TRDY#** may be indeterminate while power supply voltages are rising but are stable before the last rising edge of **RST#**. Devices are permitted combinatorially to change between conventional and PCI-X mode while **RST#** is asserted as determined by the value of the PCI-X initialization pattern.

Implementation Note: Switching to PCI-X Mode

While **RST#** is asserted, the PCI-X definition requires all state machines to reset and re-lock any PLLs, if necessary, because a frequency change is possible. Figure 9-14 illustrates this mode switching reset condition and Table 9-5 lists its timing requirements.

Since the PCI clock is not required to be stable throughout the time that **RST#** is asserted, PCI-X devices must latch their mode independent of the PCI clock. Figure 6-2 illustrates an example of the logic to support this requirement. The output of this latch is the device's PCI-X mode signal. This signal is used to switch all PCI I/O buffers and PCI interface logic to support PCI-X protocol.

The figure shows a transparent latch that allows the PCI-X mode signal to pass through the latch while **RST#** is asserted. This implementation removes any critical timing issues that may arise if the signal is heavily loaded, controlling all I/O buffers, clocking logic (PLL), and state machine. The design assumes an asynchronous delay element in the PCI-X initialization pattern decode block connected to the latch input. This delay element provides ASIC register hold time after the rising edge of **RST#**.

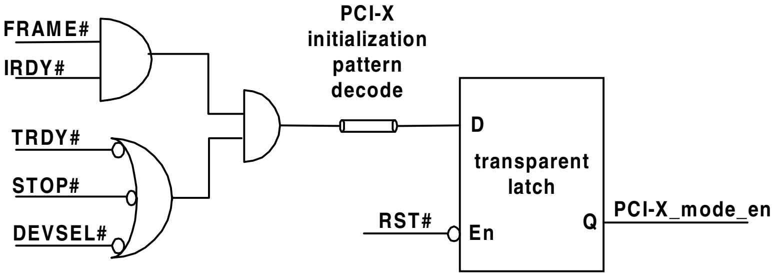


Figure 6-2: PCI-X Mode Latch

The PCI-X initialization pattern informs the device of the operating frequency range of the clock, as indicated in Table 6-2, if the bus is operating in PCI-X mode. (When in conventional mode, the device uses **M66EN** as specified in PCI 2.2 to determine operating frequency.) The device uses this information to optimize internal options that are a function of the clock frequency, e.g., **DEVSEL#** decode timing or PLL range parameters. PCI-X bridges are permitted to use this information from their primary bus to optimize the clock divider that generates the secondary clock frequency.

A device's bus interface state machines (i.e., initiator state machines, target state machines, etc.) must ignore any combination of the assertion of **DEVSEL#**, **STOP#**, and **TRDY#** while **FRAME#** and **IRDY#** are deasserted (i.e., the bus is idle). If a PCI-X device is hot-inserted onto the bus, the Hot-Plug Controller asserts some or all of these target signals when it drives the PCI-X initialization pattern on the bus to initialize the new device. Devices for which **RST#** is already deasserted (i.e., devices that are already connected to the bus) must ignore this pattern that is being applied for the benefit of the device with **RST#** asserted (i.e., the device being hot-inserted).

As in conventional PCI, if the clock frequency is higher than 33 MHz, it is guaranteed not to change (beyond the limits of Spread Spectrum Clocking specified in Section 9.4.1) except while **RST#** is asserted. If a PCI-X device uses a PLL on the input clock, that PLL cannot be enabled in conventional 33 MHz mode. It can only be enabled in PCI-X mode (as determined by the PCI-X initialization pattern at the rising edge of **RST#**) or in conventional 66 MHz mode (as determined by the state of **M66EN** at the rising edge of **RST#**). The device must detect PCI-X mode and conventional 66 MHz mode separately and enable its PLL appropriately.

Implementation Note: Internal and External PLLs

A design can implement a PLL either inside or outside the device. If the PLL is inside the device, the device requires an **M66EN** input pin to determine whether to enable the PLL in conventional PCI mode. If the PLL is external to the device, the device requires a "PCI-X mode" output pin to enable the PLL in PCI-X mode, since PCI-X mode is controlled by the states of several bus control signals at the rising edge of **RST#**.

6.2.2. System Initialization Requirements

The system is required at power-up to determine the proper operating mode for the bus and to apply the appropriate PCI-X initialization pattern to the bus before the rising edge of **RST#**. See Section 6.1.2 for the operating requirements of the system as a function of what devices are present on the bus. The timing requirements for the PCI-X initialization pattern are shown in Figure 9-14. Timing parameter values are shown in Table 9-5 along with the other **RST#** timing parameters. Because the system knows that all power supply voltages are within their respective tolerances, the system is permitted to actively assert and deassert the appropriate signals of the PCI-X initialization pattern. Alternatively, the system is permitted to drive only those signals in the PCI-X initialization pattern that are to be asserted and allow the bus pull-up resisters to deassert the rest.

The system is also required to apply the PCI-X initialization pattern with the same timing requirement any other time **RST#** is asserted on this bus; e.g., if software sets a control bit in the source bridge that asserts **RST#** to the bus.

A system that does not support hot-plug PCI-X slots is permitted to bus **PCIXCAP** for all of the slots and sense its state with a single circuit (see Section 14). A system that supports hot-plug PCI-X slots must provide a means for the Hot-Plug System Driver to determine the states of the **PCIXCAP** pins for each hot-plug slot it controls without powering on the slot.

6.2.3. Mode and Frequency Initialization Sequence

Mode and frequency initialization requirements are very similar for host bridges and PCI-X bridges. PCI-X bridge requirements are presented in Section 8.9.

6.2.3.1. System Power-Up

The system and a PCI-X host bridge initialize the devices on the PCI bus as follows:

1. Apply power to all devices on the bus. While the power supply voltages are stabilizing, float the bus control signals that are included in the PCI-X initialization pattern (as required by PCI 2.2) and assert **RST#**. The pull-up resistors on these signals deassert them. To prevent **AD**, **C/BE#**, and **PAR** signals from floating while **RST#** is asserted, the central resource optionally drives these signals while **RST#** is asserted, but only to a logic low level. They may not be driven high.. When the power supply indicates that all of the supply voltages are within the proper tolerances, proceed to the next step.
2. Sense the states of **PCIXCAP** and **M66EN** for all devices on the bus.
3. Select the appropriate mode and clock frequency for the cards present on this bus as described in Section 6.1.2.

4. If the mode is to be 33 MHz conventional PCI, deassert**M66EN** for all devices on the bus. (This requirement is automatically met if **M66EN** is bused for all devices on the bus.)
5. Apply the PCI-X initialization pattern (from Table 6-2) on the bus.
6. Deassert **RST#** to place all devices on the bus in the appropriate mode.

Implementation Note: Mode and Frequency Initialization Sequence for a Bus that Includes Hot-Plug Slots

In some cases, a source bridge for a bus that includes PCI hot-plug slots must initialize the bus in a different manner from a non-hot-plug bus. In some hot-plug platforms that support conventional 66 MHz mode, the bridge cannot determine whether conventional add-in cards are capable of 66 MHz operation without first powering up the cards to read the **M66EN** pin or to read the 66 MHz Capable bit in the device's Status register. Powering up a hot-plug slot generally requires the use of the PCI Hot-Plug Controller. If the Hot-Plug Controller is a device on the same bus or a subordinate bus to the one being initialized, the bus must be initialized twice as described below.

When such a system is first powered up, the clock frequency must be set to 33 MHz (or lower) and the bus must be set to conventional mode. In other words, at the first rising edge of **RST#**, the PCI-X initialization pattern must select conventional mode, and **M66EN** must be deasserted for all slots. System software then turns on all conventional slots and determines whether each is capable of 66 MHz operation. If a conventional 33 MHz card is found, no further action is required since the bus is already operating in that mode. However, if no conventional 33 MHz cards are found, **RST#** for this bus must be asserted again, and the clock frequency must be changed as appropriate for the system and the capabilities of the devices found there. The bridge then drives the appropriate PCI-X initialization pattern and deasserts **RST#** as described above for bridges without hot-plug slots.

6.2.3.2. Hot Insertion in a PCI-X System

As described in the PCI HP 1.0, add-in cards cannot be connected to a bus whose clock is operating at a frequency higher than the card can tolerate. Without connecting the card to the bus, the Hot-Plug System Driver must determine the maximum frequency capability of the card. (Applying power to the card is acceptable but connecting it to the bus is not.)

The add-in card uses the **M66EN** pin and the **PCIXCAP** pin (as described in Table 6-1) to indicate its capabilities. If the slot supports conventional 66 MHz timing, PCI HP 1.0 requires the Hot-Plug System Driver to determine the state of **M66EN** for each slot. PCI-X systems that include hot-plug slots must enable the Hot-Plug System Driver to determine the state (open, pull-down, or ground) of the **PCIXCAP** pin of each slot (see Section 14). The Hot-Plug System Driver must not connect a slot to a bus if the clock is too fast for the card, or if the card does not support the current operating mode of the bus.

If a PCI-X-capable card is hot-inserted onto a bus that is operating in PCI-X mode at an acceptable frequency, the Hot-Plug Controller must drive the PCI-X initialization pattern on the bus with the proper timing prior to the rising edge of **RST#** for that slot.

6.2.4. Device Number and Bus Number Initialization

As described in Section 7.2.4, each function of a PCI-X device includes a Device Number and a Bus Number register. These registers store the device number and bus number used by the device in its Requester ID and Completer ID.

All bits in these registers are set to ones when **RST#** is asserted. After **RST#** is asserted and deasserted to a device, the system must initialize these registers by executing a Configuration Write transaction that addresses the device. As described in Section 7.2.4, each time the device is addressed by a Configuration Write transaction, the device stores the device and bus number from the configuration address and attributes (see Section 2.7.2.2) in the registers.

When the system is first initialized, system configuration software writes to the Configuration Space of each device of each PCI bus in the system as part of its normal system initialization process. The Device Number and Bus Number registers are automatically initialized when the software writes to each device's Configuration Space. Similarly, after **RST#** is asserted and deasserted for any other reason, system configuration software must reinitialize the device's Configuration Space, and in the process automatically initialize the Device Number and Bus Number registers. For example, after a power-management event that includes the assertion of **RST#**, or after a hot-insertion event, system configuration software must initialize the devices' Configuration Space and in so doing automatically initializes the Device Number and Bus Number registers.

If system configuration software changes the number assigned to a PCI bus segment operating in PCI-X mode, that software must also execute Configuration Write transactions to each device on that bus to update the Bus Number registers in those devices. See for example the requirements for the Secondary Bus Number register in a PCI-X bridge in Section 8.6.1.

Implementation Note: Initiating Transactions Before the Bus and Device Number Registers are Initialized

The Bus Number and Device Number registers are initialized by Configuration Write transactions to the device. Before a device initiates any transaction other than a Split Completion, system configuration software must set the Bus Master bit by executing a Configuration Write transaction to the device's Command register (as defined in PCI 2.2). This write to the Command register initializes the Bus Number and Device Number registers.

PCI-X devices initiate Split Completion transactions independent of the state of the Bus Master bit. If a device executes Configuration Read transactions as Split Transactions, the device responds to the first Configuration Reads before the Bus Number and Device Number registers are initialized. In this case, the Completer ID in the attribute phase of these Split Completions use the uninitialized values of these registers. Although this is unusual and diagnostic equipment must be prepared to accept this case, the Requester ID (from the Split Request) is used for routing of the Split Completion, so the transaction completes properly.

If a device initiates transactions on a bus that has not been initialized by system configuration software, that device must not allow a Split Transaction to be executed with an ambiguous Requester ID. For example, if a system management device discovers that a system has failed during the boot process and the system management device initiates transactions on the bus to determine the cause of the failure, the system management device must avoid ambiguous Requester IDs in those transactions. One alternative to avoid ambiguous Requester IDs is for the system management device to assign itself one Requester ID and initiate Configuration Write transactions to all the other devices on the bus to assign them different Requester IDs. Another alternative would be for the system management device to assert **RST#** and initialize the bus in conventional PCI mode (which does not use Requester IDs). In most cases, such behavior by a system management device requires the cooperation of the source bridge, since the source bridge normally controls the assertion of **RST#**, drives the PCI-X initialization pattern, and sets the Bus Master bit in all other devices.

7. Configuration Space for Type 00h Header Devices

This section contains the Configuration Space requirements for all PCI-X devices that use a Type 00h header. Refer to Section 8.6 for the requirements for PCI-X bridges (Type 01h header).

7.1. PCI-X Effects on Conventional Configuration Space Header

PCI-X devices include the standard Configuration Space header defined in PCI 2.2. In conventional PCI mode, all of these registers function exactly as specified there. If the device is initialized to PCI-X mode (see Section 6.2), the requirements for these registers change as follows:

1. Command Register—
   Fast Back-to-Back Enable: Ignored by the device in PCI-X mode.

   Stepping Control: Ignored by the device in PCI-X mode.

   Memory Write and Invalidate Enable: Ignored by the device in PCI-X mode.

   Bus Master: Ignored by the device when initiating Split Completions.

2. Status Register—
   Capabilities List: PCI-X devices include the PCI-X Capabilities List item, so this bit is set to 1 for all PCI-X devices (both in PCI-X mode and conventional mode).

   Fast Back-to-Back Capable: This bit is allowed to have any value when the device is in PCI-X mode. (PCI-X devices never use fast back-to-back timing, regardless of the state of this bit.)

   Detected Parity Error and *Master Data Parity Error:* These bits are set as described in Section 5.4.1.

   DEVSEL timing: Indicates the device's conventional-PCI-mode **DEVSEL#** timing as defined in PCI 2.2 regardless of the actual operating mode of the device.

3. Base Address Registers—All Base Address registers that request memory resources (except the Expansion ROM Base Address register) must support 64-bit addressing using the method defined in PCI 2.2. The Prefetchable bit must be set unless the range contains locations with read side effects or locations in which the device does not tolerate write merging. (See Section 2.12.1 for more details.) The minimum memory address range requested by a Base Address register is 128 bytes. To conserve address space, it is recommended that devices request an address range no larger than the smallest integral power of two that is larger than the range actually used by the device.

4. Latency Timer Register—The default value of the Latency Timer register is 64 in PCI-X mode. (See Section 4.4 for more details.)

5. Cacheline Size Register, MIN_GNT and MAX_LAT—The device optionally uses these registers for internal optimizations beyond the scope of this specification. Such implementation must comply with the register requirements specified in PCI 2.2. System configuration software must initialize these registers as specified in PCI 2.2.

Implementation Note: Setting the Prefetchable Bit in a PCI-X Memory Base Address Register

Each PCI-X transaction includes the byte count (DWORD transactions imply a byte count of four), so no “prefetching” of data occurs. However, the term is preserved to differentiate two types of memory ranges in device Base Address registers and bridge memory range registers.

The requirements for setting the Prefetchable bit in memory Base Address registers are different in PCI-X than for conventional PCI. Only the requirements for no read side effects and the allowance of write merging apply to PCI-X systems. Only bytes for which byte enables are asserted are relevant in a PCI-X transaction (other than for parity generation and checking), so there is no requirement that all bytes be returned when reading from a prefetchable range in a PCI-X device.

The PCI-X definition requires that the Prefetchable bit be set in memory Base Address registers because range registers in PCI and PCI-X bridges for prefetchable memory are 64-bit wide. This provides the flexibility for system configuration software to locate the device above the first 4 GB boundary. Non-prefetchable ranges for devices behind a PCI or PCI-X bridge must be located below the first 4 GB boundary.

Implementation Note: Conserving Address Space

Unlike conventional PCI devices, which were permitted to decode 4 KB address ranges even if they did not need that much, PCI-X devices that implement Base Address registers are encouraged to request the minimum address space they need to support their programming interface. Available address space in some systems is congested. This is particularly true of PCI hot-plug system in which the addresses available for adding new devices must be partitioned among several slots. Available address space is further fragmented when devices and empty slots appear on the secondary side of a PCI-X bridge. Devices that don’t request more address space than they need are preferred in such systems.

7.2. PCI-X Capabilities List Item

PCI-X devices include new status and control registers that are located in the Capabilities List in Configuration Space of each function. System configuration software determines whether a device supports PCI-X by the presence of this item in the Capabilities List. This list item must appear in a PCI-X device’s Configuration Space regardless of whether the device is operating in conventional PCI mode or PCI-X mode.

A multifunction device that implements PCI-X must implement these registers in the Configuration Space of each function. (PCI-X bridge functions use the register format shown in Section 8.6.2.)

If the device is installed on an add-in card, the connection of the **PCIXCAP** pin of the add-in card must be consistent with the presence of this Capability List item in each function of the first device on the card. That is, the connection of the **PCIXCAP** pin must indicate the card is capable of operating in PCI-X mode if and only if the PCI-X Capability List item is present in the functions of the device that connects to the PCI connector of the add-in card (not behind a bridge). See Section 8.6.2 for PCI-X Capability List item for PCI-X bridge functions. See Section 9.10 for the connection of the **PCIXCAP** pin.

Unless otherwise noted, all PCI-X devices treat Configuration Space write operations to reserved registers or bits as no-ops; that is, the access completes normally on the bus and the data discarded. Read accesses to reserved or unimplemented registers or bits complete normally and a data value of 0 is returned.

As in conventional PCI, software must take care to deal correctly with bit-encoded fields that have some bits reserved for future use. On reads, software must use appropriate masks to extract the defined bits and may not rely on reserved bits being any particular value. On writes, software must ensure that the values of reserved bit positions are preserved; that is, the values of reserved bit positions must first be read, merged with the new values for other bit positions, and the data then written back.

Figure 7-1 shows the PCI-X Capabilities List item for a device with a Type 00h Configuration Space header. The corresponding item for a device with a Type 01h Configuration Space header (a PCI-X bridge) is shown in Section 8.6.2.

31	24	23	16	15	8	7	0
PCI-X Command				Next Capability		PCI-X Capability ID	
PCI-X Status							

Figure 7-1: PCI-X Capabilities List Item for a Type 00h Configuration Header

7.2.1. PCI-X ID

This register identifies this item in the Capabilities List as a PCI-X register set. It is read-only returning 07h when read. (Note that PCI-X bridges use the same PCI-X ID in the Capabilities List but use a different register format specified in Section 8.6.2.)

7.2.2. Next Capabilities Pointer

This register points to the next item in the Capabilities List, as required by PCI 2.2.

7.2.3. PCI-X Command Register

This register controls various modes and features of the PCI-X device. Bit Location 0 is the least significant bit in the register.

Table 7-1: PCI-X Command Register

Bit Location	Description
0	**Data Parity Error Recovery Enable.** (read/write) The device driver sets this bit to enable the device to attempt to recover from data parity errors as described in Section 5.4.1.1. If this bit is 0 and the device is in PCI-X mode, the device asserts **SERR#** (if enabled) whenever the Master Data Parity Error bit (Status register, bit 8) is set. State after **RST#** is 0.
1	**Enable Relaxed Ordering.** (read/write) If this bit is set, the device is permitted to set the Relaxed Ordering bit in the Requester Attributes of transactions it initiates that do not require strong write ordering (see Section 2.5 and Section 11). State after **RST#** is 1. It is permitted to be read-only and set to 0 in devices that never set the Relaxed Ordering attribute bit.

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>3-2</td><td><b>Maximum Memory Read Byte Count.</b> (read/write)
This register sets the maximum byte count the device uses when initiating a Sequence with one of the burst memory read commands. It enables system configuration software to tune system performance. Device drivers must not modify this register without an understanding of the impact on the rest of the system. See Section 13.1 for setting recommendations.

System configuration software is permitted to write to this register at any time. The most recent value of the register is used each time the device prepares a new Sequence. (In some cases, if the device has already prepared some Sequences with the previous setting but not yet initiated them, Sequences with the old setting are initiated after the new value is set.)

<table><tr><th>Register</th><th>Maximum Byte Count</th></tr><tr><td>0</td><td>512</td></tr><tr><td>1</td><td>1024</td></tr><tr><td>2</td><td>2048</td></tr><tr><td>3</td><td>4096</td></tr></table>
State after <b>RST#</b> is 0.</td></tr>
<tr><td>6-4</td><td><b>Maximum Outstanding Split Transactions.</b> (read/write)
This register sets the maximum number of Split Transactions the device is permitted to have outstanding at one time. It enables system configuration software to tune system performance. Device drivers must not modify this register without an understanding of the impact on the rest of the system. Host bridges are permitted to implement this register as read-only.

System configuration software is permitted to write to this register at any time. The most recent value of the register is used each time the device prepares a new Sequence. (In some cases, if the device has already prepared some Sequences with the previous setting but not yet initiated them, Sequences with the old setting are initiated after the new value is set.)

<table><tr><th>Register</th><th>Maximum Outstanding</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td></tr><tr><td>3</td><td>4</td></tr><tr><td>4</td><td>8</td></tr><tr><td>5</td><td>12</td></tr><tr><td>6</td><td>16</td></tr><tr><td>7</td><td>32</td></tr></table>
If <b>RST#</b> is asserted, the device initializes this register to indicate the maximum number of Split Transactions the device is designed to have outstanding when the Maximum Memory Read Byte Count register is set to 0 (512 bytes).</td></tr>
<tr><td>15-7</td><td><b>Reserved.</b></td></tr>
</table>

7.2.4. PCI-X Status Register

This register identifies the capabilities and current operating mode of the device as listed in the following table.

Table 7-2: PCI-X Status Register

Bit Location	Description
2-0	**Function Number.** (read-only) This register is read for diagnostic purposes only. It indicates the number of this function; i.e., the number in the Function Number field (**AD[10::08]**) of the address of a Type 0 configuration transaction to which this function responds. The function uses this number as part of its Requester ID and Completer ID.
7-3	**Device Number.** (read-only) This register is read for diagnostic purposes only. It indicates the number of the device containing this function, i.e., the number in the Device Number field (**AD[15::11]**) of the address of a Type 0 configuration transaction that is assigned to the device containing this function by the connection of the system hardware. Device number 00h is reserved for the source bridge. Therefore, the system must assign a device number other than 00h to all other devices. The function uses this number as part of its Requester ID and Completer ID. Each time the function is addressed by a Configuration Write transaction, the device must update this register with the contents of **AD[15::11]** of the address phase of the Configuration Write, regardless of which register in the function is addressed by the transaction. The function is addressed by a Configuration Write transaction if all of the following are true: 1. The transaction uses a Configuration Write command. 2. **IDSEL** is asserted during the address phase. 3. **AD[1::0]** are 00b (Type 0 configuration transaction). 4. **AD[10::08]** of the configuration address contain the appropriate function number. State after **RST#** is 1Fh.
15-8	**Bus Number.** (read-only) This register is read for diagnostic purposes only. It indicates the number of the bus segment for the device containing this function. The function uses this number as part of its Requester ID and Completer ID. For all devices other than the source bridge, each time the function is addressed by a Configuration Write transaction, the function must update this register with the contents of **AD[7::0]** of the attribute phase of the Configuration Write, regardless of which register in the function is addressed by the transaction. The function is addressed by a Configuration Write transaction when all of the following are true: 1. The transaction uses a Configuration Write command. 2. **IDSEL** is asserted during the address phase. 3. **AD[1::0]** are 00b (Type 0 configuration transaction). 4. **AD[10::08]** of the configuration address contain the appropriate function number. State after **RST#** is FFh

Bit Location	Description
16	**64-bit Device.** (read-only) This bit is used by system management software to assist the user in identifying the best slot for an add-in card. If the function is part of a device that is installed on an add-in card and connects directly to the PCI connector (not through a bridge), this bit is set if and only if all of the following are true: 1. The function implements a 64-bit **AD** interface. 2. The device implements a 64-bit **AD** interface. 3. The add-in card implements a 64-bit PCI connector. This requirement is independent of the width of the slot in which the card is installed. If the device is subordinate to a bridge on an add-in card, or if the device is installed on the system board (not in a slot), this bit is permitted to have any value. 0 = The bus is 32 bits wide. 1 = The bus is 64 bits wide.
17	**133 MHz Capable.** (read-only) This bit is used by system management software to assist the user in identifying the best slot for an add-in card. It is also used in some hot-plug systems to determine whether an add-in card would function properly if the bus were changed to PCI-X 133 mode. If the device is installed on an add-in card and connects directly to the PCI connector (not through a bridge), this bit indicates whether the device is capable of 133 MHz operation in PCI-X mode. The connection of the card's **PCIXCAP** pin (see Section 6.2) must be consistent with this bit. If the device is subordinate to a bridge on an add-in card, or if the device is installed on the system board (not in a slot), this bit is permitted to have any value. All functions within a multi-function device have the same value for this bit. 0 = The device's maximum operating frequency is 66 MHz. 1 = The device's maximum operating frequency is 133 MHz.
18	**Split Completion Discarded.** (write 1 to clear) This bit is set if the device discards a Split Completion because the requester would not accept it, except as noted in Section 5.4.4. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = no Split Completion has been discarded. 1 = a Split Completion has been discarded.
19	**Unexpected Split Completion.** (write 1 to clear) This bit is set if an unexpected Split Completion with this device's Requester ID is received. See Section 5.4.5 for more details. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = no unexpected Split Completion has been received. 1 = an unexpected Split Completion has been received.

Bit Location	Description
20	**Device Complexity.** (read-only) This bit indicates whether this device is a simple device or a bridge device, as defined in Section 2.13. Simple devices are subject to the posting and required acceptance rules shown in Section 2.13. If a device does not meet the definition of a simple device, it is a bridge device and must follow the rules in Section 8.2. 0 = simple device 1 = bridge device
22-21	**Designed Maximum Memory Read Byte Count.** (read-only) This register indicates a number that is greater than or equal to the maximum byte count the device-function is designed to use when initiating a Sequence with one of the burst memory read commands. The device-function must report the smallest value that correctly indicates its capability. If system configuration software sets the Maximum Memory Read Byte Count register (in the PCI-X Command register) to a value different from this register, the device uses the smaller value. Register — Maximum Byte Count 0 — 512 1 — 1024 2 — 2048 3 — 4096
25-23	**Designed Maximum Outstanding Split Transactions.** (read-only) This register indicates a number that is greater than or equal to the maximum number of Split Transactions the device-function is designed to have outstanding at one time. The device-function must report the smallest value that correctly indicates its capability. If the number depends on the value in the Maximum Memory Read Byte Count register (in the PCI-X Command register), this register must be accurate for the present setting of the Maximum Memory Read Byte Count register. If system configuration software sets the Maximum Outstanding Split Transaction register (in the PCI-X Command register) to a value different from this register, the device uses the smaller value. Register — Maximum Outstanding 0 — 1 1 — 2 2 — 3 3 — 4 4 — 8 5 — 12 6 — 16 7 — 32

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>28-26</td><td><b>Designed Maximum Cumulative Read Size.</b> (read-only)
This register indicates a number that is greater than or equal to the maximum cumulative size of all burst memory read transactions the device-function is designed to have outstanding at one time. The device-function must report the smallest value that correctly indicates its capability. If the number depends on the value in the Maximum Memory Read Byte Count register (in the PCI-X Command register), this register must be accurate for the present setting of the Maximum Memory Read Byte Count register.
<table>
<tr><th>Register</th><th colspan="2">Maximum Outstanding</th></tr>
<tr><th></th><th>ADQs</th><th>bytes (ref)</th></tr>
<tr><td>0</td><td>8</td><td>1 KB</td></tr>
<tr><td>1</td><td>16</td><td>2 KB</td></tr>
<tr><td>2</td><td>32</td><td>4 KB</td></tr>
<tr><td>3</td><td>64</td><td>8 KB</td></tr>
<tr><td>4</td><td>128</td><td>16 KB</td></tr>
<tr><td>5</td><td>256</td><td>32 KB</td></tr>
<tr><td>6</td><td>512</td><td>64 KB</td></tr>
<tr><td>7</td><td>1024</td><td>128 KB</td></tr>
</table></td></tr>
<tr><td>29</td><td><b>Received Split Completion Error Message.</b> (write 1 to clear)
This bit is set if the device receives a Split Completion Message with the Split Completion Error attribute bit set. See Section 5.4.6 for details. Once set, this bit remains set until software writes a 1 to this location. State after <b>RST#</b> is 0.
0 = no Split Completion error message received.
1 = a Split Completion error message has been received.</td></tr>
<tr><td>31-30</td><td>Reserved</td></tr>
</table>

Implementation Note: Updating the Bus Number and Device Number

The Bus Number and Device Number registers are updated on every Configuration Write transaction that addresses the function. It is important that each function update these registers *each* time they are addressed by a Configuration Write (not just the first one after power-up). In some systems, system-configuration software changes PCI bus segments one or more times after power-up. For example, if an add-in card containing a bridge was hot-added to a system, the system-configuration software might renumber other buses to make room for the new one.

The device number is assigned by the connection of the system hardware and, therefore, should not change after **RST#** deasserts. However, for consistency with the Bus Number register, the Device Number register is also required to be updated on every Configuration Write transaction that addresses the device. Future versions of this specification may depend upon this behavior.

Implementation Note: Using the 64-bit Device and 133 MHz Capable Status Bits.

The 64-bit Device and 133 MHz Capable bits in the PCI-X Status register (and the PCI-X Bridge Status register) are intended for use by system management software to assist the user in identifying the best slot for an add-in card. For system management software to make recommendations, it must know not only the characteristics of the add-in cards but also the characteristics of the slots. The method by which software determines the characteristics of the slots is beyond the scope of this specification.

The states of these bits are not useful for devices other than the first device of an add-in card. The user cannot affect the connection of devices behind a bridge on an add-in card, or devices permanently installed on the system board (not in a slot). In these cases the bits are permitted to have any value. System management software is recommended not to report the states of the bits in these cases.

Some implementations of multi-function devices may contain both 64-bit and 32-bit functions within the same device. System management software should recommend that the user place an add-in card in a 64-bit slot, if the 64-bit Device bit is set for any function of a multi-function device that is the first device on the card.

A device designed for use both on 64-bit and 32-bit add-in cards must implement a method for the card designer to set the 64-bit Device bit only in 64-bit add-in card applications. The method for setting this bit is not specified, but commonly used techniques include pull-up or pull-down resistors on pins that are outputs after the rising edge of **RST#**, or serial EEPROMs that are down-loaded when the device powers up.

The frequency indicated by the connection of the **PCIXCAP** pin must be consistent with the 133 MHz Capable bit of the first device on the card.

7.3. Use of I/O Space

I/O Space is limited, especially in hot-plug systems, and I/O references are generally slower than memory references. PCI-X devices are discouraged from using I/O Space. If I/O Space is required, the device must also provide access to the same registers in Memory Space. In other words, if the device uses a Base Address register (BAR) to request I/O Space, it must also use another BAR to request Memory Space for the same resource. If sufficient I/O Space is not available, system configuration software only assigns Memory Space resources. (PCI 2.2 recommends this mapping of the device into both address spaces.)

8. PCI-X Bridge Additional Design Requirements

A PCI-X bridge is a device capable of connecting two buses operating in PCI-X mode. Since any bus capable of operating in PCI-X mode must operate in conventional PCI mode when a conventional device is installed on that bus, a PCI-X bridge must operate with either or both of its interfaces in conventional mode. When opera ting in conventional mode, the bridge's behavior is governed by Bridge 1.1.

Unless otherwise specified in this section, a PCI-X bridge must meet all the requirements specified throughout this document for PCI-X devices both on its primary and secondary interfaces. As in conventional PCI, a PCI-X bridge creates a new bus segment in the PCI configuration hierarchy. The PCI-X bridge is the source bridge for this segment and must meet all the requirements specified throughout this document for a source bridge for this segment. For example, the PCI-X bridge must drive the PCI-X initialization pattern on the secondary bus before deasserting secondary **RST#** to place secondary bus devices in the proper mode (conventional or PCI-X) and to indicate the secondary bus clock frequency (in PCI-X mode).

This section includes additional requirements that are unique to bridges. Not all bridge requirements are shown in this section. Some bridge requirements that are similar to or related to requirements for all devices are shown elsewhere. For example:

• Requester Attributes	Section 2.5
• Configuration Transactions	Section 2.7.2.2
• Split Transactions	Section 2.10
• Split Completion Error Message Reporting	Section 5.4.6

8.1. Summary of Key Requirements

The following list is a summary of some of the key requirements of a PCI-X bridge:

- Each interface must operate in conventional PCI mode if a conventional PCI device is installed there (**PCIXCAP** connected to ground). The source bridge for the primary bus informs the PCI-X bridge of the mode of the primary bus with the PCI-X initialization pattern at the rising edge of primary **RST#**. The PCI-X bridge must sense the state of secondary **PCIXCAP** (see Section 14) and initialize the secondary bus devices properly (see Section 8.9).

- Like all PCI-X devices, PCI-X bridges must support 64-bit addressing on both interfaces. They are permitted to implement either a 64-bit or 32-bit **AD** bus on either interface.

- PCI-X bridges must complete all DWORD transactions and all burst memory read transactions as Split Transactions, if the transaction crosses the bridge (i.e., the requester is on one interface and the completer is on the other) and the originating interface is in PCI-X mode. Transactions that address locations internal to the bridge have the same requirements described throughout this document for other PCI-X devices.

- As in conventional PCI, PCI -X bridges use a Type 01h Configuration Space header. The PCI-X registers in the Capabilities List item are different for Type 01h devices than for other devices (see Section 8.6).

- System topologies, Special Cycle, and Interrupt Acknowledge cases listed as unsupported in Bridge 1.1 are not supported by PCI-X bridges.
- As in conventional PCI, support for 66 MHz conventional PCI timing is optional for both interfaces.

8.2. PCI-X Bridges and Application Bridges

All of the requirements of this section apply to PCI-X devices that identify themselves as PCI-X bridges by using a Type 01h Configuration Space header and Base Class 06h and Sub-Class 04h.

In some cases, a device that uses a Type 00h Configuration Space header and a different Base Class or Sub-Class code exhibits some of the characteristics of a bridge. As described in PCI 2.2, a device that implements internal posting of memory write transactions that the device must initiate on the PCI-X interface is considered a bridge. (In most cases, such bridges connect a local intelligent subsystem to the PCI-X interface.) Because such devices use a Base Class and Sub-Class that reflects the function they perform, this document refers to them as application bridges. Host bridges and bridges to other buses that use a Type 00h Configuration Space header are application bridges. Application bridges identify themselves as bridges by setting the Device Complexity bit in the PCI-X Status register (see Section 7.2.4).

Except as noted below, application bridge must meet all the requirements of simple devices described throughout this document. Additionally, application bridges must meet at least the following bridge requirements. Additional PCI-X bridge requirements may be necessary, depending upon the complexity of the application bridge:

1. Transaction ordering and deadlock avoidance rules presented in Section 8.4.4
2. Required acceptance rules presented in Section 8.4.5
3. Exclusive access rules presented in Section 8.5 unless the system guarantees that no exclusive access ever addresses a completer on the other side of the bridge

PCI-X bridges and application bridges are exempt from the following PCI-X simple-device requirements for transactions that cross from one interface to another:

1. Maximum Completion Time limit presented in Section 2.13. Bridges must forward transactions as quickly as the ordering rules permit. If internal buffers for memory writes or for Split Requests are full, the bridge terminates subsequent transactions with Retry.
2. Retry and disconnection of Split Completions. Bridges terminate Split Completions with Retry or Disconnect at Next ADB if required by the transaction ordering rules or if buffer space designed for Split Completions is full of previous Split Completions.

8.3. Address Decoding

PCI-X bridges decode Memory Space, I/O Space, and Configuration Space the same as conventional PCI bridges. All memory and I/O range registers are programmed and interpreted the same way. Configuration transactions are forwarded based on bus number the same as conventional PCI.

Split Completions are forwarded based on the Requester Bus Number field in the Split Completion address (similar to configuration transactions) as described in Section 2.10.3.

8.4. Bridge Operation

As in conventional PCI, PCI-X bridges are required to post memory write transactions that cross the bridge in either direction if space is available in the bridge. PCI-X bridges are required to terminate memory read transactions, I/O transactions, and configuration transactions with Split Response if the transactions cross the bridge, space for the request is available in the bridge, and the bridge's requester-side interface is in PCI-X mode. (See Section 8.7.1.2 for an exception for data parity errors on non-posted write transactions.) If bridge buffers used for these transactions are full, and the transaction addresses a completer on the other side of the bridge, the bridge is allowed to terminate the transaction with Retry.

See Section 8.4.2.1 for management of Split Completion buffers.

Buffer requirements specified throughout this section for transactions flowing upstream are independent of transactions flowing downstream, and vice versa.

8.4.1. Buffer Size Requirements

PCI-X bridges must provide at least two ADQs of buffer space for memory write data (except as allowed in Section 8.4.6). A bridge's memory write buffer area is considered full when less than two ADQs of buffer space are available (except as allowed in Section 8.4.6). Bridges are encouraged to implement much larger buffers to enable the posting of multiple and/or longer burst memory write transactions.

PCI-X bridges must provide at least two ADQs of buffer space for Split Completion data, with one exception described below. Except in this one case, a bridge's Split Completion buffer area is considered full when less than two ADQs of buffer space are available. Bridges are encouraged to implement much larger buffers to enable the storing of multiple and/or longer Split Completions.

In the exception case, a PCI-X bridge is permitted to accept a Split Completion transaction with less than two ADQs of buffer space available if that bridge provides alternate means for guaranteeing that it never holds a Split Completion transaction that is too short to forward correctly, as described below.

A bridge with less than two ADQs of buffer space for Split Completions is not permitted to signal Disconnect at Next ADB on the first data phase of a Split Completion if both of the following are true:

- The Split Completion would otherwise cross the ADB.
- The Split Completion begins less than four data phases from the ADB.

Implementation Note: Buffer Space for Split Completion Data

A bridge holds a portion of the Split Completion that is too small to forward correctly on the destination bus if all of the following are true:

- The Split Completion begins less than four data phases from an ADB.
- The byte count is such that the Split Completion would cross the ADB.
- A bridge forwarding the Split Completion has space available for only one ADQ and signals Disconnect at Next ADB on the first data phase of the Split Completion.

In such a case the bridge would hold less than four data phases of the Split Completion, but the byte count would indicate that the transaction extended beyond the ADB. If the bridge were to attempt to forward such a partial Split Completion to the completer, it would be unable to disconnect the transaction at the ADB, but would not have the data to proceed beyond the ADB. (See Section 8.4.6 for a similar situation for memory write transactions.)

The bridge avoids this problem if it has a minimum of two ADQs of buffer space available for storing Split Completions. If a Split Completion arrives when the bridge has only one ADQ of buffer space available, the bridge signals Retry. (See Section 8.4.5 for additional restrictions on the use of Retry.)

PCI-X bridges must have available a minimum buffer space of two ADQs for holding immediate read data before initiating a read request.

Implementation Note: Buffer Space for Immediate Read Data

If a bridge forwards a long burst read transaction and the target responds immediately with data, the bridge must accept the data at least to the first ADB. If the starting address of the transaction is less than four data phases from an ADB, the bridge is not able to disconnect on that ADB and must proceed to the next. In this case, the bridge must have two ADQ buffers, one for the data between the starting address and the first ADB and the other for the data between the first and second ADBs.

If the Split Transaction Commitment Limit field in the bridge's Split Transaction Control register is set no larger than the Split Transaction Capacity field, the bridge always has buffer space available for the entire Sequence. In this case, no additional action is required to guarantee that two ADQ buffers are available before initiating the transaction.

8.4.2. Forwarding Split Transactions

A PCI-X bridge must terminate with Split Response all transactions that address a completer on the other side of the bridge and use one of the following commands. (Bridges are also allowed to terminate with Retry any transaction that crosses the bridge if bridge buffers for those transactions are filled with previous transactions crossing the bridge in the same direction.)

- Memory Read DWORD
- Memory Read Block
- Alias to Memory Read Block
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write

In most cases, a PCI-X bridge forwards a Split Request from one bus to another and forwards the Split Completion in the opposite direction without modifying the transactions or keeping track of what transactions are outstanding (other than to reserve an amount of buffer space for the Split Completion as described in Section 8.4.2.1). The following exceptions to this rule are specified elsewhere:

• Configuration Transactions	Section 2.7.2.2
• Completer executes the transaction as an Immediate Transaction	Section 8.4.2.2
• One or more of the bridge interfaces is in conventional mode	Section 8.4.3

Implementation Note: Decomposing Split Transactions

A bridge is not obligated to forward any Split Transaction to the destination bus in the same size that it received it on the originating bus. However, since a bridge with a Type 01h Configuration Space header does not include a PCI-X Command register, decomposing one request into multiple requests reduces the ability of the system to manage Split Transaction resources through the use of the Maximum Outstanding Split Transactions register. Furthermore, decomposing one large request into multiple smaller ones generally increases complexity of the bridge and often leads to lower efficiency on the destination bus. Therefore, this behavior is discouraged. The following discussion illustrates some of the additional complexity that such behavior would introduce.

If a bridge decomposes a request it terminated with Split Response on the originating bus, it must generate unique Sequence IDs for each of the decomposed requests. This generally requires the bridge to use its own Requester ID for the destination bus because that is the only way the bridge guarantees that the Sequence ID is unique. When the bridge initiates such a Sequence on its primary interface, the bus number would be the number in the Primary Bus Number register (which is the same as the number in the Bus Number field in the PCI-X Bridge Status register). The device number would be the number in the Device Number field in the PCI-X Bridge Status register. When the bridge initiates such a Sequence on its secondary interface, the bus number would be the number in the Secondary Bus Number register. The device number would be 00h, since this device number is reserved for the source bridge of any bus. When the Split Completions return on the destination bus, the bridge must convert back to the original Sequence ID and must return the data in address order.

PCI-X bridges are generally not permitted to combine separate read Sequences into a single Sequence. Combining of read Sequences generally requires knowledge of the completer to avoid crossing a device boundary. Such knowledge is beyond the scope of the PCI-X definition.

A PCI-X bridge forwards a Split Request solely according to its starting address. If the starting address of a read transaction addresses a device on the other side of a PCI-X bridge, but one or more addresses between the starting address and ending address do not, the bridge forwards the Split Request unmodified.

Implementation Note: Burst Read Sequences that Cross Bridge Boundaries

Since normally functioning requesters understand the address range of the completer, and since combining of separate read Sequences by bridges is generally not allowed, read Sequences cross a bridge boundary only under abnormal conditions. However, if such a transaction crosses a bridge, the bridge simply forwards it based on the transaction's starting address.

If a bridge forwards such a transaction, the bridge must be prepared for the Sequence to complete on the destination bus in any of the following ways:

- The completer signals Split Response, initiates Split Completion transactions with data up to its device boundary, and then initiates a Split Completion Message indicating the byte count is out of range. (See Section 2.10.6.)
- The completer completes the transaction as an Immediate Transaction up to its device boundary and then disconnects the transaction. When the bridge attempts to continue the Sequence, it ends with Master-Abort.
- The completer signals Target-Abort.

A PCI-X bridge is permitted to combine Split Completions that are part of the same Sequence, provided that such combining does not violate the bridge ordering rules. (See Section 8.4.4.)

8.4.2.1. Split Completion Buffer Allocation

PCI-X bridges contain two registers that limit the forwarding of Split Requests (see Sections 8.6.2.5 and 8.6.2.6). The Split Transaction Capacity register indicates the amount of buffer space the bridge has for storing Split Completions. If the bridge stores Split Completions for burst memory read requests in a separate area from other Split Completions, this register indicates the size (in units of ADQs) of the area for storing Split Completions for burst memory reads. If the bridge stores Split Completions for burst memory reads in the same area as some or all other Split Completions, this register indicates the size of this area in units of ADQs.

The Split Transaction Commitment Limit registers indicate the cumulative Sequence size of the appropriate Split Transactions (see Sections 8.6.2.5 and 8.6.2.6) the bridge is allowed to have outstanding at one time (in units of ADQs). If the bridge enqueues a request to be forwarded and the size of that request plus all those the bridge presently has outstanding in that direction exceeds the contents of the Split Transaction Commitment register, the bridge is not permitted to assert **REQ#** for this request. After sufficient Split Completion transactions have been forwarded to their respective requesters such that the size of the request plus the total outstanding commitment is less than the commitment limit, the bridge is permitted to assert **REQ#** to forward the transaction.

If the bridge stores Split Completions for burst memory read requests in a separate area from other Split Completions, the Split Transaction Commitment Limit register applies only to burst memory reads. Such a bridge must never forward other Split Requests (e.g., I/O Read, I/O write, etc.) unless it has a place to store the corresponding Split Completions. If the bridge stores Split Completions for burst memory reads in the same area as some or all other Split Completions, this register applies to all Split Transactions stored with burst memory read transactions.

At power-up, the Split Transaction Commitment Limit register defaults to the same value as the Split Transaction Capacity register. At this setting, the bridge is allowed to forward Split Transactions whose cumulative size exactly fills the bridge buffers. If the Split Transaction Commitment Limit register is programmed to a value greater than the value of the Split Transaction Capacity register, the bridge is allowed to forward Split Transactions up to the Split Transaction Commitment Limit even though not all of the Split Completions for these transactions would fit in the bridge at one time. See Section 13.2 for recommendations for optimizing the setting of the Split Transaction Commitment Limit registers.

Unexpected Split Completion exceptions (see Section 5.4.5) that cross the bridge prevent the bridge from accurately tracking its Split Transaction commitment.

A setting of FFFFh in the Split Transaction Commitment Limit register allows the bridge to forward all Split Transactions (in the appropriate direction) without regard to the Sequence size or the amount of buffer space available in the bridge. The bridge is not required to track the size of outstanding commitments if the register is set to FFFFh. However, if the register is changed from FFFFh to a smaller value and all outstanding Split Transactions that cross the bridge complete, the bridge must begin to accurately track new outstanding commitments.

Implementation Note: Accurate Tracking of Outstanding Commitments after a Setting of FFFFh

The bridge is not required to track outstanding Split Transaction commitment if the Split Transaction Commitment Limit is set to FFFFh. If the register is later set to something smaller, the bridge has no way of knowing the size of the Split Transactions that are outstanding. If the bridge does not implement a method for synchronizing its commitment counters to the actual size of outstanding commitments, the bridge would regulate its outstanding commitments to the wrong limit indefinitely.

To synchronize the bridge's commitment counters, the bridge must set its commitment count to zero and immediately begin incrementing it when new Split Requests are forwarded across the bridge and decrementing it when Split Completions are forwarded to their requesters across the bridge. However, if a Split Completion would decrement the commitment count below zero, the commitment count must be set to zero. If at some point all outstanding Split Transactions finish, the bridge's commitment count is also zero. From that point on, the commitment limit is accurate.

If a Split Request must be forwarded by a bridge and the Sequence size of that one Sequence exceeds the Split Transaction Capacity of the bridge (for the appropriate direction), the bridge must wait to forward that Split Request until the bridge has no other Split Transactions outstanding in that direction. If the bridge allows other Split Requests to pass the large Split Request (see Section 8.4.4), the bridge must forward the large request eventually.

Implementation Note: Split Transaction Buffer Allocation Algorithm

The following is an example implementation that would meet the requirements defined above for PCI-X bridges. If the Split Transaction Commitment Limit register is set to FFFFh, the bridge forwards all requests regardless of size and does not track the number of outstanding Split Transactions. If the register is set to some other value, the bridge implements the following expression independently for Split Transactions crossing in either direction:

$$\text{TOST} + \text{NST} \leq \text{STCL}$$

Where:
TOST = Total outstanding Split Transactions in ADQs.
NST = Size of next Split Transaction in number of ADQs. Note that this is a function of the starting and ending addresses not just the byte count. If a transaction begins or ends between two ADBs, NST includes the whole ADQ.
STCL = Contents of the Split Transaction Commitment Limit register.

To implement the expression, the bridge would provide the Upstream and Downstream Split Transaction Control registers (see Sections 8.6.2.5 and 8.6.2.6) and the following capabilities (independently for transactions flowing upstream and downstream):

- TOST is a 16-bit counter that indicates the number of Split Transactions (in number of ADQs) that are outstanding from the bridge (in one direction). That is the number of ADQ-size buffers necessary to hold all the Split Completions for all Split Requests forwarded to the completer by the bridge but not yet returned to the requester by the bridge. The default value of TOST after power-up is 0.
- Whenever the bridge forwards a Split Request of size NST, the bridge increments TOST by the size NST.
- Before the bridge can forward the next pending Split Request, it must check whether the request is allowed to issue based on the expression above. If the value of TOST + NST is greater than STCL, the bridge must wait until enough Split Completions drain out of the bridge to satisfy the expression before forwarding the next Split Request. If TOST is 0, the bridge must forward the transaction regardless of its size. This guarantees that a Sequence of a size larger than the bridge's Split Transaction capacity is forwarded when the bridge is empty.
- When the bridge initiates a Split Completion transaction, it decrements TOST by 1 each time the Split Completion crosses an ADB (making one ADQ-size buffer available for another Split Completion). If the Sequence ends with a Split Completion Message, the bridge decrements TOST according to the starting address and byte count of the rest of the Sequence (included in the Split Completion Message). TOST must not decrement below zero. (Synchronizes to the actual commitment level if STCL used to be set to FFFFh. See implementation note above.)

If the bridge mixes I/O and configuration read and write completions in the same buffer area with memory read completions, the algorithm applies to all Split Transactions forwarded by the bridge. If the bridge segregates memory read transactions from the rest, the algorithm applies only to memory read transactions. Note that in such a bridge, the Split Transaction Control registers apply only to memory reads.

For good performance and scalability, it is assumed that the maximum programmable size read request that a device is programmed to be allowed to issue is set to a value significantly smaller (e.g., 1/4) than the total read completion capacity of any PCI-X bridge above the adapter.

8.4.2.2. Immediate Completion by the Completer

A PCI-X bridge forwarding a Split Request must be prepared for the completer to complete the transaction immediately (i.e., execute the transaction as an Immediate Transaction rather than a Split Transaction) or for the transaction to end with Master-Abort.

If the completer completes the transaction immediately, the bridge must create a Split Completion transaction to return to the requester. (Note that this differs from the case in which the completer responds with Split Response. In that case, the completer creates the Split Completion and the bridge simply forwards it.) When the bridge creates the Split Completion, the bridge creates the Split Completion address and Completer Attributes. It creates the Split Completion address from the original request the same way a completer would (see Section 2.10.3). For the Completer Attributes, the bridge creates the Completer ID for the bus on which the immediate completion occurred. If the immediate completion occurred on the primary bus, the bridge supplies the bus number, device number, and function number from its PCI-X Bridge Status register. If the immediate completion occurred on the secondary bus, the bridge supplies the bus number from the Secondary Bus Number register and sets the Device Number and Function Number fields to zero.

Implementation Note: Mixing Immediate and Split Completion

If a PCI-X bridge forwards a burst memory read Sequence and the completer completes a portion of the Sequence immediately, the bridge must create a Split Completion for this portion of the Sequence as described above. If the completer signals Split Response when the bridge continues the Sequence on the destination bus, the completer creates the Split Completion for the remainder of the Sequence.

In this case, the Split Completion for the first portion of the Sequence uses a Completer ID created by the bridge (as described above). Furthermore, the bridge would be permitted to set the Byte Count Modified bit in the Completer Attributes and modify the byte count of this Split Completion to disconnect it at the first ADB of the Sequence. The Split Completion for the remainder of the Sequence uses the Completer ID of the completer. Since the continuation of the Sequence starts on an ADB following a range that the completer completed as an Immediate Transaction, the completer is not permitted to set the Byte Count Modified bit or use a byte count other than the full remaining byte count of the Sequence. (See Section 2.10.2.) The bridge transaction ordering rules require the bridge to return the two portions of the Sequence to the requester in address order (see Section 8.4.4).

Implementation Note: Buffering Data from Single Data Phase Disconnection

If a PCI-X bridge forwards a burst read request and the completer on the destination bus signals Single Data Phase Disconnect, the bridge must continue the Sequence on the destination bus and accumulate data phases at least up to the next ADB (or until the byte count is satisfied or an error occurs) before it can create the Split Completion and forward it to the requester. The process of accumulating data phases for the Split Completion generally requires multiple transactions on the destination bus. If the bridge is designed not to allow one Split Completion to pass another (i.e., Rows D and E, column 5 in Table 8-3 are implemented as “No”), the performance of other efficient Split Completion transactions is degraded by this slow Single Data Phase Disconnect process.

PCI-X bridges can generally avoid this performance problem by accumulating Single Data Phase Disconnect data phases in a separate buffer, or by otherwise keeping them from blocking other Split Completions in the general Split Completion buffer area until enough data phases from the Single Data Phase Disconnect have accumulated to reach an ADB.

If the Split Request is a write transaction and the completer completes it immediately, the bridge also creates a Split Completion Message (see Section 2.10.6.1) for the data phase of the Split Completion.

Transactions that end with Master-Abort or Target-Abort have similar requirements described in Sections 8.7.1.5 and 8.7.1.6 respectively.

8.4.2.3. Split Request Capacity Recommendations

A PCI-X bridge is required to accept a minimum of one Split Request at a time in both directions, but implementations are encouraged to accept more to improve performance.

Implementation Note: Optimum Size Split Request Buffer

The optimum size of the buffer a PCI-X bridge uses to store Split Requests before they are forwarded is influenced by several factors. If the buffer is too small, in some cases, the buffer empties (underruns) on the completer side before other requesters are able to issue their next request. However, if the bridge keeps requests in strict order and the buffer is large and fills with long requests, the latency for a new request (even a short one) is increased by the long requests enqueued in front of it.

The optimum number of Split Requests that a PCI-X bridge should buffer when requests are not being forwarded (because of lack of resources in the path toward the completer) is the minimum necessary to avoid buffer underrun when requests start flowing again. In most systems, the optimum buffer capacity is approximately four Split Requests.

8.4.3. Connecting PCI-X and Conventional PCI Interfaces

This section provides requirements for the translation of commands and protocol between conventional and PCI-X interfaces. In all cases, an interface of a PCI-X bridge that is operating in conventional mode must meet the requirements of PCI 2.2 and Bridge 1.1. If both interfaces of a PCI-X bridge are operating in conventional mode, the bridge requirements are completely specified by PCI 2.2 and Bridge 1.1. The following list summarizes some of the requirements of translating between these two interfaces:

- Conversion between PCI-X protocol and conventional PCI protocol.
- Translation between PCI-X commands and conventional PCI commands.
- Conversion of **AD[1::0]** as appropriate for the command.
- The byte count and other attributes must be created for transactions translated to PCI-X.
- Conversion between Split Transactions and Delayed Transactions.
- PCI-X data parity error recovery capabilities are not available for devices on a bus in conventional mode. Data parity errors on a conventional interface must be serviced with conventional means.

8.4.3.1. Conventional Requester, PCI-X Completer

8.4.3.1.1. Conventional PCI to PCI-X Command Translation and Byte Count Generation

Table 8-1 summarizes the command translation requirements from a conventional PCI transaction to a PCI-X transaction.

Table 8-1: Conventional PCI to PCI-X Command Translation

Conventional PCI Command	PCI-X Command
I/O Read	I/O Read
I/O Write	I/O Write
Configuration Read	Configuration Read
Configuration Write	Configuration Write
Memory Read	Memory Read DWORD or Memory Read Block
Memory Read Line	Memory Read Block
Memory Read Multiple	Memory Read Block
Memory Write	Memory Write or Memory Write Block
Memory Write and Invalidate	Memory Write Block

Conventional I/O and configuration transactions that cross the bridge translate to the same command on the PCI-X interface. PCI-X I/O transactions are limited to a single DWORD, so the PCI-X bridge must disconnect the conventional requester after each data phase.

The bridge must translate the conventional Memory Read command to either the Memory Read DWORD or Memory Read Block PCI-X command. If the requester is 32 bits wide and deasserts **FRAME#** when it asserts **IRDY#** (indicating the transaction has only a single data phase), the most efficient PCI-X command to use is Memory Read DWORD. The length of the conventional transaction is not known in any other case, so the PCI-X bridge must implement the same prefetch algorithms used by conventional PCI bridges. Such prefetch algorithms are beyond the scope of the PCI-X definition. If the PCI-X bridge prefetches more than a single DWORD, it must use the Memory Read Block command. If a Memory Read Block command is used, the byte count is controlled by the bridge's prefetch algorithm.

The bridge must translate the conventional Memory Read Line and Memory Read Multiple commands to the PCI-X Memory Read Block command. The byte count for this command is controlled by the bridge's prefetch algorithm which is beyond the scope of the PCI-X definition.

The bridge must buffer memory write transactions from its conventional interface and count the number of bytes to be forwarded to the PCI-X interface. If the conventional transaction uses the Memory Write command and some byte enables are deasserted, the bridge must use the PCI-X Memory Write command. If the conventional command is Memory Write and all the byte enables are asserted, the bridge is permitted to use either the Memory Write or the Memory Write Block PCI-X command. If the conventional transaction uses the Memory Write and Invalidate command, the bridge must use the PCI-X Memory Write Block command.

8.4.3.1.2. Delayed Transaction to Split Transaction Conversion

If the PCI-X bridge forwards a transaction other than a memory write from a conventional requester to a PCI-X completer, the bridge must follow Delayed Transaction rules on the requester side and Split Transaction rules on the completer side.

All of the Delayed Transaction requirements specified in Bridge 1.1 apply to the transactions the PCI-X bridge forwards from its conventional interface. For example, the bridge must terminate all these transactions with Retry, store the address, command, etc., and enqueue a Delayed Request. When the bridge has finished the request on the destination bus, the bridge enqueues a Delayed Completion. The next time the requester repeats the transaction, the bridge supplies the Delayed Completion.

All the Split Transaction rules of the PCI-X definition apply to the transactions the bridge initiates on its PCI-X interface.

Transactions that originate on a conventional interface of a bridge follow the conventional ordering and deadlock-avoidance rules shown in PCI 2.2 and Bridge 1.1. All of the bypass cases required to avoid deadlock are the same for conventional PCI and PCI-X, so the translation introduces no additional requirement to avoid deadlocks. The Relaxed Order attribute is never set for transactions the bridge translates from conventional PCI, so the special case for Split Transactions in PCI-X bridges that applies only when this bit is set (Row D, Column 2b in Table 8-3) does not apply to these transactions. All other ordering-rule requirements for transactions in the PCI-X environment are the same as (or are more conservative than) the conventional PCI requirements.

8.4.3.1.3. Conventional PCI to PCI-X Attribute Creation

If a PCI-X bridge forwards any transaction from a conventional requester to a PCI-X completer, the bridge must create Requester Attribute bits for the PCI-X transaction. Generation of the byte count is described in Section 8.4.3.1.1. The bridge uses the bus number for its conventional interface (from either the Primary Bus Number register or the Secondary Bus Number register) and sets the Device Number and Function Number fields to 0. (When the Split Completion returns to the bridge, the bridge forwards it to the conventional requester based on the bus number in the Split Completion address, the same as it does for all other cases. The Device Number and Function Number fields in the Split Completion address are ignored in this case.)

The bridge is permitted to assign Tag numbers to these transactions using any algorithm. For example, if the bridge enqueues multiple Delayed Transactions on the conventional interface, the Tag could be assigned according to the Delayed Transaction with which it is associated.

The bridge must never set the Relaxed Order or No Snoop attribute bits on transactions forwarded from a conventional bus.

8.4.3.2. PCI-X Requester, Conventional Completer

8.4.3.2.1. PCI-X to Conventional PCI Command Translation

Table 8-2 summarizes the translation requirements from a PCI -X command to a conventional PCI command.

Table 8-2: PCI-X to Conventional PCI Command Translation

PCI-X Command	Conventional PCI Command
I/O Read	I/O Read
I/O Write	I/O Write
Configuration Read	Configuration Read
Configuration Write	Configuration Write
Memory Read DWORD	Memory Read
Memory Read Block	Memory Read or Memory Read Line or Memory Read Multiple
Memory Write	Memory Write or Memory Write and Invalidate
Memory Write Block	Memory Write or Memory Write and Invalidate

PCI-X I/O and configuration transactions that cross the bridge translate to the same command on the conventional PCI interface.

The bridge must translate a PCI-X Memory Read DWORD command into a conventional Memory Read command.

The bridge must translate a PCI-X Memory Read Block command into one of the three conventional PCI memory read commands based on the byte count and starting address. Following the guidelines in PCI 2.2, if the starting address and byte count are such that only a single DWORD (or less) is being read, the conventional transaction uses the Memory Read command. If the PCI-X transaction reads more than one DWORD but does not cross a cacheline boundary (as indicated by the Cacheline Size register in the conventional Configuration Space header), the conventional transaction uses the Memory Read Line command. If the PCI-X transaction crosses a cacheline boundary, the conventional transaction uses the Memory Read Multiple command.

If all byte enables of a PCI-X Memory Write command are set and the command starts and ends on a cacheline boundary, the PCI-X bridge optionally translates the command either to the Memory Write or Memory Write and Invalidate command on the conventional PCI interface. Otherwise, the PCI-X Memory Write command translates to the conventional Memory Write command.

If a PCI-X transaction using the Memory Write Block command starts and ends on a cacheline boundary, the PCI-X bridge optionally translates the command either to the Memory Write or Memory Write and Invalidate commands on the conventional PCI interface. Otherwise, the PCI-X Memory Write Block command translates to the conventional Memory Write command.

8.4.3.2.2. Split Transaction to Delayed Transaction Conversion

If the PCI-X bridge forwards a transaction other than a memory write from a PCI-X requester to a conventional completer, the bridge must follow Split Transaction rules on the requester side and Delayed Transaction rules on the completer side.

All of the Delayed Transaction requirements specified in Bridge 1.1 apply to the transactions the PCI-X bridge initiates on its conventional interface. For example, the bridge must continue to repeat any transaction terminated with Retry until the target completes it with some other termination.

All the Split Transaction rules of the PCI-X definition apply to the transactions the bridge forwards from its PCI-X interface.

Transactions that originate on a PCI -X interface of a bridge follow the PCI-X ordering and deadlock-avoidance rules shown in Table 8-3. All of the bypass cases required to avoid deadlock are the same for conventional PCI and PCI-X, so the translation introduces no additional requirement to avoid deadlocks. If the bridge executes several Delayed Read Transactions on the conventional interface to collect the data for a single Split Read Request on the PCI-X interface, the read data must be returned to the PCI-X requester in address order. If the Relaxed Order attribute is set, the relaxed order bypass path for PCI-X bridges (Row D, Column 2b in Table 8-3) is permitted (even though the corresponding cases in conventional PCI is not allowed).

8.4.3.2.3. Creating a Split Completion

If a PCI-X bridge forwards any transaction other than a memory write from a PCI-X requester to a conventional completer, the bridge must terminate the transaction on the originating bus with Split Response. After the bridge executes the transaction on the conventional interface, the bridge must create the Split Completion to return to the PCI-X requester.

When the bridge creates the Split Completion, the bridge creates the Split Completion address and Completer Attributes. It creates the Split Completion address from the original request, the same way a PCI-X completer would (see Section 2.10.3). For the Completer Attributes, the bridge creates a Completer ID that partially describes the location of the conventional completer. If the conventional interface is the primary bus, the bridge supplies the bus number from the Primary Bus Number register in the conventional PCI Configuration Space header. If the conventional interface is the secondary bus, the bridge supplies the bus number from the Secondary Bus Number register. In both cases, the bridge sets the Device Number and Function Number fields to zero.

8.4.4. Transaction Ordering and Passing Rules for Bridges

The rules presented in this section apply both to PCI-X bridges (Type 01h header and Base Class 06h, Sub-Class 04h) and to application bridges (Type 00h header, Device Complexity bit in PCI-X Status registers is 1, see Section 7.2.4).

PCI-X introduces two features that affect transaction ordering and passing rules that are not present in conventional PCI. The first new feature is the Relaxed Ordering attribute bit. See Sections 2.5 and 11 for a description of the cases in which this bit is set.

If the Relaxed Ordering attribute bit is set for a read transaction, the completion for that transaction is permitted to pass previously posted memory write transactions traveling in the direction of the completion (Row D, Col 2b in Table 8-3). See Section 11 for more details.

The Relaxed Ordering attribute bit for memory write transactions is used by host bridge but not PCI-X bridges. If the Relaxed Ordering attribute bit is set for a memory write transaction, that transaction is permitted to pass previously posted memory write transactions moving in the same direction in the host bridge (Row A, Col 2b in Table 8-3). In addition, the bytes within that transaction are permitted to be written to system memory in any order. (The bytes must be written to the correct system memory locations. Only the order in which they are written is unspecified). PCI-X bridges must ignore the Relaxed Ordering attribute bit for a memory write transaction and maintain the order of all memory write transactions that cross them.

Implementation Note: Relaxed Write Ordering in Host Bridges and PCI-X Bridges

Host bridges that connect the PCI-X bus to multiple main-memory subsystems benefit greatly from the use of the Relaxed Ordering attribute on memory write transactions. In such systems, an unordered write to main memory is faster because writes to one memory controller are not required to wait for the completion of previous writes to another memory controller. (See Section 11 for additional details.)

PCI-X bridges are not allowed to rearrange the order of memory write transactions for two reasons. First, there would be little benefit to the system in allowing it. Most memory write transactions moving upstream share a common target, the host bridge. If the host bridge cannot accept one memory write, it is likely that it cannot accept any memory writes. Furthermore, the required acceptance rules for devices guarantee that memory write transactions moving downstream are rarely blocked (see Section 2.13).

The second reason the PCI-X definition does not allow PCI-X bridges to rearrange the order of memory write transactions is that some host bridges invalidate ranges of main memory locations based on the starting address and byte count of a write transaction. If a PCI-X bridge were to rearrange two memory write transactions from the same Sequence (same Sequence ID), the second transaction would invalidate the memory locations updated by the first.

The second new feature is Split Transactions. Split Transaction ordering and deadlock-avoidance rules are almost identical to the rules for Delayed Transactions in conventional PCI. The order of transactions is established when they complete. Split Requests can be reordered with respect to other Split Requests. If an initiator requires two Split Transactions to complete in order, the initiator must not issue the second request until the first Split Transaction completes.

Split Completions have the same ordering requirements as Delayed Completions in conventional PCI, except in two cases. First, Split Read Completions with the same Sequence ID (that is, Split Read Completion transactions that originate from the same Split Read Request) must stay in address order (Row D, Col 5b in Table 8-3). The completer must supply the Split Read Completions on the bus in address order, and any intervening bridges must preserve this order. This guarantees that the requester always receives the data in its natural order. Split Read Completions with different Sequence IDs have no ordering restrictions (Row D, Col 5a in Table 8-3, the same as Delayed Read Completions). The second case in which Split Read Completion ordering rules are different from Delayed Read Completion rules is if the Relaxed Ordering bit is set (Row D, Col 2b in Table 8-3) as described above.

Table 8-3 lists the ordering requirements for all Split Transactions and memory write transactions. The columns represent the first of two transactions, and the rows represent the second. The table entry indicates what a bridge operating on both transactions is required to do. The choices are:

- Yes—the second transaction must be allowed to pass the first to avoid deadlock.
- Y/N—there are no requirements. The bridge may optionally allow the second transaction to pass the first or be blocked by it.
- No—the second transaction must not be allowed to pass the first transaction. This is required to preserve strong write ordering.

Table 8-3: Transactions Ordering and Deadlock-Avoidance Rules

Row pass Col.?	Memory Write (Col 2)	Split Read Request (Col 3)	Split Write Request (Col 4)	Split Read Completion (Col 5)	Split Write Completion (Col 6)
Memory Write (Row A)	a) No b) Y/N	Yes	Yes	Yes	Yes
Split Read Request (Row B)	No	Y/N	Y/N	Y/N	Y/N
Split Write Request (Row C)	No	Y/N	Y/N	Y/N	Y/N
Split Read Completion (Row D)	a) No b) Y/N	Yes	Yes	a) Y/N b) No	Y/N
Split Write Completion (Row E)	Y/N	Yes	Yes	Y/N	Y/N

Case-by-case discussion:

A2a For host bridges when the Relaxed Ordering attribute bit is not set and for PCI-X bridges, a memory write transaction must not pass any other memory writes. (Same as conventional PCI.)

A2b For host bridges when the Relaxed Ordering attribute bit is set, that memory write transaction is permitted to pass all previously posted memory writes in the host bridge (not PCI-X bridges). In addition, the data within that transaction is permitted to be re-ordered enroute to system memory.

A3, A4 A memory write transaction must be allowed to pass Split Requests to avoid deadlocks. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

A5, A6 A memory write transaction must be allowed to pass Split Completions to avoid deadlocks. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

B2, C2 Split Requests cannot pass a memory write transaction. This preserves strong write ordering as did the analogous rule for Delayed Requests in conventional PCI.

B3, B4, C3, C4 Split Requests are permitted to be blocked by or to pass other Split Requests. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

B5, B6, C5, C6 Split Requests are permitted to be blocked by or to pass Split Completions. In most PCI-X implementations, Split Requests are managed in separate buffers from Split Completions, so Split Requests naturally pass Split Completions. However, no deadlocks occur if Split Completions block Split Requests.

D2a Unless the Relaxed Ordering attribute bit is set, Split Read Completions cannot pass a memory write. This preserves strong write ordering as did the analogous rule for Delayed Completions in conventional PCI.

D2b If the Relaxed Ordering attribute bit is set, that Split Read Completion is permitted to pass a previously posted memory write transaction.

D3, D4, E3, E4 Split Completions must be allowed to pass Split Requests to avoid deadlocks. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

D5a Unless two Split Read Completions are part of the same Sequence (i.e., they have the same Sequence ID), they are allowed to be blocked by or to pass each other. (Split Read Completions with *different* Sequence ID in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

D5b Split Read Completions with the *same* Sequence ID must remain in address order.

D6 Split Read Completions are permitted to be blocked by or to pass Split Write Completions. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

E2 Split Write Completions are permitted to be blocked by or to pass memory write transactions. Such write Sequences are actually moving in the opposite direction and, therefore, have no ordering relationship. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

E5, E6 Split Write Completions are permitted to be blocked by or to pass Split Read Completions and Split Write Completions. (These Split Transactions in PCI-X have the same requirements as Delayed Transactions in conventional PCI.)

8.4.5. Required Acceptance Rules for Bridges

The rules presented in this section apply both to PCI-X bridges (Type 01h header and Base Class 06h, Sub-Class 04h) and to application bridges (Type 00h header, Device Complexity bit in PCI-X Status registers is 1; see Section 7.2.4).

A bridge must never make the acceptance (posting) of a memory write transaction as a target contingent on the prior completion of a non-locked transaction that the bridge initiates on the same bus.

A bridge is permitted to terminate with Retry or Disconnect at Next ADB a memory write transaction that crosses the bridge only if the bridge's locations for storing such transactions are full of previously posted memory write transactions moving in the same direction. Bridges are not subject to the Maximum Completion Time limit that simple devices have for accepting memory write transactions. However, to provide backward compatibility with PCI-to-PCI bridges designed to revision 1.0 of the *PCI-to-PCI Bridge Architecture Specification* (prior to Delayed Transactions), all PCI-X bridges are required to accept memory write transactions regardless of how many previous Split Transactions the bridge has enqueued. (This is analogous to the requirement in PCI 2.2 for conventional bridges to accept memory writes even while executing Delayed Transactions.)

A bridge that is executing one Split Transaction from one interface (i.e., issued a Split Response on that interface) is permitted to terminate with Retry a non-posted transaction on that interface until the previous Split Transaction is complete (i.e., the bridge sent all Split Completion data for the Sequence or a Split Completion Message to the requester). Bridges are permitted to execute a limited number of Split Transactions at a time.

Implementation Note: Retry of a Read Transaction to Flush a Prior Posted Memory Write

PCI 2.2 permits a bridge acting as a target to terminate a read transaction with Retry if the ordering rules require the bridge to initiate a previously posted memory write transaction. This case is not allowed for PCI-X bridges operating in PCI-X mode. In most cases, PCI-X bridges terminate read transactions with Split Response. The ordering rules require the bridge to initiate a previously posted memory write before initiating the Split Completion.

In some cases, the bridge's Split Request resources are consumed with previously enqueued Split Requests. In such cases, the bridge terminates read transactions with Retry until Split Request resources become available. The transaction ordering rules require the bridge to continue to initiate memory writes during this time.

A bridge is permitted to terminate a Split Completion transaction with Retry or Disconnect at Next ADB when its buffers are full for one of following reasons:

1. The contents of the Split Transaction Commitment Limit field is larger than the contents of the Split Transaction Capacity field in the appropriate Split Transaction Control register, allowing the bridge to forward more Split Requests than it has room for Split Completions.
2. A corrupted Split Completion (i.e., a Split Completion whose size or address did not match its Split Request, or a corrupt Requester Bus Number field in the Split Completion address caused it to cross the wrong bridge) crossed the bridge some time since the last rising edge of **RST#**.

Section 8.4.1 describes when bridge buffers are considered full.

8.4.6. Forwarding Memory Write Transactions

As in conventional PCI, PCI-X bridges are required to post memory write transactions that cross the bridge in either direction if space is available in the bridge. The conditions under which the bridge is permitted to terminate a memory write transaction with Retry are specified in Section 8.4.5.

With one exception, a PCI-X bridge's memory write buffers are considered full when less than two ADQs of buffer space are available. In the exception case, a PCI-X bridge is permitted to accept a new memory write transaction with less than two ADQs of buffer space available if that bridge provides alternate means for guaranteeing that it never holds a memory write transaction that is too short to forward correctly, as described below.

A bridge with less than two ADQs of buffer space is not permitted to terminate a memory write transaction on the originating bus with Disconnect at Next ADB if both of the following are true:

- The transaction would otherwise cross the ADB
- Such a disconnection would cause the bridge to hold a portion of the transaction that would occupy less than four data phases on the destination data bus.

Implementation Note: Terminating a Memory Write Transaction with Disconnect at Next ADB

If a memory write Sequence addresses a completer on the other side of a PCI-X bridge, a bridge with less than two ADQs of buffer space for memory write transactions must not signal Disconnect at Next ADB if such a disconnection would cause the bridge to hold a portion of the memory write Sequence that is too small to forward correctly on the destination bus. The problem occurs if the byte count of the Sequence indicates the Sequence extends beyond the next ADB, but the portion of the Sequence that the bridge holds would require less than four data phases on the destination bus. If the bridge attempted to forward such a portion of a memory write Sequence, and the target on the destination bus (completer or bridge) signaled Data Transfer (indicating its ability to accept data beyond the ADB), the bridge would be unable to disconnect the transaction at the ADB. The byte count of the Sequence would indicate to the target that the transaction should continue, but the write data would not be available in the bridge.

Although that PCI-X bridge would be allowed to signal Disconnect at Next ADB for transactions other than the problem case described above, the logic required to select precisely this case is complex. The problem case is a function of the starting address, the width of the destination bus, and the width of the completer. The recommended simpler alternative is to provide a minimum of two ADQs of buffer space for memory write transactions. In such an implementation, the bridge would signal Retry to memory write transactions if the buffer space available in the bridge for memory write data was less than two ADQs. With a minimum of two ADQs of buffer space, a bridge would not signal Disconnect at Next ADB until it reached the end of the second ADQ, thereby eliminating the risk of holding too little of the write data.

Memory write transactions that are part of the same Sequence have the following characteristics:

- They have the same Sequence ID and other attributes (except byte count).
- The address of each transaction increments by the number of bytes in the previous transaction of the Sequence.
- The byte count of each transaction is the total number of bytes remaining in the Sequence.

If both interfaces of a PCI-X bridge are operating in PCI-X mode, and the bridge forwards a memory write Sequence from a requester on one side of the bridge to a completer on the other side of the bridge, the bridge must preserve the integrity of the memory write Sequence on the destination bus. That is, transactions that are part of the same Sequence on one side of the bridge must remain part of the same Sequence on the other side of the bridge. The Sequence on the destination bus must use the same Sequence ID and other attributes (except byte count) as the Sequence on the source bus, and the byte count of each transaction must be the full remaining byte count of the Sequence. (See Section 8.4.3 for the case in which one or the other interface is operating in conventional mode.)

In some cases in which a host bridge or conventional PCI bridge has combined write transactions, a single write Sequence crosses a PCI-X bridge range address. If the starting address of the Sequence addresses a completer on the other side of a bridge but one or more addresses within the Sequence do not, the bridge must do both of the following:

- Forward the portion of the write transaction that addresses the completer on the other side of the bridge. The bridge must modify the byte count of the Sequence on the destination bus to be the number of bytes between the starting address and the address limit of the bridge.
- Disconnect the Sequence on the source bus when it reaches the bridge's address limit.

Implementation Note: Burst Write Sequences that Cross Bridge Boundaries

Burst write Sequences cross a bridge range address as a result of write combining in a host bridge or conventional PCI bridge. The PCI-X bridge requirements in this case effectively undo that combining. By restoring the byte count on the destination bus to the number of bytes between the starting address and the bridge limit, the bridge guarantees that a write Sequence is not initiated on the destination bus with a byte count that will not be completed on that bus. (See Section 2.1 for the requirement for the requester to deliver the full byte count of a burst write Sequence.)

By disconnecting the Sequence on the source bus when it reaches the bridge limit, the bridge allows another completer (or bridge) to claim the remainder of the Sequence on the source bus.

A bridge is permitted to disconnect memory write transactions on the destination bus (e.g., if the bridge's data buffers become empty). Memory write transactions are also subject to being disconnected by the target on the destination bus. However, when the PCI-X bridge continues forwarding a memory write Sequence after a disconnection, it does so with transactions that preserve the integrity of the original Sequence.

Combining memory write transactions that originate from the *same* Sequence is permitted. That is, if two or more memory write transactions are part of the same Sequence on the source bus, the bridge is permitted to combine them into a single transaction on the destination bus, provided that such combining does not violate the bridge ordering rules. (See Section 8.4.4.) Combining memory write transactions that originate from *different* Sequences is *not* permitted.

8.5. Exclusive Access

An exclusive access is one or more Sequences that use the **LOCK#** signal as described in this section to guarantee that other Sequences that share a PCI-X bridge in the path to the completer are not executed until after the exclusive access is complete. A Sequence that is part of an exclusive access is referred to as a locked Sequence.

As in conventional PCI, a host bridge can initiate an exclusive access only to prevent a deadlock for a Sequence that originates on the host bus. As in conventional PCI, PCI -X bridges only propagate an exclusive access downstream (away from the host bridge) and are never allowed to initiate an exclusive access of their own.

Exclusive accesses on a bus segment operating in PCI-X mode are only defined for Sequences that cross a PCI-X bridge or application bridge and are initiated by a host bridge or another PCI-X bridge. If an expansion-bus bridge (e.g., PCI-to-EISA) operating in PCI-X mode must initiate an exclusive access on the PCI bus, the expansion-bus bridge must use sideband signaling or other methods beyond the scope of this specification. All other PCI-X devices must ignore **LOCK#**.

The bridge (host bridge or PCI-X bridge) closest to the completer initiates an exclusive access on the bus with the completer the same as it would if the Sequences crossed an additional bridge on their way to the completer. However, the completer ignores **LOCK#** and executes the Sequences the same as a non-exclusive access.

As in conventional PCI, the PCI -X exclusive access mechanism allows non-exclusive accesses to proceed in the face of exclusive accesses if there is no conflict for a shared resource. This allows the host bridge to extend an exclusive access across several Sequences without interfering with non-exclusive data transfers, such as real-time video, between two other devices on the same bus segment. The mechanism is based on locking only the conventional PCI and PCI-X bridges in the path between the host bridge and the completer and is called a resource lock.

Since upstream exclusive accesses are not supported by the PCI-X definition (except for expansion-bus bridges, which are beyond the scope of this specification), the PCI-X definition assumes that the source bridge is the only device that is permitted to initiate an exclusive access. That is, the source bridge has exclusive use of **LOCK#**. For clarity, only exclusive accesses that flow downstream are described. For example, the source bridge (host bridge or PCI-X bridge) is always described as initiating the exclusive access on its secondary bus. A downstream bridge is described as responding on its primary bus. The direction of flow of an exclusive access from an expansion-bus bridge that addresses main memory is not described, but would be opposite.

The PCI-X definition describes the establishment of lock on a single bus. In this section, the PCI-X definition uses the terms “upstream bridge” and “downstream bridge” to refer to the two bridges that establish lock on a single bus. A PCI-X bridge that is the downstream bridge on its primary bus is the upstream bridge when it forwards the locked Sequence to its secondary bus. To execute an exclusive access, lock state must be established on each PCI bus between the requester and the completer.

The following paragraphs describe the behavior of an upstream bridge and a downstream bridge for an exclusive access. A detailed discussion of how to start, continue, and complete an exclusive access follows the summary of the rules.

Upstream bridge rules for supporting **LOCK#**:

1. All Sequences of a single exclusive access address the same completer.
2. The first transaction of an exclusive access must be a read transaction.
3. **LOCK#** must be asserted the clock[2] following the address phase and kept asserted to maintain control.
4. **LOCK#** must be released if the initial transaction of the exclusive access is terminated with Retry[3]. (Lock was not established.)
5. **LOCK#** must be released whenever a locked Sequence is terminated by Target-Abort or Master-Abort.
6. **LOCK#** must be released between consecutive[4] exclusive accesses.

[2] For a single address cycle, this is the clock after the address phase. For a dual address cycle, this is the clock after the first address phase.

[3] Once lock has been established, the initiator retains ownership of **LOCK#** when terminated with Retry.

7. To release **LOCK#**, the initiator must deassert **LOCK#** for a minimum of one clock while the bus is in the Idle state.

Downstream bridge rules for supporting **LOCK#**:

1. A bridge acting as a target of a transaction locks its primary interface when **LOCK#** is deasserted during the first (or only) address phase and is asserted on the following clock.
2. Once in a locked state (as described in Section 8.5.1), a bridge's primary interface remains locked until both **FRAME#** and **LOCK#** are deasserted on the primary bus, regardless of how transactions are terminated on the primary bus.
3. The bridge is not allowed to accept any new requests from the primary bus while it is in a locked state except from the initiator of the exclusive access (as described in Section 8.5.2). The bridge accepts Split Completions from any initiator while in a locked state.

8.5.1. Starting an Exclusive Access

As in conventional PCI, when a device (host bridge or PCI -X bridge) initiates an exclusive access, it checks the internally tracked state of **LOCK#** before asserting its **REQ#**. When the initiator is granted access to the bus, the initiator is free to start an exclusive access when the current transaction ends. The first transaction of an exclusive access must be a memory read transaction. If the transaction uses a single address cycle, the initiator must assert **LOCK#** on the clock following the address phase. If the transaction uses a dual address cycle, the initiator must assert **LOCK#** on the clock following the first address phase.

There are three cases for the first transaction of an exclusive access, depending upon the termination signaled by the target. Each of these cases is presented separately.

- Retry, Target-Abort, and Master-Abort.
- Other Immediate Transactions (Data Transfer, Single Data Phase Disconnect, Disconnect at Next ADB).
- Split Transaction (Split Response).

If the target terminates the first transaction of an exclusive access with Retry, Target-Abort, or Master-Abort, the initiator terminates the transaction and releases **LOCK#**.

If the target executes the first transaction of an exclusive access as any other Immediate Transaction (target signals Data Transfer, Single Data Phase Disconnect, Disconnect at Next ADB), lock is established on the bus when the target signals its acceptance of the first data phase of the transaction. Since a PCI-X bridge is required to complete all memory read transactions as Split Transactions (if the bus is operating in PCI-X mode), this case occurs only if the target is the completer. Once lock is established, the upstream bridge continues to assert **LOCK#** even after the end of the first locked transaction.

Figure 8-1 illustrates starting an exclusive access with an Immediate Transaction. **LOCK#** is deasserted during the address phase (or first address phase for dual address cycle) and asserted one clock later (clock 4) to start the exclusive access.

[4] Consecutive refers to back-to-back exclusive accesses and not a continuation of the current exclusive access.

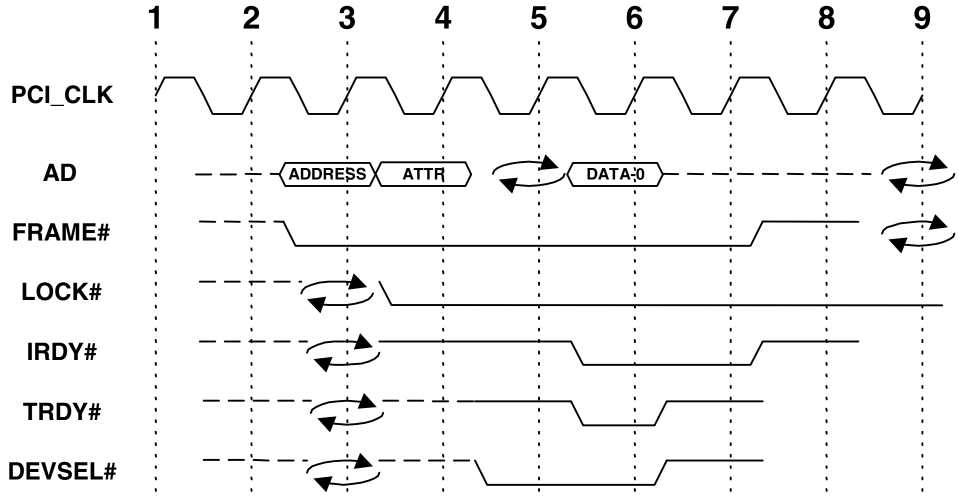


Figure 8-1: Starting an Exclusive Access with an Immediate Transaction

Starting an exclusive access with a Split Transaction is similar to starting an exclusive access with a Delayed Transaction in conventional PCI. If the first transaction of an exclusive access is executed as a Split Transaction, the upstream bridge continues to assert the **LOCK#** signal even after the target (completer or downstream bridge) signals Split Response. This condition is referred to as *initiator-lock*. In the initiator-lock state the upstream bridge continues to accept upstream transactions.

In the initiator-lock state the upstream bridge is permitted to initiate other unlocked downstream transactions, including Split Completions, by keeping **LOCK#** asserted throughout the transaction (including the first address phase, see Section 8.5.3). However, the upstream bridge must not depend upon such transactions completing until after the exclusive access. Otherwise a deadlock could occur, since such an unlocked transaction could address a completer on the other side of a locked downstream bridge and be terminated with Retry by the bridge. (See Section 8.5.3.)

The downstream bridge locks its primary interface when it terminates the first locked read request with Split Response, even though no data has transferred. As in conventional PCI, this condition is referred to as *target-lock* because only the downstream bridge's primary target interface is locked. A downstream bridge acting as a target for an exclusive accesses must latch **LOCK#** during the first (or only) address phase. Otherwise, it cannot determine if the access is locked when decode completes. If the bus is operating in PCI-X mode, a downstream bridge enters the target-lock state if and only if **LOCK#** is deasserted during the first (or only) address phase and is asserted on the next clock, and the downstream bridge terminates the transaction with Split Response.

While in the target-lock state, the downstream bridge enqueues no new requests on the primary interface and terminates them with Retry. The downstream bridge is permitted to accept Split Completions from its primary interface and to initiate transactions on its primary interface while in the target-lock state. The downstream bridge executes all previously enqueued requests flowing downstream before forwarding the first locked read request of an exclusive access to the secondary bus. The downstream bridge locks its secondary interface when it repeats the process described above and establishes lock as the upstream bridge on its secondary bus.

If the Split Completion for the locked read request is a Split Completion Message that indicates an error occurred (Split Completion Error bit set in the completer attributes, see Section 2.10.4), the upstream bridge exits the initiator-lock state and releases **LOCK#**.

Otherwise, the initiator-lock state on the upstream bridge and the target-lock state on the downstream bridge both become *full-lock* states when the upstream bridge accepts at least one data phase of the Split Completion transaction for the locked read request. Lock is established on the bus when the upstream bridge enters the full-lock state.

Figure 8-2 illustrates starting an exclusive access with a Split Transaction.

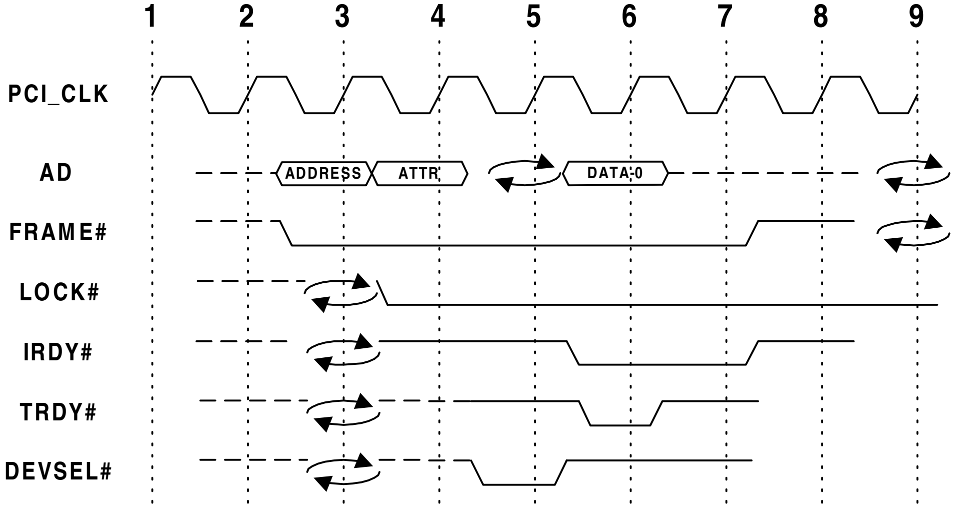


Figure 8-2: Starting an Exclusive Access with a Split Transaction

Once lock is established on the bus, the upstream bridge keeps **LOCK#** asserted until either the exclusive access completes or an error (Master-Abort, Target-Abort, or a Split Completion Message indicating an error) causes an early termination. Target termination of Retry is allowed after lock is established. If a locked transaction from the upstream bridge is terminated by the target (downstream bridge or completer) with Retry after lock has been established, the target is indicating it is currently busy and unable to complete the requested data phase. Lock remains established and both the upstream and downstream bridges remain in the full-lock state. The target accepts the transaction when it is not busy.

While lock is established, the upstream bridge does not accept any upstream transactions, except Split Completions. The upstream bridge terminates all other upstream transactions with Retry. While lock is established, the downstream bridge must only accept requests on its primary interface if **LOCK#** is deasserted during the first (or only) address phase (which indicates that the transaction is a continuation of the exclusive access by the upstream bridge), or if the transaction is a Split Completion. Otherwise, the downstream bridge terminates all transactions (other than Split Completions) with Retry on its primary bus. A downstream bridge remains in the locked state until both **FRAME#** and **LOCK#** are deasserted on the primary bus.

Note that a bridge that forwards a locked Sequence controls **LOCK#** on its secondary bus but not its primary bus. The host bridge that initiates the Sequence controls **LOCK#** on its PCI bus. If a locked bridge forwards a transaction upstream (including a Split Completion associated with a locked Split Request), the primary bus **LOCK#** remains asserted throughout the transaction. (That is, the downstream bridge does not own **LOCK#** on the primary bus interface and, therefore, does not have control of **LOCK#** to deassert it on the primary bus during the address phase.)

All bridges must continue to accept outstanding Split Completions moving in either direction (except as noted in Section 8.4.5) while lock is established.

Non-exclusive accesses to other completers on the same bus segment or behind other unlocked bridges are allowed to execute while **LOCK#** is asserted. However, the requester must not depend upon such transactions completing until after the exclusive access. (Such an unlocked transaction could be blocked by other transactions that cross a locked bridge.)

8.5.2. Continuing an Exclusive Access

A PCI-X initiator continues an exclusive access the same way as in conventional PCI. Figure 8-3 shows an upstream bridge continuing an exclusive access, and the completer (on the same bus segment) executing the transaction as an Immediate Transaction. When the upstream bridge is granted access to the bus, it starts another locked Sequence. **LOCK#** is deasserted during the first (or only) address phase to continue the exclusive access. The completer ignores **LOCK#** and responds to the request. The upstream bridge asserts **LOCK#** on clock 4 to keep lock established beyond the end of the current transaction.

If the upstream bridge is continuing the exclusive access, it continues to assert **LOCK#**. When the upstream bridge completes the exclusive access, it deasserts **LOCK#** after the completion of the last data phase, which occurs on clock 8. Refer to Section 8.5.4 for more information on completing an exclusive access.

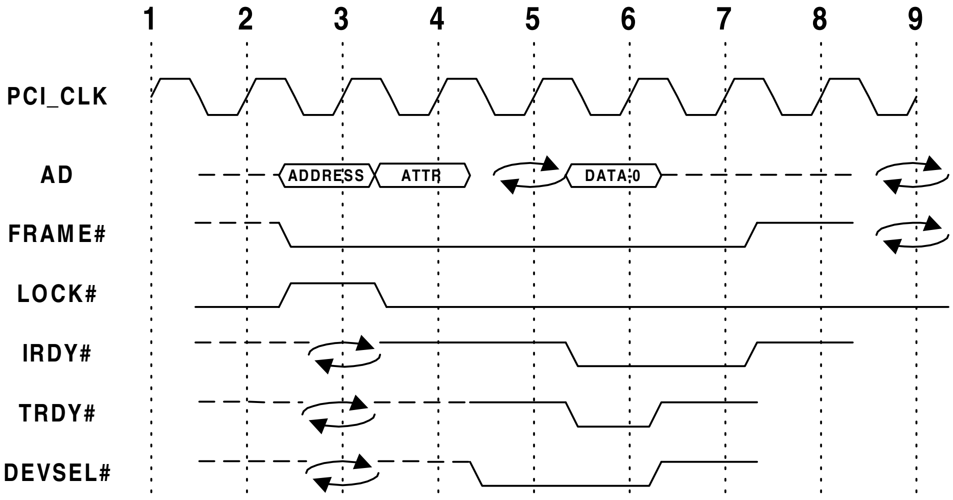


Figure 8-3: Continuing an Exclusive Access, Immediate Transaction

If an upstream bridge continues an exclusive access with a read transaction, and the completer is behind a downstream bridge (and the bus is operating in PCI-X mode), the downstream bridge signals Split Response to the read transaction. Such a transaction would appear the same as shown in Figure 8-3, except the target termination would be Split Response rather than Data Transfer. In such a case, the upstream bridge deasserts **LOCK#** during the first (or only) address phase to continue the exclusive access, and asserts **LOCK#** one clock later to keep lock established beyond the end of the current transaction. The downstream bridge recognizes the request as a continuation of the exclusive access and responds to the request with Split Response. After the downstream bridge completes the transaction on its secondary bus, it initiates the Split Completion on its primary bus. **LOCK#** remains asserted throughout the Split Completion, since **LOCK#** is asserted by the upstream bridge.

8.5.3. Accessing a Locked Bridge

Figure 8-4 shows an initiator trying a non-exclusive, downstream access (other than a Split Completion) to a locked bridge. If the downstream bridge's primary interface is locked (full-lock or target-lock) and **LOCK#** is asserted during the address phase, the downstream bridge terminates the transaction with Retry and no data is transferred.

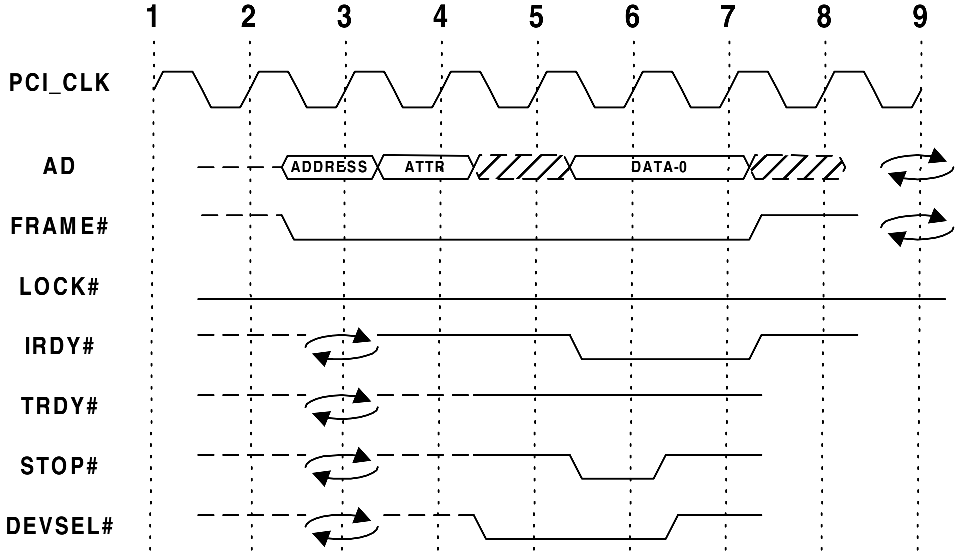


Figure 8-4: Accessing a Locked Downstream Bridge

8.5.4. Completing an Exclusive Access

If the final transaction of an exclusive access is an Immediate Transaction, **LOCK#** is deasserted during the first (or only) address phase and then re-asserted until the transaction terminates successfully (as described in Section 8.5.2). The upstream bridge is permitted to deassert **LOCK#** on any clock after the transaction has completed. However, it is recommended that the upstream bridge deassert **LOCK#** when it deasserts **IRDY#** following the completion of the last data phase of the transaction. In some cases, deasserting **LOCK#** at any other time results in a subsequent transaction being terminated with Retry unnecessarily.

If the final transaction of an exclusive access is a Split Transaction, the upstream bridge keeps **LOCK#** asserted until the end of the Split Completion transaction that terminates the Sequence (i.e., byte count for the request is satisfied or an error occurs). The upstream bridge is permitted to deassert **LOCK#** on any clock after the Split Completion has completed. However, it is recommended that the upstream bridge deassert **LOCK#** when it deasserts **TRDY#** at the end of the Split Completion.

The downstream bridge unlocks itself whenever **LOCK#** and **FRAME#** are deasserted on its primary interface. The downstream bridge deasserts **LOCK#** on its secondary bus on any clock after that, and is recommended to do so as soon as possible to avoid subsequent transactions being terminated with Retry unnecessarily.

If an upstream bridge wants to execute two independent exclusive accesses on the bus, it must ensure a minimum of one clock between exclusive accesses in which both **FRAME#** and **LOCK#** are deasserted. This ensures any downstream bridge locked by the first exclusive access is released prior to starting the second.

8.6. PCI-X Bridge (Type 01h) Configuration Registers

8.6.1. PCI-X Effects on Conventional Bridge Configuration Space Header

PCI-X bridges include the standard Type 01h Configuration Space header defined in Bridge 1.1. In conventional PCI mode, all of these registers function exactly as specified there. If either interface of the device is initialized to PCI-X mode, the requirements for these registers change as follows:

1. Base Address Registers—If the primary interface is in PCI-X mode and the Base Address registers (BARs) (other than the Expansion ROM Base Address register) request memory resources, the BARs must support 64-bit addressing as described in PCI 2.2. The Prefetchable bit must be set unless the range contains locations with read side effects. (See Section 2.12.1 for more details.)
2. If the primary interface is in PCI-X mode, the Prefetchable Memory Base and Limit registers and Prefetchable Base and Limit Upper 32 Bits registers are required.
3. Secondary Bus Number—System configuration software must not change the value in the Secondary Bus Number register while secondary devices have incomplete Split Transactions anywhere in the system. This is generally done by changing the Secondary Bus Number registers only when the system is being initialized (before device drivers load), or after all devices have been quiesced (for a hot-plug operation), or the secondary **RST#** signal from the bridge is asserted. After the Secondary Bus Number register is changed, system configuration software must execute at least one Configuration Write transaction to each device on the bridge's secondary bus. (This initializes the Bus Number registers in the secondary devices. See Sections 2.7.2.2 and 7.2.4.)
4. Latency Timer Register—The default value of the appropriate Latency Timer register is 64 if that interface is in PCI-X mode. (See Section 4.4 for more details.)
5. Cacheline Size Register—The contents of this register are ignored by an interface in PCI-X mode. If one interface is in conventional PCI mode, that interface continues to use this register as defined in PCI 2.2.
6. The Discard Timer control bits in the Bridge Control register are ignored if the appropriate interface is in PCI-X mode. Delayed Transactions are not supported in PCI-X mode.
7. Command Register—If the primary interface is in PCI-X mode, the Command register is restricted as described in Section 7.1.
8. Status Register—If the primary interface is in PCI-X mode, the Status register is restricted as described in Section 7.1.

9. Bridge Control Register—
   Fast Back-to-Back Enable: Ignored by the bridge if the secondary interface is in PCI-X mode.

   Primary Discard Timer: Ignored by the bridge if the primary interface is in PCI-X mode.

   Secondary Discard Timer: Ignored by the bridge if the secondary interface is in PCI-X mode.

   Discard Timer Status: This bit is never set for an interface that is in PCI-X mode.

   Discard Timer **SERR#** *Enable:* Ignored by the bridge in PCI-X mode.

10. Secondary Status Register—If the secondary interface is in PCI-X mode, the Secondary Status register is restricted as described below:
    Fast Back-to-Back Capable: This bit must be set to 0 in PCI-X mode.

    Detected Parity Error and *Master Data Parity Error:* These bits are set as described in Section 5.4.1.

    DEVSEL timing: Indicates conventional **DEVSEL#** timing regardless of the operating mode.

8.6.2. PCI-X Bridge Capabilities List Item

PCI-X bridges include a Type 01h Configuration Space header as defined in Bridge 1.1 and include a PCI-X Capabilities List item as shown in Figure 8-5.

If the bridge is installed on an add-in card, the connection of the **PCIXCAP** pin of the add-in card must be consistent with the presence of this Capability List item in the first bridge on the card. That is, the connection of the **PCIXCAP** pin must indicate the card is capable of operating in PCI-X mode if and only if the PCI-X Capability List item is present in the bridge that connects to the PCI connector of the add-in card (not behind another bridge). If that bridge is part of a multifunction device, all functions within that device must include a PCI-X Capability List item. See Section 7.2 for PCI-X Capability List item for non-bridge functions. See Section 9.10 for the connection of the **PCIXCAP** pin.

31 24 23 16	15 8	7 0
PCI-X Secondary Status	Next Capability	PCI-X Capability ID
PCI-X Bridge Status		
Upstream Split Transaction Control		
Downstream Split Transaction Control		

Figure 8-5: PCI-X Capabilities List Item for a Type 01h Configuration Header

Implementation Note: PCI-X Bridge Registers Optimized for Forwarding Transactions

The PCI-X Bridge Capability List item is structured for forwarding transactions from one interface of the bridge to another. It does not include features found in the PCI-X Command and Status registers of non-bridge devices, such as control of data parity error recovery, relaxed ordering, or number and size of Split Transactions the device is allowed to have outstanding (see Sections 7.2.3 and 7.2.4).

In some applications a bridge device also includes other features that are beyond the scope of this specification, such as a DMA engine. If a bridge with a Type 01h Configuration Space header includes such features in a single device-function, the system has no PCI-X Command register with which to manage the Sequences initiated by the device. In some systems this prevents standard software from controlling recovery from parity errors from the device, or leads to uneven sharing of system resources or suboptimal system performance.

Preferably, the additional features can be implemented as a separate device-function (i.e., a multi-function device with the bridge). This second function would use a Type 00h Configuration Space header and a standard PCI-X Capability List item, which includes the PCI-X Command register and the (Type 00h) PCI-X Status register. In this implementation, the system is able more effectively to manage the additional function and Sequences it initiates.

Implementation Note: PCI-X Capabilities List Item for Application Bridges

Application bridges use a Type 00h Configuration Space header and a PCI-X Capabilities List item defined in Section 7.2. Since the structure of that list item is defined to meet the needs of general PCI-X devices, it does not include registers for bridge-specific functions. Some application bridges require configuration of features similar to those described here for the PCI-X bridges. Such application bridges must implement additional registers in device-specific Configuration Space as required to support their application. These registers must be initialized by the device driver or by local intelligence within the application hardware.

8.6.2.1. PCI-X ID

This register identifies this item in the Capabilities List as a PCI-X register set. It is read-only, returning 07h when read (the same as PCI-X devices with a Type 00h Configuration Space header).

8.6.2.2. Next Capabilities Pointer

This register points to the next item in the Capabilities List as required by PCI 2.2.

8.6.2.3. PCI-X Secondary Status Register

This register reports status information about the secondary bus.

Table 8-4: PCI-X Secondary Status Register

Bit Location	Description
0	**64-bit Device.** (read-only) This bit indicates the width of the bridge's secondary **AD** interface. 0 = The bus is 32 bits wide. 1 = The bus is 64 bits wide.
1	**133 MHz Capable.** (read-only) This bit indicates that the bridge's secondary interface is capable of 133 MHz operation in PCI-X mode. 0 = The maximum operating frequency is 66 MHz. 1 = The maximum operating frequency is 133 MHz.
2	**Split Completion Discarded.** (write 1 to clear) This bit is set if the bridge discards a Split Completion moving toward the secondary bus because the requester would not accept it. See Sections 8.7.1.5 and 8.7.1.6 for details. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = No Split Completion has been discarded. 1 = A Split Completion has been discarded.
3	**Unexpected Split Completion.** (write 1 to clear) This bit is set if an unexpected Split Completion with a Requester ID equal to the bridge's secondary bus number, device number 00h, and function number 0 is received on the bridge's secondary interface. See Section 5.4.5 for more details. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = No unexpected Split Completion has been received. 1 = An unexpected Split Completion has been received.
4	**Split Completion Overrun.** (write 1 to clear) This bit is set if the bridge terminates a Split Completion on the secondary bus with Retry or Disconnect at Next ADB because the bridge buffers are full. It is used by algorithms that optimize the setting of the downstream Split Transaction Commitment Limit register. See Sections 8.4.2.1 and 13.2 for more details. The bridge is also permitted to set this bit in other situations that indicate that the bridge commitment limit is too high. For example, if the bridge stores immediate completion data in the same buffer area as Split Completion data, the completer executes the transaction as an Immediate Transaction, and the bridge disconnects the transaction because the buffers became full. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = The bridge has accepted all Split Completions. 1 = The bridge has terminated a Split Completion with Retry or Disconnect at Next ADB because the bridge buffers were full.

<table>
<tr><th>Bit Location</th><th>Description</th></tr>
<tr><td>5</td><td><b>Split Request Delayed.</b> (write 1 to clear)
This bit is set any time the bridge has a request to forward a transaction on the secondary bus but cannot because there is not enough room within the limit specified in the Split Transaction Commitment Limit field in the Downstream Split Transaction Control register. It is used by algorithms that optimize the setting of the downstream Split Transaction Commitment Limit register. See Sections 8.4.2.1 and 13.2 for more details. Once set, the bit remains set until software writes a 1 to this location.
0 = The bridge has not delayed a Split Request.
1 = The bridge has delayed a Split Request.</td></tr>
<tr><td>8-6</td><td><b>Secondary Clock Frequency.</b> (read-only)
This register enables configuration software to determine to what mode and (in PCI-X mode) what frequency the bridge set the secondary bus the last time secondary <b>RST#</b> was asserted. This is the same information the bridge used to create the PCI-X initialization pattern on the secondary bus the last time secondary <b>RST#</b> was asserted.
<table>
<tr><th>Reg</th><th>Max Clock Frequency (MHz) (ref)</th><th>Minimum Clock Period (ns)</th></tr>
<tr><td>0</td><td>conventional mode</td><td>N/A</td></tr>
<tr><td>1</td><td>66</td><td>15</td></tr>
<tr><td>2</td><td>100</td><td>10</td></tr>
<tr><td>3</td><td>133</td><td>7.5</td></tr>
<tr><td>4</td><td>reserved</td><td>reserved</td></tr>
<tr><td>5</td><td>reserved</td><td>reserved</td></tr>
<tr><td>6</td><td>reserved</td><td>reserved</td></tr>
<tr><td>7</td><td>reserved</td><td>reserved</td></tr>
</table>
An equivalent feature is required for host bridges.</td></tr>
<tr><td>15-9</td><td>Reserved</td></tr>
</table>

8.6.2.4. PCI-X Bridge Status Register

This register identifies the capabilities and current operating mode of the bridge on its primary bus as listed in the following table.

Table 8-5: PCI-X Bridge Status Register

Bit Location	Description
2-0	**Function Number.** (read-only) This register is read for diagnostic purposes only. It indicates the number of this function; i.e., the number in the Function Number field (**AD[10::08]**) of the address of a Type 0 configuration transaction to which this bridge responds. The bridge uses the Bus Number, Device Number, and Function Number fields to create the Completer ID when responding with a Split Completion to a read of an internal bridge register. These fields are also used for cases when one interface is in conventional mode and the other is in PCI-X mode (see Sections 8.4.3.1.3 and 8.4.3.2.3).
7-3	**Device Number.** (read-only) This register is read for diagnostic purposes only. It indicates the number of this device; i.e., the number in the Device Number field (**AD[15::11]**) of the address of a Type 0 configuration transaction that is assigned to this bridge by the connection of the system hardware. The bridge uses this number as described for the Function Number field above. Each time the bridge is addressed by a Configuration Write transaction, the bridge must update this register with the contents of **AD[15::11]** of the address phase of the Configuration Write, regardless of which register in the bridge is addressed by the transaction. The bridge is addressed by a Configuration Write transaction if all of the following are true: 1. The transaction uses a Configuration Write command. 2. **IDSEL** is asserted during the address phase. 3. **AD[1::0]** are 00b (Type 0 configuration transaction). 4. **AD[10::08]** of the configuration address contain the appropriate function number. State after **RST#** is 1Fh.
15-8	**Bus Number.** (read-only) This register is read for diagnostic purposes only. It is an additional address from which the contents of the Primary Bus Number register in the Type 01h Configuration Space header is read. The bridge uses this number as described for the Function Number field above.

Bit Location	Description
16	**64-bit Device.** (read-only) This bit is used by system management software to assist the user in identifying the best slot for an add-in card. If the bridge is part of a device that is installed on an add-in card and connects directly to the PCI connector (not through another bridge), this bit is set if and only if all of the following are true: 1. The bridge function implements a 64-bit **AD** interface on its primary side. 2. The device implements a 64-bit **AD** interface on its primary side. 3. The add-in card implements a 64-bit PCI connector. This requirement is independent of the width of the slot in which the card is installed. If the bridge is subordinate to another bridge on an add-in card, or if the bridge is installed on the system board (not in a slot), this bit is permitted to have any value. 0 = The bus is 32 bits wide. 1 = The bus is 64 bits wide.
17	**133 MHz Capable.** (read-only) This bit is used by system management software to assist the user in identifying the best slot for an add-in card. It is also used in some hot-plug systems to determine whether an add-in card would function properly if the bus were changed to PCI-X 133 mode. If the bridge is installed on an add-in card and connects directly to the PCI connector (not through another bridge), this bit indicates whether the bridge's primary interface is capable of 133 MHz operation in PCI-X mode. The connection of the card's **PCIXCAP** pin (see Section 6.2) must be consistent with this bit. If the bridge is subordinate to another bridge on an add-in card, or if the bridge is installed on the system board (not in a slot), this bit is permitted to have any value. All functions within a multi-function device have the same value for this bit. 0 = The maximum operating frequency is 66 MHz. 1 = The maximum operating frequency is 133 MHz.
18	**Split Completion Discarded.** (write 1 to clear) This bit is set if the bridge discards a Split Completion because the requester on the primary bus would not accept it. See Sections 8.7.1.5 and 8.7.1.6 for details. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = No Split Completion has been discarded. 1 = A Split Completion has been discarded.
19	**Unexpected Split Completion.** (write 1 to clear) This bit is set if an unexpected Split Completion with a Requester ID equal to the bridge's primary bus number, device number, and function number is received on the bridge's primary bus. See Section 5.4.5 for more details. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = No unexpected Split Completion has been received. 1 = An unexpected Split Completion has been received.

Bit Location	Description
20	**Split Completion Overrun.** (write 1 to clear) This bit is set if the bridge terminates a Split Completion on the primary bus with Retry or Disconnect at Next ADB because the bridge buffers are full. It is used by algorithms that optimize the setting of the upstream Split Transaction Commitment Limit register. See Sections 8.4.2.1 and 13.2 for more details. The bridge is also permitted to set this bit in other situations that indicate that the bridge commitment limit is too high. For example, if the bridge stores immediate completion data in the same buffer area as Split Completion data, the completer executes the transaction as an Immediate Transaction, and the bridge disconnects the transaction because the buffers became full. Once set, this bit remains set until software writes a 1 to this location. State after **RST#** is 0. 0 = The bridge has accepted all Split Completions. 1 = The bridge has terminated a Split Completion with Retry or Disconnect at Next ADB because the bridge buffers were full.
21	**Split Request Delayed.** (write 1 to clear) This bit is set any time the bridge has a request to forward a transaction on the primary bus but cannot because there is not enough room within the limit specified in the Split Transaction Commitment Limit field in the Upstream Split Transaction Control register. It is used by algorithms that optimize the setting of the upstream Split Transaction Commitment Limit register. See Sections 8.4.2.1 and 13.2 for more details. Once set, the bit remains set until software writes a 1 to this location. 0 = The bridge has not delayed a Split Request. 1 = The bridge has delayed a Split Request.
31-22	Reserved

Implementation Note: The Primary Bus Number and PCI-X Bus Number Registers

A PCI-X bridge’s primary bus number is initialized in one location but can be read from two. The value is initialized in the Primary Bus Number in the standard Type 01h Configuration Space header and can be read both there and in the PCI-X Capabilities List item in the Bus Number register. The second “read-only” location is provided to keep the programming model of the PCI-X Bridge Status register consistent with the PCI-X Status register for other PCI-X devices.

8.6.2.5. Upstream Split Transaction Register

This register controls behavior of the bridge buffers for forwarding Split Transactions from a secondary bus requester to a primary bus completer.

Table 8-6: Upstream Split Transaction Register

Bit Location	Description
15-0	**Split Transaction Capacity.** (read-only) Some bridges store Split Completions for memory reads in a separate buffer from Split Completions for I/O and configuration reads and writes. For such bridges, this register indicates the size of the buffer (in number of ADQs) for storing Split Completions for memory reads for requesters on the secondary bus addressing completers on the primary bus. If the bridge stores Split Read Completions in the same buffer as other Split Completions, this register indicates the size of this buffer in units of ADQs.
31-16	**Split Transaction Commitment Limit.** (read-write) Some bridges store Split Completions for memory reads in a separate buffer from Split Completions for I/O and configuration reads and writes. For such a bridge, this register indicates the cumulative Sequence size for all memory read transactions forwarded by the bridge from requesters on the secondary bus addressing completers on the primary bus. (See Section 8.4.2.1 for a detailed discussion of Split Transaction commitment.) If the bridge stores Split Read Completions in the same buffer as other Split Completions, this register indicates the size of all upstream Split Transactions of these types that the bridge is permitted to commit to at one time. This register indicates the size of the commitment limit in units of ADQs. Software is permitted to program this register to any value greater than or equal to the contents of the Split Transaction Capacity register. A value less than the contents of the Split Transaction Capacity register causes unspecified results. If this register is set to FFFFh, the bridge is permitted to forward all Split Request of any size regardless of the amount of buffer space available. Software is permitted to change this register at any time. The most recent value of the register is used each time the bridge forwards a Split Transaction. If the register value is set to FFFFh, the bridge does not track the outstanding commitment. If the register is later set to something else, the bridge does not accurately track outstanding commitments until all outstanding commitments complete. Systems that require accurate limitation of Split Transactions must never set this register to FFFFh, or they must quiesce all devices that initiate traffic that crosses the bridge in this direction after the register setting is changed from FFFFh. An algorithm for setting this register is not specified. System software is permitted to use any method for selecting the value for this register. Individual devices and device drivers are not permitted to change the value of this register except under control of a system-level configuration routine. See Section 13.2 for more details and setting recommendations. State after **RST#** is the same as the Split Transaction Capacity register.

8.6.2.6. Downstream Split Transaction Register

This register controls behavior of the bridge buffers for forwarding Split Transactions from a primary bus requester to a secondary bus completer.

Table 8-7: Downstream Split Transaction Register

Bit Location	Description
15-0	**Split Transaction Capacity.** (read-only) Some bridges store Split Completions for memory reads in a separate buffer from Split Completions for I/O and configuration reads and writes. For such a bridge, this register indicates the size of the buffer (in number of ADQs) for storing Split Completions for memory reads for requesters on the primary bus addressing completers on the secondary bus. If the bridge stores Split Read Completions in the same buffer as other Split Completions, this register indicates the size of this buffer in units of ADQs.
31-16	**Split Transaction Commitment Limit.** (read-write) Some bridges store Split Completions for memory reads in a separate buffer from Split Completions for I/O and configuration reads and writes. For such bridges, this register indicates the cumulative Sequence size for all memory read transactions forwarded by the bridge from requesters on the primary bus addressing completers on the secondary bus. (See Section 8.4.2.1 for a detailed discussion of Split Transaction commitment.) If the bridge stores Split Read Completions in the same buffer as other Split Completions, this register indicates the size of all downstream Split Transactions of these types that the bridge is permitted to commit to at one time. This register indicates the size of the commitment limit in units of ADQs. Software is permitted to program this register to any value greater than or equal to the contents of the Split Transaction Capacity register. A value less than the contents of the Split Transaction Capacity register causes unspecified results. If this register is set to FFFFh, the bridge is permitted to forward all Split Request of any size regardless of the amount of buffer space available. Software is permitted to change this register at any time. The most recent value of the register is used each time the bridge forwards a Split Transaction. If the register value is set to FFFFh, the bridge does not track the outstanding commitment. If the register is later set to something else, the bridge does not accurately track outstanding commitments until all outstanding commitments complete. Systems that require accurate limitation of Split Transactions must never set this register to FFFFh, or they must quiesce all devices that initiate traffic that crosses the bridge in this direction after the register setting is changed from FFFFh. An algorithm for setting this register is not specified. System software is permitted to use any method for selecting the value for this register. Individual devices and device drivers are not permitted to change the value of this register except under control of a system-level configuration routine. See Section 13.2 for more details and setting recommendations. State after **RST#** is the same as the Split Transaction Capacity register.

8.7. PCI-X Bridge Error Support

Some of the PCI-X bridge error support requirements vary depending upon the mode (PCI-X or conventional PCI) of the bridge's interface on the side from which the transaction originated. Requirements for these two cases are presented separately below.

The originating side of the bridge is the bridge interface that responds as a target to the transaction. For Split Requests and memory writes, this is the side closest to the requester. For Split Completions, this is the side closest to the completer. The destination side is opposite the originating side.

If a PCI-X bridge detects a data parity error, in most cases, it forwards the transaction with the error (the bridge drives **AD**, **C/BE#**, **PAR64**, and **PAR** on the destination bus exactly as observed on the originating bus, including the parity error, for each data phase). This enables the error to be returned to the requester so the requester can attempt to recover or report the error to the system. In the following discussion, the phrase "drives bad parity" is used to describe this case.

8.7.1. PCI-X Originating Bus

This section describes the bridge error support requirements for transaction that cross the bridge if the originating side of the bridge is operating in PCI-X mode (regardless of the mode of the other side of the bridge).

8.7.1.1. Data Parity Error on an Immediate Read

If the bridge detects a data parity error on the destination bus while forwarding a read transaction that the completer completes immediately, the bridge sets the appropriate error status bits and asserts **PERR#** as described in Section 5.2 for that interface. When the bridge creates the Split Completion and returns it to the requester, the bridge drives bad parity. The read data parity error does not affect the bridge's behavior in any other way. After a data parity error on the destination bus for an immediate read transaction, the bridge continues to fetch data until the byte count is satisfied or the target on the destination bus ends the Sequence in some other way.

8.7.1.2. Data Parity Error on a Non-Posted Write

If the bridge detects a data parity error on the originating bus for a non-posted write transaction that crosses the bridge, the bridge asserts **PERR#** and sets the appropriate error status bits as described in Section 5.4.1 for that interface. The bridge optionally signals either Data Transfer or Split Response for this transaction. If the bridge signals Data Transfer, the bridge discards the transaction and does not forward it. If the bridge signals Split Response, the bridge forwards the transaction and drives bad parity. The bridge must not signal Retry or Target-Abort solely because of a data parity error.

Implementation Note: Discarding or Forwarding a Non-Posted Write with a Data Parity Error

If a non-posted write transaction has a parity error on the originating bus, the bridge is allowed to terminate it with Data Transfer and to discard the transaction for consistency with conventional PCI bridges (which discard non-posted write transactions with data parity errors). However, implementing this option requires the bridge to delay the signaling of Split Response for all non-posted write transactions until write data parity is sampled and checked.

Signaling Split Response before write data parity is checked allows the bridge to respond sooner on all non-posted write transactions. However, once the bridge signals Split Response, it must forward the transaction.

If the bridge observes **PERR#** asserted on the destination bus while forwarding a non-posted write transaction, the bridge sets the appropriate error status bits as described in Section 5.4.1 for that interface. If the target completes the transaction immediately (i.e., signals Data Transfer, Single Data Phase Disconnect, or Disconnect at Next ADB), the bridge generates the appropriate Split Completion Message (see Section 8.8) to report the error to the requester. If the target signals Split Response, the bridge terminates the transaction as it would for a Split Request that did not have an error and takes no further action. (When the Split Completion returns, the bridge forwards it normally.)

8.7.1.3. Data Parity Error on a Split Completion

If the bridge detects a data parity error on the originating bus for a Split Completion other than a Split Completion Message, the bridge asserts **PERR#** and sets the appropriate error status bits as described in Section 5.4.1 for that interface. The bridge then drives bad parity when it forwards the Split Completion. The bridge takes no other action on that data parity error.

If the bridge detects a data parity error on the originating bus for a Split Completion Message, it asserts **SERR#** (if enabled) and discards the transaction.

If the bridge observes **PERR#** asserted on the destination bus while forwarding a Split Completion, the bridge sets the appropriate error status bits as described in Section 5.4.1 for that interface. The bridge takes no other action on that data parity error.

The bridge forwards Split Completion Messages without decoding the message. The bridge forwards Split Completion Messages the same way regardless of whether they indicate normal completion or that some other device detected an error.

8.7.1.4. Data Parity Error on a Posted Write

PCI-X bridge behavior for parity errors on posted write transactions is the same as for conventional PCI bridges. If the bridge detects a data parity error on the originating bus for a posted write transaction that crosses the bridge, the bridge asserts **PERR#** and sets the appropriate error status bits as described in Section 5.4.1 for that interface. The bridge then forwards the posted write transaction and drives bad parity. The bridge takes no other action on that data parity error. If the bridge observes **PERR#** asserted on the destination bus for this transaction, the bridge sets the appropriate error status bits as described in Section 5.4.1 for that interface and takes no further action for this error.

If the bridge observes **PERR#** asserted on the destination bus for a posted write transaction that was error-free on the originating bus, the bridge sets the appropriate error status bits as described in Section 5.4.1 for that interface and asserts **SERR#** (if enabled).

8.7.1.5. Master-Abort

As in conventional PCI, the requirements for a PCI-X bridge that encounters a Master-Abort when forwarding a transaction vary depending on the state of the Master-Abort Mode bit in the Bridge Control register and the type of transaction. As described below, PCI-X bridge requirements differ from conventional bridges in that error conditions that require conventional bridges to signal Target-Abort in most cases require PCI-X bridges to send a Split Completion Message. Furthermore, PCI-X bridge requirements are the same for exclusive and non-exclusive accesses.

If the bridge encounters a Master-Abort on the destination bus for any transaction except memory writes and Split Completions, the bridge sets the appropriate Received Master-Abort status bit (as specified in Bridge 1.1) and creates a Split Completion Message. The Split Completion address and Completer Attributes are created as described for immediate completion in Section 8.4.2.2. The Split Completion Message is created as described in Section 2.10.6 with the PCI-X Bridge Error class code and Master-Abort error message index as described in Section 8.8. If the transaction terminated with Master-Abort is a DWORD transaction, the error Split Completion Message replaces the normal Split Completion for this transaction. The bridge's behavior in such cases is independent of the state of the Master-Abort Mode bit.

Implementation Note: Read Data Values after a Master-Abort Condition

Some system configuration software depends on reading a data value of FFFF FFFFh when Configuration Read transactions encounter a Master-Abort condition. A PCI-X bridge is required by the preceding paragraph to generate a Split Completion Message when any non-posted transaction (including Configuration Read) ends in Master-Abort. Host bridges intended for use with software that depends on a read-data value of FFFF FFFFh after a Master-Abort must decode Split Completion Messages that are PCI-X Bridge class and Master-Abort error index and create the appropriate read-data pattern for the software.

The requirements for PCI-X bridges that encounter Master-Abort conditions on memory write transactions are the same as for conventional PCI bridges (as described in Bridge 1.1). If the bridge encounters a Master-Abort on the destination bus for a posted write transaction, the bridge sets the appropriate Received Master-Abort status bit. The bridge disconnects the transaction as soon as possible on the originating side, if it is still in progress (generally the next ADB, see Section 2.11.2), and discards the entire transaction. If the Master-Abort Mode bit is cleared, the bridge takes no further action on the error. If the Master-Abort Mode bit is set, the bridge asserts **SERR#** (if enabled) on the primary interface.

If the bridge initiates a Split Completion transaction on the primary bus and encounters a Master-Abort, the bridge sets the Received Master-Abort bit in the Status register (as specified in Bridge 1.1) and the Split Completion Discarded bit in the PCI-X Bridge Status register (as specified in Section 8.6.2.4). If the bridge initiates a Split Completion transaction on the secondary bus and encounters a Master-Abort, the bridge sets the Received Master-Abort status bit in the Secondary Status register (as specified in Bridge 1.1) and the Split Completion Discarded bit in the PCI-X Secondary Status register (as specified in Section 8.6.2.3). In both cases, the bridge discards the entire transaction and asserts **SERR#** (if enabled) on the primary interface independent of the state of the Master-Abort Mode bit.

8.7.1.6. Target-Abort

PCI-X bridges signal Target-Abort only if the bridge asserts **DEVSEL#** to claim a transaction but error conditions prevent the bridge from signaling any other termination, for example, a parity error in the address phase. As in conventional PCI, the bridge sets the appropriate status bits.

If the bridge encounters a Target-Abort on the destination bus for any transaction except memory writes and Split Completions, the bridge sets the appropriate Received Target-Abort status bit (as specified in Bridge 1.1) and creates a Split Completion Message. The Split Completion address and attributes are created as described for immediate completion in Section 8.4.2.2. The Split Completion Message is created as described in Section 2.10.6 with the PCI-X Bridge Error class code and Target-Abort error message index as described in Section 8.8. If the transaction was a DWORD transaction, the error Split Completion Message replaces the normal Split Completion for this transaction. If the transaction was a burst, the bridge is permitted to send the error Split Completion Message in lieu of the first Split Completion for this Sequence or any continuation of the Sequence after a disconnection on an ADB. (Unlike conventional PCI, there is no way for a PCI-X bridge to indicate on which data phase the Target-Abort occurred.)

The requirements for PCI-X bridges that encounter Target-Abort conditions on memory write transactions are the same as for conventional PCI bridges (as described in Bridge 1.1). If the bridge encounters a Target-Abort on the destination bus for a posted write transaction, the bridge sets the appropriate Received Target-Abort status bit. The bridge disconnects the transaction as soon as possible on the originating side, if it is still in progress (generally the next ADB, see Section 2.11.2), and discards the entire transaction. The bridge asserts **SERR#** (if enabled) on the primary interface.

If the bridge initiates a Split Completion transaction on the primary bus and encounters a Target-Abort, the bridge sets the Received Target-Abort bit in the Status register (as specified in Bridge 1.1) and the Split Completion Discarded bit in the PCI-X Bridge Status register (as specified in Section 8.6.2.4). If the bridge initiates a Split Completion transaction on the secondary bus and encounters a Target-Abort, the bridge sets the Received Target-Abort status bit in the Secondary Status register (as specified in Bridge 1.1) and the Split Completion Discarded bit in the PCI-X Secondary Status register (as specified in Section 8.6.2.3). In both cases, the bridge discards the entire transaction and asserts **SERR#** (if enabled) on the primary interface.

Implementation Note: Asserting SERR# after a Master-Abort or Target-Abort for a Split Completion

A properly functioning requester in a properly functioning system takes all the data indicated by the byte count of the original Split Request without signaling Target-Abort or allowing a Master-Abort. A Master-Abort or Target-Abort termination of a Split Completion indicates the existence of a serious problem in the system. Such problems can lead to Split Completions from one requester appearing to match outstanding Split Requests from another requester. The bridge must assert **SERR#** in this case to prevent or limit further data corruption in the system.

8.7.2. Conventional PCI Originating Bus

If the originating bus is in conventional PCI mode, PCI -X bridge requirements are the same as described in Bridge 1.1 in all cases except errors that occur in the PCI-X environment and are reported to the bridge in the form of a Split Completion Message.

If the PCI-X bridge forwards a read transaction from a conventional interface to a PCI-X interface and the transaction completes with a Split Completion Message, the bridge completes the transaction normally on the conventional interface and returns read-data of FFFF FFFFh if all of the following are true:

- The Split Completion Message indicates a Master-Abort condition (i.e., PCI-X Bridge class and Master-Abort error index).
- The Master-Abort Mode bit in the Bridge Control register is cleared.
- The read transaction is non-exclusive.

For all other cases in which a read transaction completes with a Split Completion Message, the bridge terminates the transaction on the conventional interface with Target-Abort.

If the PCI-X bridge forwards a non-posted write transaction from a conventional interface to a PCI-X interface and the transaction completes with a Split Completion Message, the bridge completes the transaction normally on the conventional interface in the following two cases:

- The transaction completes with a Split Completion Message that indicates Normal Completion (i.e., Write Completion class and Normal Completion index).
- The transaction completes with a Split Completion Message that indicates Master-Abort (i.e., PCI-X Bridge class and Master-Abort error index), and the Master-Abort Mode bit in the Bridge Control register is cleared, and the write transaction is non-exclusive.

If the Split Completion Message indicates the occurrence of a write-data parity error (i.e., PCI-X Bridge class and Write Data Parity Error index), the bridge asserts **PERR#** and sets the appropriate bits in the Status register when the transaction completes on the conventional interface. For all other cases in which a non-posted write transaction completes with a Split Completion Message, the bridge terminates the transaction on the conventional interface with Target-Abort.

8.8. PCI-X Bridge Error Class Split Completion Message

If a PCI-X bridge forwards a Split Transaction and encounters an error on the destination bus that ends the Sequence, the bridge generates a Split Completion Message with the PCI-X Bridge Error class code (1h) and returns it to the requester in lieu of a normal Split Completion transaction.

The PCI-X Bridge Error class is used by bridges between two PCI buses operating either in PCI-X or conventional mode. Bridges to other buses must not use this message class.

Such error conditions are special cases of Immediate Transactions discussed in Section 8.4.2.2. The rules for creating the Split Completion address and Completer Attributes are described in that section for the case in which the error occurred on a bus segment operating in PCI-X mode. Refer to Section 8.4.3.2.3 for the case in which the error occurred on a conventional bus segment. The Lower Address field in the Split

Completion address is always set to zero and the Byte Count field in the Completer Attributes is always set to four for a Split Completion Message.

The format of the Split Completion Message is specified in Section 2.10.6. The Message Class is 1. The Remaining Byte Count field in the data phase of this Split Completion Message is the number of bytes remaining in this Sequence. (If the bridge has already received some data as an immediate response to previous transactions in the same Sequence, the byte count of the transaction that encountered the error is less than the original request.) The bridge sets all reserved bits in the Split Completion Message to 0.

Table 8-8 shows the index values defined for this message class. All other indexes are reserved.

Table 8-8: PCI-X Bridge Error Messages Indices (Class 1)

Index	Message
00h	**Master-Abort.** The PCI-X bridge encountered a Master-Abort on the destination bus. (See Section 8.7.1.5.)
01h	**Target-Abort.** The PCI-X bridge encountered a Target-Abort on the destination bus. (See Section 8.7.1.6.)
02h	**Write Data Parity Error.** The PCI-X bridge encountered a data parity error on a non-posted write transaction on the destination bus. (See Section 8.7.1.2.)

8.9. Secondary Bus Mode and Frequency Initialization Sequence

A PCI-X bridge places its secondary bus in PCI-X mode based on the capabilities of the devices connected there, independent of the mode of the primary bus. If only one side of a bridge is operating in PCI-X mode, the bridge must translate the protocol between the two buses as described in Section 8.4.3.

This section describes the clock on the secondary bus as if it were generated inside the bridge. Such discussion is not intended to preclude other alternatives such as having the clock generated by a component separate from the bridge.

As in conventional PCI, if primary **RST#** is asserted into a PCI-X bridge, the bridge must clear all of its internal state machines, assert its secondary **RST#**, float the secondary bus control signals (including the ones in the PCI-X initialization pattern), and park the secondary **AD** and **C/BE#** buses in the low logic-level state. At the rising edge of primary **RST#**, a PCI-X bridge latches the frequency and mode of its primary bus. It must then initialize the secondary bus as follows. (Note that for most bridges, the initialization of the secondary clock must wait for the rising edge of primary **RST#** to capture the proper frequency range of the primary clock. Bridges that generate secondary clock independent of primary clock are permitted to perform some of the following steps prior to the rising edge of primary **RST#**.)

1. Sense the states of **PCIXCAP** and **M66EN** for all devices on the secondary bus. See Section 14 for examples of methods for detecting the state of **PCIXCAP**.
2. Select the appropriate mode and clock frequency for the cards present on this bus as described in Section 6.1.2. (The design of the bridge and the electrical length and number of load on the bus determine the actual frequency at which the bus operates in each mode.)

3. If the mode is to be 33 MHz conventional PCI, deassert **M66EN** for all devices on the secondary bus. (This requirement is automatically met if **M66EN** is bused for all devices on the secondary bus.)
4. Assert the appropriate signals for the PCI-X initialization pattern (from Table 6-2) on the secondary bus. Leave the others floating so the pull-up resistors deassert them. (The bridge must not actively deassert the bus control signals while **RST#** is deasserted, because one of the power supply voltages could be out of range.) The timing requirements for this pattern are shown in Section 9.6.
5. Deassert secondary **RST#** to place all devices on the secondary bus in the appropriate mode.

The bridge must generally wait for its clock divider/multiplier and internal PLL to stabilize before the secondary clock is stable. In some implementations, this increases the delay between the time primary **RST#** deasserts and the time the bridge deasserts its secondary **RST#**. In all cases, the bridge must guarantee that secondary **RST#** does not deassert until after the secondary clock is stable at the proper frequency for the length of time specified in Table 9-5.

The PCI-X bridge is also required to apply the PCI-X initialization pattern with the same timing requirement any other time **RST#** deasserts on this bus; e.g., if software sets and clears the Secondary Bus Reset bit in the Bridge Control register specified in Bridge 1.1.

9. Electrical Specification

9.1. DC Specifications

The following table shows the DC specifications for devices operating in PCI-X mode. Conventional 3.3V signaling DC specifications are included for reference.

Table 9-1: DC Specifications for PCI-X Devices

			PCI-X		3.3V Conventional PCI (ref)			
Sym	**Parameter**	**Condition**	**Min**	**Max**	**Min**	**Max**	**Units**	**Notes**
V_{cc}	Supply Voltage		3.0	3.6	3.0	3.6	V	
V_{ih}	Input High Voltage		$0.5V_{cc}$	$V_{cc} + 0.5$	$0.5V_{cc}$	$V_{cc} + 0.5$	V	
V_{il}	Input Low Voltage		-0.5	$0.35V_{cc}$	-0.5	$0.3V_{cc}$	V	
V_{ipu}	Input Pull-up Voltage		$0.7V_{cc}$		$0.7V_{cc}$		V	1
I_{il}	Input Leakage Current	$0 < V_{in} < V_{cc}$		±10		±10	µA	2
V_{oh}	Output High Voltage	I_{out} = -500 µA	$0.9V_{cc}$		$0.9V_{cc}$		V	
V_{ol}	Output Low Voltage	I_{out} = 1500 µA		$0.1V_{cc}$		$0.1V_{cc}$	V	
C_{in}	Input Pin Capacitance			8		10	pF	3
C_{clk}	**CLK** Pin Capacitance		5	8	5	12	pF	
C_{IDSEL}	**IDSEL** Pin Capacitance			8		8	pF	4
L_{pin}	Pin Inductance			15		20	nH	5
I_{Off}	**PME#** input leakage	$V_o \leq 3.6$ V V_{cc} off or floating	–	1	–	1	µA	6

Notes:

1. This specification should be guaranteed by design. It is the minimum voltage to which pull-up resistors are calculated to pull a floated network. Applications sensitive to static power utilization must assure that the input buffer is conducting minimum current at this input voltage.
2. Input leakage currents include hi-Z output leakage for all bi-directional buffers with tri-state outputs.
3. Absolute maximum pin capacitance for a PCI/PCI-X input except **CLK** and **IDSEL**.
4. For conventional PCI only, lower capacitance on this input-only pin allows for non-resistive coupling to **AD[xx]**. PCI-X configuration transactions drive the **AD** bus four clocks before **FRAME#** asserts (see Section 2.7.2.1).
5. For conventional PCI, this is a recommendation not an absolute requirement. For PCI-X, this is a requirement.
6. This input leakage is the maximum allowable leakage into the **PME#** open drain driver when power is removed from V_{cc} of the component. This assumes that no event has occurred to cause the device to attempt to assert **PME#**.

9.2. AC Specifications

The output drive characteristics for devices operating in PCI-X mode over the full range of output voltages are shown Table 9-2 and Table 9-3. Conventional PCI 66 MHz values are included for reference. For clarity, AC output characteristic equations define lines that pass through the origin. Actual device requirements when the output voltage is above V_{oh} or below V_{ol} are specified in the DC characteristics (Table 9-1). As in conventional PCI, the DC characteristics are the only conditions under which steady-state operation is intended. The higher current portions of the AC characteristics are intended to be reached only during switching transients.

Table 9-2: AC Specifications

Symbol	Parameter	Condition	Min	Max	Unit	Note
			PCI-X			
Output Buffer Drive Currents						
$I_{oh(AC)}$	Switching Current High	$0 < V_{cc}-V_{out} \le 3.6V$		$-74(V_{cc}-V_{out})$	mA	
		$0 < V_{cc}-V_{out} \le 1.2V$	$-32 (V_{cc}-V_{out})$		mA	1
		$1.2V < V_{cc}-V_{out} \le 1.9V$	$-11 (V_{cc}-V_{out}) -25.2$		mA	1
		$1.9V < V_{cc}-V_{out} \le 3.6V$	$-1.8 (V_{cc}-V_{out}) -42.7$		mA	1
$I_{ol(AC)}$	Switching Current Low	$0 \le V_{out} \le 3.6V$		$100V_{out}$	mA	
		$0 < V_{out} \le 1.3V$	$48 V_{out}$		mA	1
		$1.3V < V_{out} \le 3.6V$	$5.7 V_{out} + 55$		mA	1
Clamp Currents						
I_{cl}	Low Clamp Current	$-3V < V_{in} \le -0.8875V$	$-40 + (V_{in}+1) / 0.005$		mA	
		$-0.8875V < V_{in} \le -0.625V$	$-25 + (V_{in}+1) / 0.015$		mA	
I_{ch}	High Clamp Current	$0.8875V \le V_{in}-V_{cc} < 4V$	$40 + (V_{in}-V_{cc}-1) / 0.005$		mA	
		$0.625V \le V_{in}-V_{cc} < 0.8875V$	$25 + (V_{in}-V_{cc}-1) / 0.015$		mA	
			66 MHz Conventional PCI (ref)			
AC Drive Points						
$I_{oh(AC)}$	Switching Current High	$V_{out} = 0.7V_{cc}$		$-32V_{cc}$	mA	
		Vout = $0.3V_{cc}$	$-12V_{cc}$		mA	
$I_{ol(AC)}$	Switching Current Low	$V_{out} = 0.18V_{cc}$		$38V_{cc}$	mA	
		$V_{out} = 0.6V_{cc}$	$16V_{cc}$		mA	
Clamp Currents						
I_{ch}	High clamp current	$V_{cc}+4 > V_{in} \ge V_{cc}+1$	$25 + (V_{in}-V_{cc}-1) / 0.015$		mA	
I_{cl}	Low clamp current	$-3 < V_{in} \le -1$	$-25 + (V_{in}+1) / 0.015$		mA	

Note:

1. In conventional PCI switching, current characteristics for **REQ#** and **GNT#** are permitted to be one half of that specified here; i.e., half size drivers may be used on these signals. In PCI-X devices, **REQ#** and **GNT#** must have full-size drivers. This specification does not apply to **CLK** and **RST#** which are system outputs. "Switching Current High" specifications are not relevant to **SERR#, INTA#, INTB#, INTC#**, and **INTD#,** which are open drain outputs.

Table 9-3: Output Slew Rates

			PCI-X		Conventional PCI 66 (ref)			
Symbol	Parameter	Condition	Min	Max	Min	Max	Units	Note
t_r	Output rise slew rate	$0.3V_{CC}$ to $0.6V_{CC}$	1	6	1	4	V/ns	1
t_f	Output fall slew rate	$0.6V_{CC}$ to $0.3V_{CC}$	1	6	1	4	V/ns	1

Note:

1. This parameter is to be interpreted as the cumulative edge rate across the specified range rather than the instantaneous rate at any point within the transition range. The test load is specified in Figure 9-11 (66 MHz reference values use the test load in Figure 9-10). The specified load is optional. The designer may elect to meet this parameter with an unloaded output per revision 2.0 of the PCI specification. However, adherence to both maximum and minimum parameters is required (the maximum is not simply a guideline). Rise slew rate does not apply to open drain outputs.

Output drive current limits from Table 9-2 are illustrated in Figure 9-1 and Figure 9-2. Conventional PCI 33 MHz 3.3V limits are included for reference.

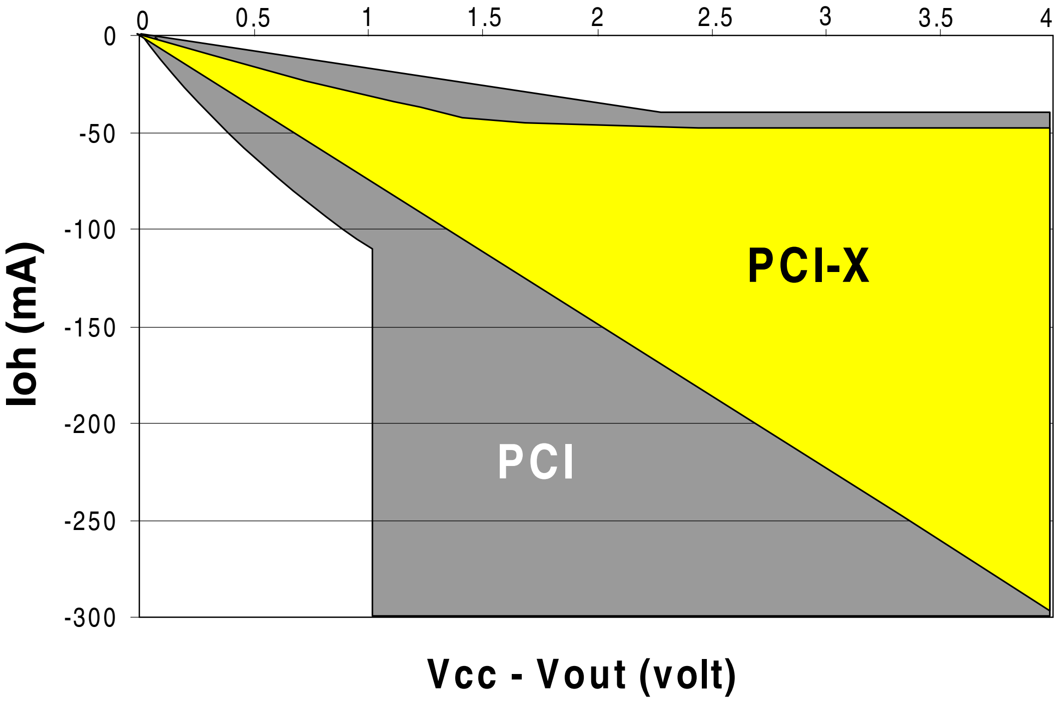


Figure 9-1: PCI-X Pull-Up Output Buffer I/V Curves

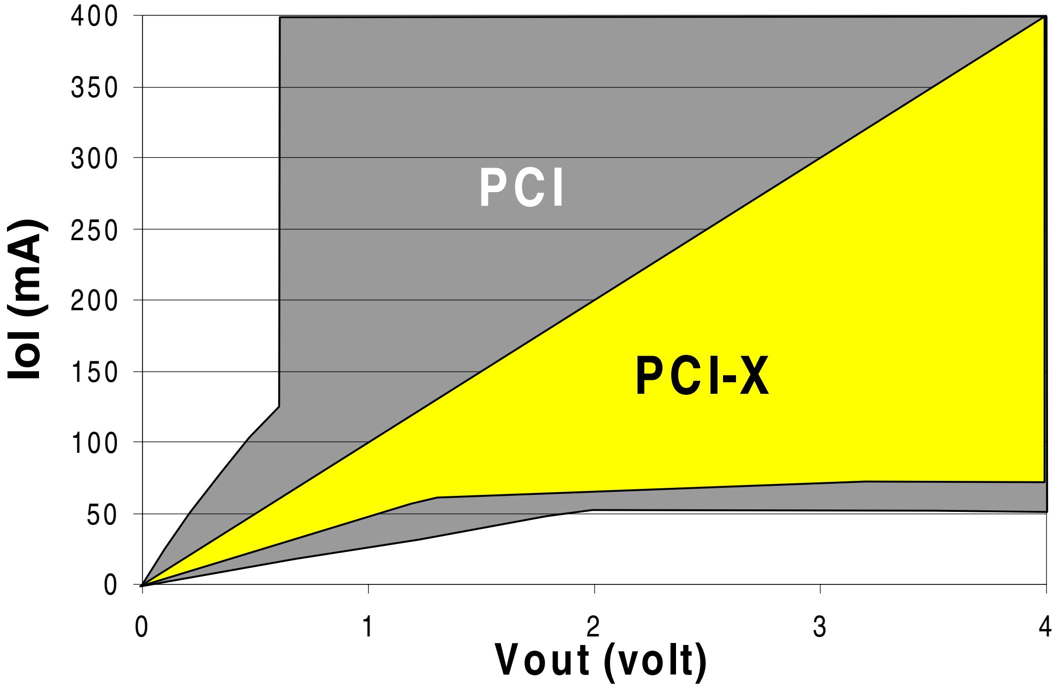


Figure 9-2: PCI-X Pull-Down Output Buffer I/V Curves

Figure 9-3 and Figure 9-4 are reference information showing the same output drive characteristics as Figure 9-1 and Figure 9-2 with the axes the same as PCI 2.2.

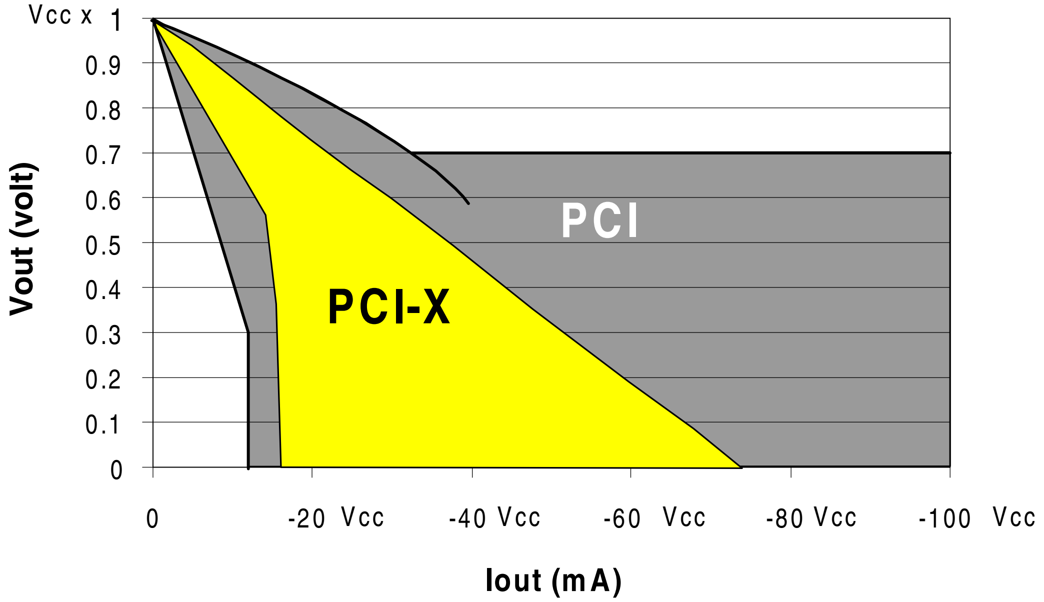


Figure 9-3: PCI-X Pull-Up Output Buffer V/I Curves (ref)

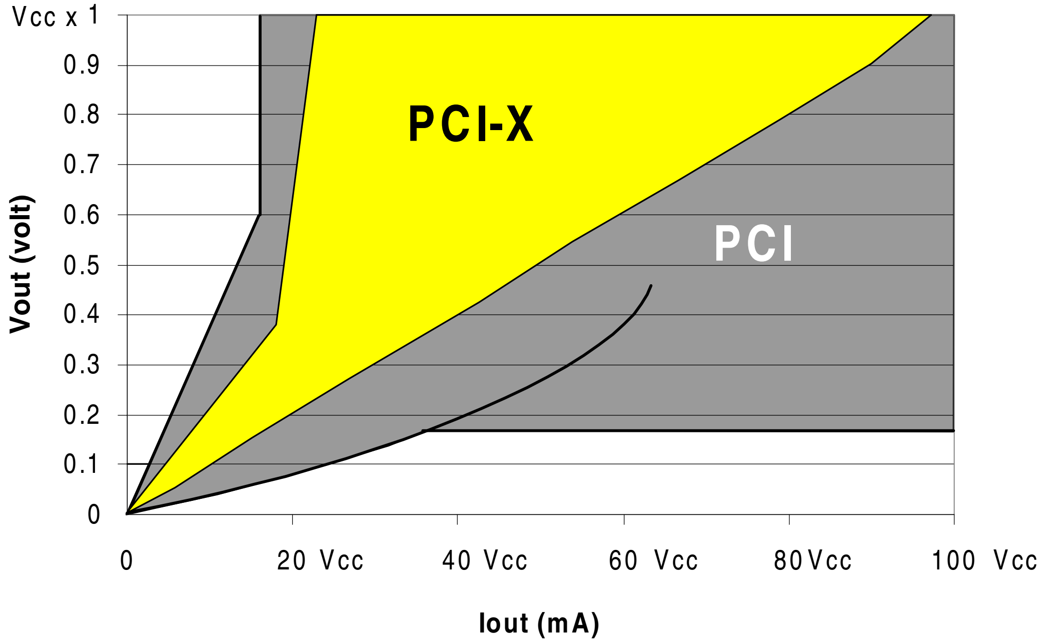


Figure 9-4: PCI-X Pull-Down Output Buffer V/I Curves (ref)

9.3. Maximum AC Ratings and Device Protection

Maximum AC rating and device protection requirements are the same for PCI-X devices as for conventional PCI devices in a 3.3 V signaling environment.

9.4. Timing Specification

9.4.1. Clock Specification

Clock measurement conditions are the same for PCI-X devices as for conventional PCI devices in a 3.3 V signaling environment except for voltage levels specified in Table 9-1.

In the case of add-in cards, compliance with the clock specification is measured at the add-in card component not at the connector. As with conventional PCI, devices used behind a PCI-to-PCI bridge on an add-in card use the clock output specification of the selected bridge rather than the specification shown here. Some PCI-to-PCI bridges have different clock output specifications.

The same spread-spectrum clocking techniques are allowed in PCI-X as for 66 MHz conventional PCI. If a device includes a PLL, that PLL must track the input variations of spread-spectrum clocking specified in Table 9-4.

T_{cyc} in Table 9-4 indicates the minimum and maximum **CLK** cycle times for the various specified frequencies. The minimum and maximum clock period specifications must not be violated for any single clock cycle. System designers must assure that the system clock period, including all sources of clock period variation (e.g., tolerance and jitter), is always within the minimum and maximum defined ranges. For example, if a specific system clock design has a maximum clock period variation of 180 ps, the maximum that the nominal frequency can be set for such a clock for 133 MHz operation, is

$$1/(7.5 \text{ ns} + 0.18 \text{ ns}) = 130.208 \text{ MHz}.$$

This setting of the nominal frequency accounts for the clock period variation without violating the minimum clock period of 7.5 ns.

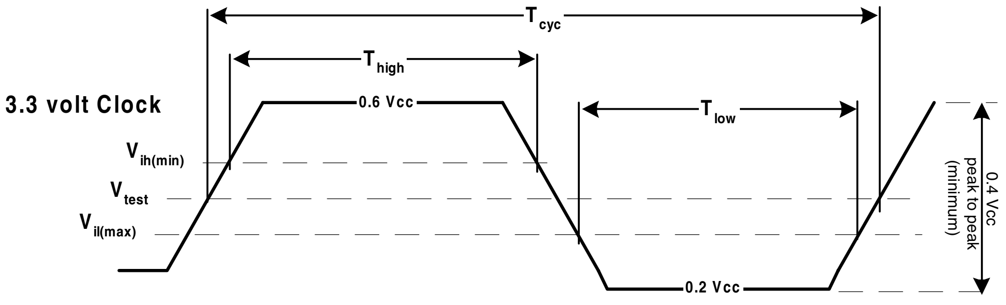


Figure 9-5: 3.3V Clock Waveform

Table 9-4: Clock Specifications

		PCI-X 133		PCI-X 66		Conv PCI 66 (ref)		Conv PCI 33 (ref)			
Sym	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Unit**	**Notes**
T_{cyc}	**CLK** Cycle Time	7.5	20	15	20	15	30	30	∞	ns	1,3,4
T_{high}	**CLK** High Time	3		6		6		11		ns	
T_{low}	**CLK** Low Time	3		6		6		11		ns	
-	**CLK** Slew Rate	1.5	4	1.5	4	1.5	4	1	4	V/ns	2,4
Spread Spectrum Requirements											
f_{mod}	modulation frequency	30	33	30	33	30	33			kHz	
f_{spread}	frequency spread	-1	0	-1	0	-1	0			%	

Notes:

1. For clock frequencies above 33 MHz, the clock frequency may not change beyond the spread-spectrum limits except while **RST#** is asserted.
2. This slew rate must be met across the minimum peak-to-peak portion of the clock waveform as shown in Figure 9-5.
3. The minimum clock period must not be violated for any single clock cycle, i.e., accounting for all system jitter.
4. All PCI-X 133 devices must also be capable of operating in PCI-X 66. All PCI-X devices must be capable of operating in conventional PCI 33 mode and optionally are capable of conventional PCI 66 mode.

9.4.2. Timing Parameters

Table 9-5 shows the timing specifications for all signals other than the clock.

Table 9-5: General Timing Parameters

Symbol	Parameter	PCI-X 133		PCI-X 66		Conventional PCI 66 (ref)		Conventional PCI 33 (ref)			
		Min	Max	Min	Max	Min	Max	Min	Max	Units	Notes
T_{val}	**CLK** to Signal Valid Delay - bused signals	0.7	3.8	0.7	3.8	2	6	2	11	ns	1, 2, 3, 10, 11
$T_{val(ptp)}$	**CLK** to Signal Valid Delay - point to point signals	0.7	3.8	0.7	3.8	2	6	2	12	ns	1, 2, 3, 10, 11
T_{on}	Float to Active Delay	0		0		2		2		ns	1, 7, 10, 11
T_{off}	Active to Float Delay		7		7		14		28	ns	1, 7, 11
T_{su}	Input Set up Time to **CLK** - bused signals	1.2		1.7		3		7		ns	3, 4, 8
$T_{su(ptp)}$	Input Set up Time to **CLK** - point to point signals	1.2		1.7		5		10,12		ns	3, 4
T_h	Input Hold Time from **CLK**	0.5		0.5		0		0		ns	4
T_{rst}	Reset Active Time	1		1		1		1		ms	5
$T_{rst-clk}$	Reset Active Time after **CLK** stable	100		100		100		100		µs	5
$T_{rst-off}$	Reset Active to output float delay		40		40		40		40	ns	5, 6
T_{rrsu}	**REQ64#** to **RST#** setup time	10		10		10		10		clocks	
T_{rrh}	**RST#** to **REQ64#** hold time	0	50	0	50	0	50	0	50	ns	9
T_{rhfa}	**RST#** high to first Configuration access	2^{26}		2^{26}		2^{25}		2^{25}		clocks	
T_{rhff}	**RST#** high to first **FRAME#** assertion	5		5		5		5		clocks	
T_{pvrh}	Power valid to **RST#** high	100		100		100		100		ms	
T_{prsu}	PCI-X initialization pattern to **RST#** setup time	10		10						clocks	
T_{prh}	**RST#** to PCI-X initialization pattern hold time	0	50	0	50					ns	9
T_{rlcx}	Delay from **RST#** low to **CLK** frequency change	0		0						ns	

Notes:

1. See the timing measurement conditions in Figure 9-6.
2. Minimum times are measured at the package pin (not the test point) with the load circuit shown in Figure 9-10. Maximum times are measured with the test point and load circuit shown in Figure 9-8 and Figure 9-9.
3. Setup time for point-to-point signals applies to **REQ#** and **GNT#** only. All other signals are bused.
4. See the timing measurement conditions in Figure 9-7.
5. **RST#** is asserted and deasserted asynchronously with respect to **CLK**.
6. All output drivers must be floated when **RST#** is active.
7. For purposes of Active/Float timing measurements, the Hi-Z or "off" state is defined to be when the total current delivered through the component pin is less than or equal to the leakage current specification.
8. Setup time applies only when the device is not driving the pin. Devices cannot drive and receive signals at the same time.
9. Maximum value is also limited by delay to the first transaction (T_{rhff}). The PCI-X initialization pattern control signals and **REQ64#** after the rising edge of **RST#** must be deasserted no later than two clocks before the first **FRAME#** and must be floated no later than one clock before **FRAME#** is asserted.
10. A PCI-X device is permitted to have the minimum values shown for T_{val}, $T_{val(ptp)}$, and T_{on} only in PCI-X mode. In conventional mode, the device must meet the requirements specified in PCI 2.2 for the appropriate clock frequency.
11. Device must meet this specification independent of how many outputs switch simultaneously.

9.4.3. Measurement and Test Conditions

Timing measurement and test conditions are the same as for conventional PCI, except for the output slew rate test load shown in Figure 9-11. Figure 9-6 shows the output waveform measurement conditions. Figure 9-7 shows the input waveform measurement conditions.

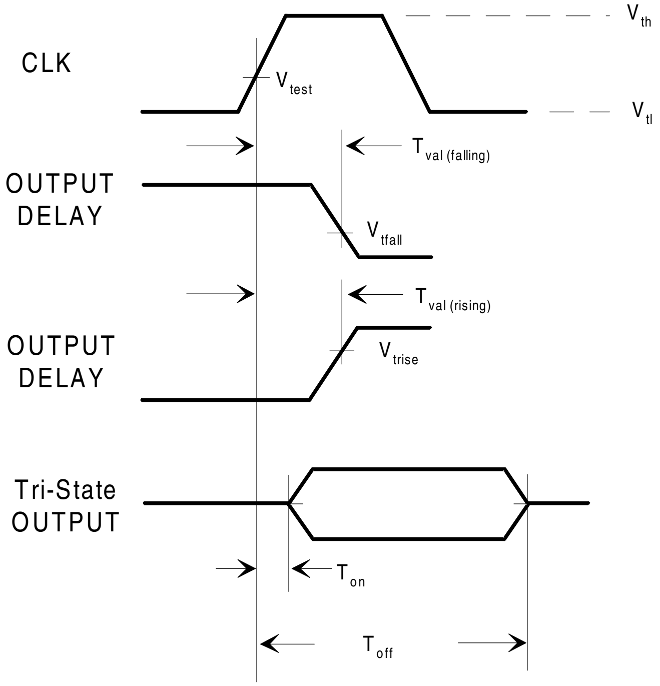


Figure 9-6: Output Timing Measurement Conditions

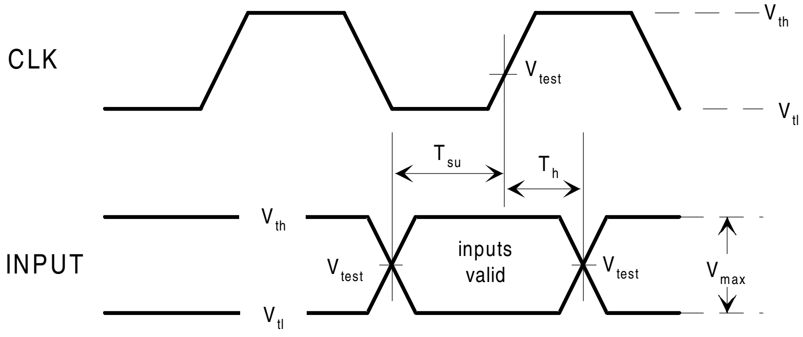


Figure 9-7: Input Timing Measurement Conditions

Table 9-6: Measurement Condition Parameters

Symbol	PCI-X	3.3V Signaling Conventional PCI 66(ref)	Units	Notes
V_{th}	$0.6V_{CC}$	$0.6V_{CC}$	V	1
V_{tl}	$0.25V_{CC}$	$0.2V_{CC}$	V	1
V_{test}	$0.4V_{CC}$	$0.4V_{CC}$	V	
V_{trise}	$0.285V_{CC}$	$0.285V_{CC}$	V	2
V_{tfall}	$0.615V_{CC}$	$0.615V_{CC}$	V	2
V_{max}	$0.4V_{CC}$	$0.4V_{CC}$	V	1
Input Signal Slew Rate	1.5	1.5	V/ns	3

Notes:

1. The test for the 3.3V environment is done with 0.1^*V_{CC} of overdrive. V_{max} specifies the maximum peak-to-peak waveform allowed for measuring input timing. Production testing is permitted to use different voltage values but must correlate results back to these parameters.
2. V_{trise} and V_{tfall} are reference voltages for timing measurements only.
3. Input signal slew rate in PCI-X mode is measured between V_{il} and V_{ih}.

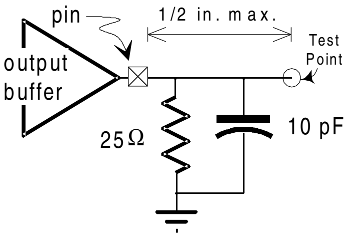


Figure 9-8: T_{val}(max) Rising Edge Test Load

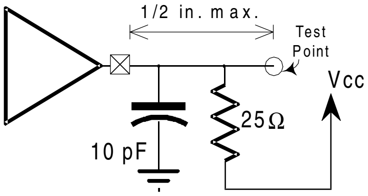


Figure 9-9: T_{val}(max) Falling Edge Test Load

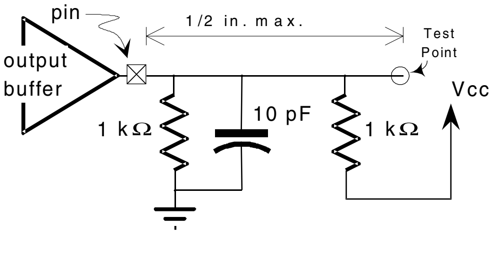


Figure 9-10: T_{val}(min) Test Load

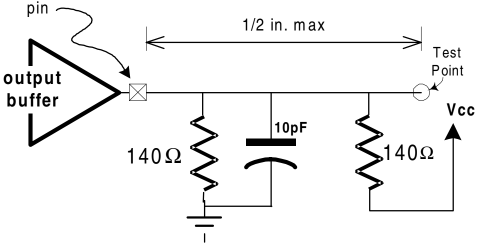
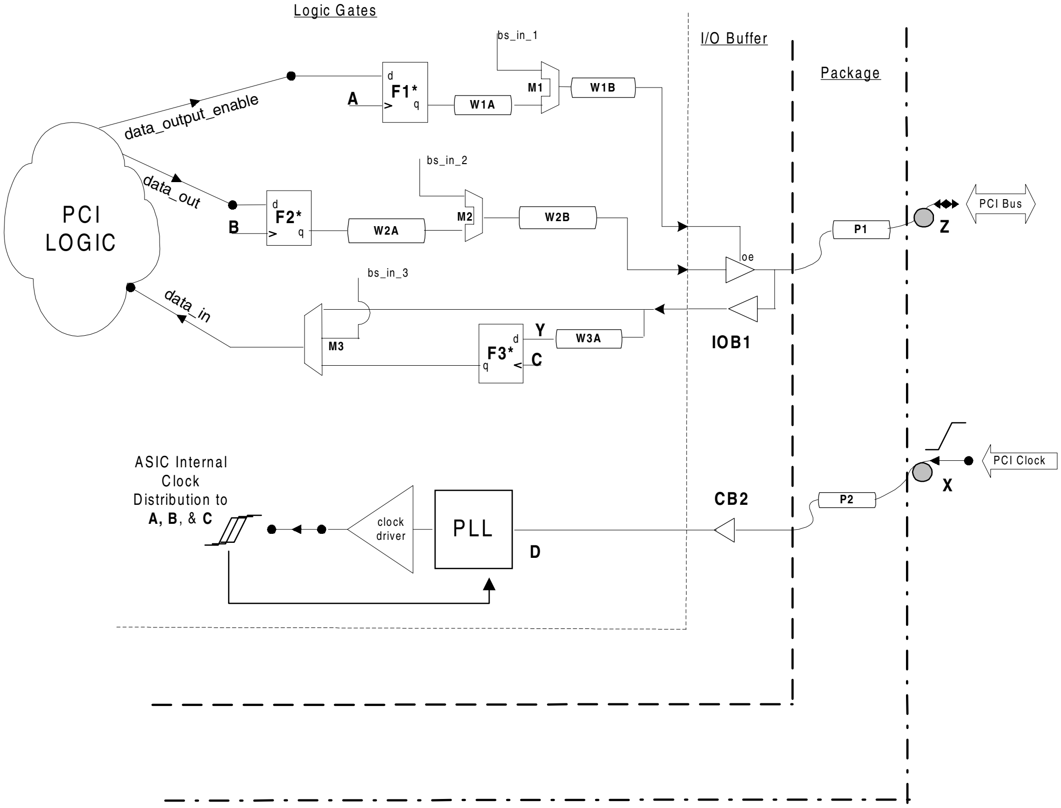


Figure 9-11: Output Slew Rate Test Load

9.4.4. Device Internal Timing Examples

Figure 9-12 shows a typical PCI-X device implementation including wire-delay elements. Table 9-7 and Table 9-8 show how the delay elements are combined to calculate T_{val}, T_{su}, and T_h for this example device.



* -- All Flip Flops are assumed to be rising edge triggered

Figure 9-12: Device Internal Timing Example

For this example, the maximum value of T_{val} is determined by the slower of two paths, the output enable path through flip-flop F1 and the data path through flip-flop F2. Table 9-7 shows both calculations.

Table 9-7: T_{val} Delay Paths

X to A to Z		X to B to Z	
Parameter	**Description**	**Parameter**	**Description**
P2	Clock Input Package Delay	P2	Clock Input Package Delay
CB2	Clock Input Buffer Delay	CB2	Clock Input Buffer Delay
PLL	PLL Jitter/Phase Error/Clock Wire Delay	PLL	PLL Jitter/Phase Error/Clock Wire Delay
F1	Flop-1 Clock to Q Delay	F2	Flop-2 Clock to Q Delay
W1A	Wire Delay	W2A	Wire Delay
M1	Mux-1 Delay	M2	Mux-2 Delay
W1B	Wire Delay	W2B	Wire Delay
IOB1(oe_)	I/O Buffer Turn On Delay	IOB1(output)	I/O Buffer Output Delay
P1	Output Signal Package Delay	P1	Output Signal Package Delay
	Sum is X→ A → Z delay		*Sum is X→ B → Z delay*

T_{su} and T_h are calculated with the following equations:

$$T_{su} = F3_{su} + (Z \rightarrow Y)_{max} - (X \rightarrow C)_{min}$$

$$T_h = F3_{hold} - (Z \rightarrow Y)_{min} + (X \rightarrow C)_{max}$$

where:

$F3_{su}$ is the setup time for flip-flop F3.

$F3_{hold}$ is the hold time for flip-flop F3.

(Z → Y) is the delay from point Z to point Y in Figure 9-12 as shown in Table 9-8, and

(X→C) is the delay from point X to point C in Figure 9-12 as shown in Table 9-8.

Table 9-8: T_{su} and T_h Delay Paths

Z to Y		X to C	
Parameter	**Description**	**Parameter**	**Description**
P1	Package Delay	P2	Package Delay
IOB1(Input)	I/O Buffer Input Delay	CB2	Clock Input Buffer Delay
W3A	Wire Delay	PLL	PLL Jitter/Phase error/Clock Wire Delay
	Sum is Z → Y delay		*Sum is X→ C delay*

9.5. Clock Uncertainty

The maximum allowable clock uncertainty including jitter is shown in Table 9-9 and Figure 9-13. This specification applies not only at a single threshold point but at all points on the clock edge between V_{il} and V_{ih}. For add-in cards, the maximum skew is measured between component pins not between connectors.

Table 9-9: Clock Uncertainty Parameters

Symbol	PCI-X	Conventional PCI 66 (ref)	3.3V Signaling Conventional PCI 33 (ref)	Units
$V_{test\text{-}clk}$	$0.4V_{CC}$	$0.4V_{CC}$	$0.4V_{CC}$	V
T_{skew}	0.5 (max)	1 (max)	2 (max)	ns

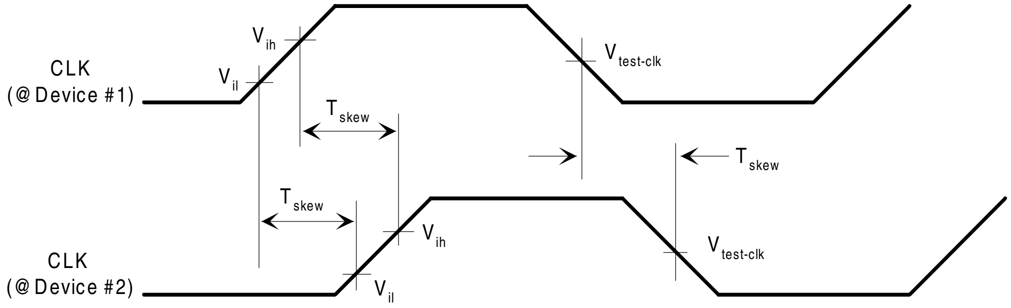


Figure 9-13: Clock Uncertainty Diagram

9.6. Reset

PCI-X introduces one new timing case related to **RST#** not present in conventional PCI. Figure 9-14 shows the timing requirements for the PCI -X initialization pattern when switching into PCI-X mode. Parameter values are shown in Table 9-5.

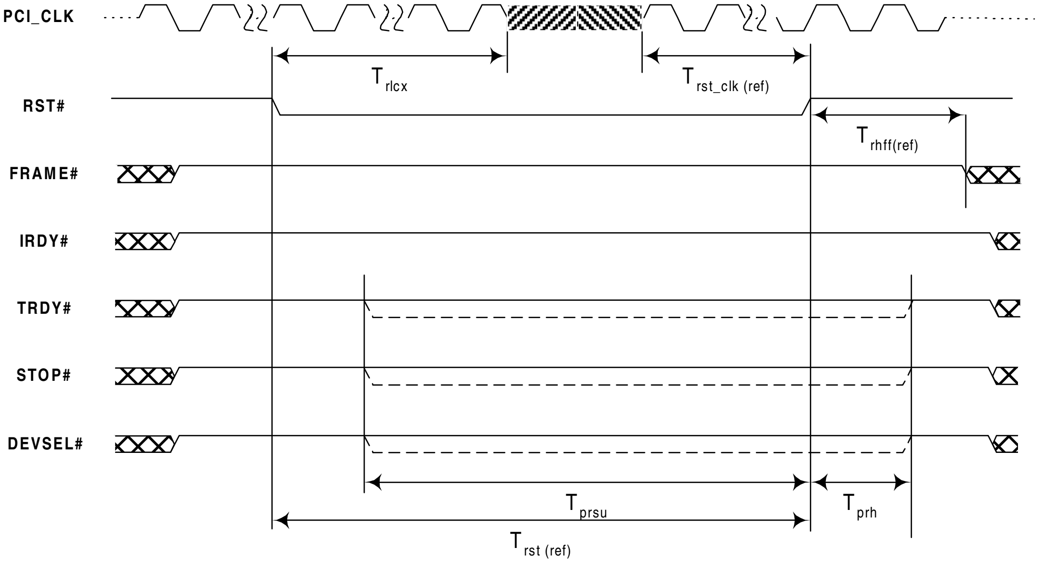


Figure 9-14: RST# Timing for Switching to PCI-X Mode Pull-ups

Implementation Note: PLL Lock Times

As in conventional PCI, PLLs in some PCI-X devices do not lock to the input clock until after the rising edge of **RST#**. For example, PCI 2.2 requires the input clock to be stable only 100 µs before the rising edge or **RST#** (T_{rst_clk} in Table 9-5). If the clock is stable no more than the minimum time before the rising edge or **RST#** (e.g., after a hot-insertion), and the device implements an internal PLL that requires longer than this time to lock, the clock internal to the device will not be stable at the rising edge of **RST#**. Such devices are advised by PCI HP 1.0 not to leave the reset state until the PLL is locked and the bus is idle (**FRAME#** and **IRDY#** deasserted) to avoid the possibility of leaving the reset state in the middle of a transaction between two other devices.

In PCI-X systems, the PCI-X initialization pattern is not guaranteed to be stable until ten clocks before the rising edge of **RST#** (T_{prsu} in Table 9-5). If the PLL requires this information to lock, the lock time in PCI-X systems is delayed even farther past the rising edge of **RST#**.

A PLL that controls the clock to the bus interface of a PCI-X device must be stable early enough for the device to respond to its first Configuration Read transaction, which is specified to be no earlier than T_{rhfa} (see Table 9-5) after the rising edge of **RST#**. No method is specified for PCI-X devices to know when their PLL is locked. Some PLL designs detect when they are locked and indicate this to the affected logic. Other PLLs are designed to lock after a fixed length of time or a limited number of input clocks. Devices that use the second kind of PLLs must count input clocks or implement other suitable means to indicate that the PLL is locked.

9.7. Pull-ups

A bus capable of PCI-X operation has the same requirement for pull-ups as described in PCI 2.2, with the following exceptions:

1. **PCIXCAP** pin connection described in Section 9.10.
2. Minimum pull-up resistor value is 5 kΩ.

As in conventional PCI, the PCI -X central resource is permitted to implement active bus keepers rather than passive pull-up resistors. A complete specification of such bus keepers is beyond the scope of this document; however, the following guidelines generally apply:

1. Keepers must supply sufficient drive current to overcome the input leakage current of a fully loaded bus and maintain a valid logic level.
2. Keepers optionally may be designed either to hold the bus in its present low or high state, or may be designed only to hold the bus in the high state and work in concert with central-resource logic that precharges an idle bus to the high logic level.
3. Keepers integrated with the source bridge must not adversely affect source bridge timing beyond the limits of the PCI-X definition or PCI 2.2 (as appropriate for the mode of operation).

9.8. System Noise Budget

This section allocates the minimum input noise immunity of a device to the different sources of noise.

The total noise budget allocated for PCI-X is very similar to convention PCI, with the low noise budget identified as the difference between V_{ol} and V_{il}, and the high noise budget identified as the difference between the V_{oh} and V_{ih}. Figure 9-15 graphically illustrates the total noise budget.

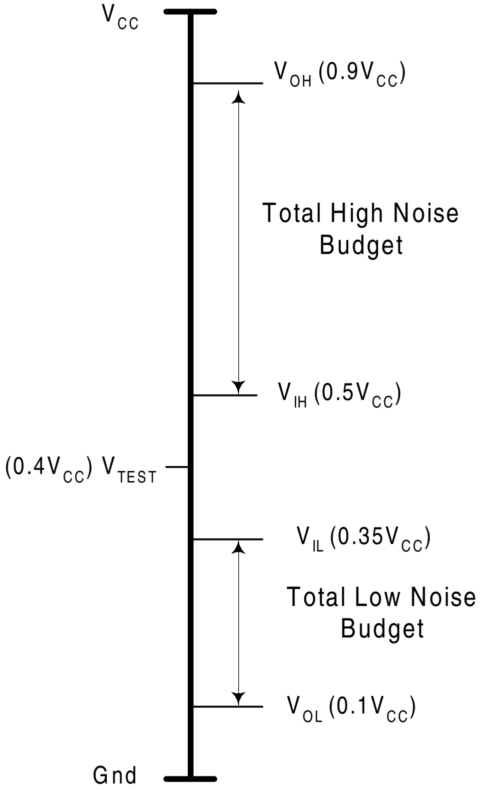


Figure 9-15: PCI-X Noise Budget

Table 9-10 allocates this noise margin to the various sources of noise in the system and assigns responsibility for meeting that budget.

Table 9-10: PCI-X System Noise Budget

Noise Source	Responsibility	High Noise Budget	Low Noise Budget
Reflective Noise	Platform	0.30V_{CC}	0.15V_{CC}
Crosstalk	Platform	0.05V_{CC}	0.05V_{CC}
Input Reference Offset	Device	0.05V_{CC}	0.05V_{CC}
Total		0.4V_{CC}	0.25V_{CC}

Reflective noise includes those aspects of signal quality caused by impedance mismatches and resultant signal reflections. The platform vendor adjusts system-board loading and routing to meet this requirement. The add-in card must be designed according to the requirements of Section 9.13.

Crosstalk includes noise caused by switching of adjacent signals that is induced onto the signal in question through capacitive and inductive coupling between traces on the circuit board. The platform vendor must control signal spacing and length to meet this budget. The add-in card must be designed according to the requirements of Section 9.13.3.

Input Reference Offset is the offset in the reference power supply levels used by the input buffer inside a device. It is caused by the inductance in the supply paths of the device and changes in supply current. This budget applies during the input setup time. (The effects of changing supply currents on output buffers is included in T_{val} in Table 9-5.)

9.9. System Timing Budgets

PCI-X system timing is measured using the same techniques as specified in PCI 2.2 for 66 MHz conventional PCI. Platform designers are permitted to implement any system topology. Platform designers must guarantee that all devices and add-in cards designed to this specification operate properly in any location in that topology. Platform designers are permitted to reduce the operating clock frequency to the limit specified in Table 9-4 to allow more time for signals to propagate and settle at all inputs with the specified setup time.

Table 9-11 shows the system-timing budget for the standard PCI-X operating frequencies.

Table 9-11: Setup Time Budget

Parameter	PCI-X 133 MHz	PCI-X 100 MHz	PCI-X 66 MHz	Conventional PCI 66 MHz (ref)	Conventional PCI 33 MHz (ref)	Units
T_{val} (max)	3.8	3.8	3.8	6	11	ns
T_{prop} (max)	2.0	4.5	9.0	5	10	ns
T_{skew} (max)	0.5	0.5	0.5	1	2	ns
T_{su} (min)	1.2	1.2	1.7	3	7	ns
T_{cyc}	7.5	10.0	15.0	15	30	ns

Table 9-12 shows the system-timing budget for minimum output delay and signal propagation time. Minimum signal propagation time (T_{prop}) is measured in a manner similar to the maximum delay described in PCI 2.2 for 66 MHz conventional PCI. Minimum T_{prop} begins when the voltage at the output buffer would have crossed the threshold point (V_{trise} or V_{tfall}) had the output been driving the T_{val} (min) test load shown in Figure 9-10. It ends the first time the signal crosses the input voltage limit for the initial logic level at any input pin. That is, for a falling signal T_{prop} (min) ends the first time the signal crosses V_{ih} (min) at any input pin. For a rising signal T_{prop} (min) ends the first time the signal crosses V_{il} (max) at any input pin.

Table 9-12: Hold Time Budget

Parameter	PCI-X	Conventional PCI 66 MHz (ref)	Conventional PCI 33 MHz (ref)	Units
T_{val} (min)	0.7	2	2	ns
T_{prop} (min)	0.3	0	0	ns
T_{skew} (max)	0.5	1	2	ns
T_h (min)	0.5	0	0	ns

9.10. PCIXCAP Connection

Add-in cards indicate that they are capable of PCI-X operation and, if so, at what frequency, by the connection of one pin on the PCI expansion connector, **PCIXCAP**, which is pin 38B. The connection of this pin must be consistent with the PCI-X Capability List item (see Sections 7.2 and 8.6.2) and the 133 MHz Capable bit in the PCI-X Status register or PCI-X Bridge Status register (see Sections 7.2.4 and 8.6.2.4). Conventional cards connect this pin directly to ground. PCI-X 133 cards connect **PCIXCAP** to ground through a 0.01 µF ±10% capacitor to provide an AC signal return path. PCI-X 66 cards connect **PCIXCAP** to ground through a 10 kΩ ±5% resistor in parallel with a 0.01 µF ±10% capacitor to provide an AC signal return path.

The maximum trace length between the resistor (if installed), capacitor, and the connector contact is 0.25 inches. The maximum trace length between the resistor (if installed), capacitor, and ground is 0.1 inches.

A PCI-X card is not permitted to connect **PCIXCAP** to anything else including supply voltages and device input and output pins. (In some PCI hot-plug systems, the system board detects the state of this pin before applying power to the slot.)

A PCI-X system provides a circuit for sensing the state of the **PCIXCAP** pin (see Section 14). Suitable decoupling to provide an AC signal return path is also required on the system board. A system that is capable of operation in PCI-X mode only at 66 MHz or below distinguishes PCI-X 133 cards from PCI-X 66 cards only for reporting card capabilities to the user, which can also be done by reading the 133 MHz Capable bit in the PCI-X Status register. If the system does not support PCI hot-plug, the **PCIXCAP** pin for multiple connectors is permitted to be bused and connected to a single system-board sensing circuit. PCI hot-plug systems must provide a means for the software to determine the states of **PCIXCAP** for each add-in slot independently.

9.11. IDSEL Connection to AD Bus

PCI-X systems drive the address four clocks before asserting **FRAME#** for configuration transactions (see Section 2.7.2.1). This allows additional settling time for the **IDSEL** input of devices. As in conventional PCI, PCI-X systems are permitted to connect the **IDSEL** input of devices to individual bits of the **AD** bus through series resistors. (Connection through a resistor reduces the effect on system timing of the extra load on the **AD** bit.) Such PCI-X systems must use a 2 kΩ ± 5% resistor, or a smaller value if system analysis guarantees that timing and noise budgets for the **AD** bus are met.

No device is permitted to connect **IDSEL** to **AD[16]** (device number 0), since this device number is reserved for the source bridge. For systems that have add-in board connectors and connect **IDSEL** to the **AD** bus, the first four add-in board connectors are recommended to be connected according to Table 9-13 to minimize the length of the **IDSEL** trace.

Table 9-13: IDSEL to AD Bit Assignment

Slot #	AD bit	Device Number
1	17	1
2	18	2
3	19	3
4	20	4

9.12. Power

9.12.1. Power Requirements

Device and add-in card power supply voltages and tolerances and add-in card load limits are the same as for conventional PCI devices and add-in cards.

9.12.2. Sequencing

As for conventional PCI, PCI-X devices have no power supply sequence requirements. The supply voltages are permitted to rise and fall in any order.

9.12.3. Decoupling

Same as conventional PCI.

9.13. Add-in Card Routing Requirements

As in conventional PCI, unless otherwise specified, signals listed as “Interrupt Pins,” and “JTAG Pins” in PCI 2.2 are exempt from the PCI-X routing requirements.

Platform routing characteristics are not specified. The platform designer is responsible for guaranteeing that add-in cards and component devices that conform to this specification function properly in any location.

9.13.1. Signal Loading

As in conventional PCI, PCI-X add-in cards are permitted to have no more than a single electrical load on each signal, including **CLK** and **RST#**.

9.13.2. Trace Length

PCI-X add-in card signal lengths are shown in Table 9-14. **REQ64#** and **ACK64#** are in the 32-bit portion of the connector and, therefore, are subject to the 32-bit signal length requirement. Conventional add-in card lengths are shown for reference. The trace length of **RST#** is also shown in the table. (**RST#** switches asynchronously with respect to **CLK**. The length of **RST#** is limited to guarantee **REQ64#** and the PCI-X initialization pattern are stable for the appropriate time after the rising edge of **RST#**.)

Table 9-14: Add-in Card Trace Length

Parameter	PCI-X		Conventional PCI (ref)		Units
	Min	Max	Min	Max	
CLK length	2.4	2.6	2.4	2.6	inch
32-bit interface signal length	0.75	1.5	-	1.5	inch
64-bit interface extension signal length	1.75	2.75	-	2.0	inch
RST# length	0.75	3.0	-	-	inch

9.13.3. Crosstalk

Trace spacing, geometries, and materials on add-in cards must limit cumulative crosstalk to 5% of the amplitude of the aggressor signal. Cumulative crosstalk is the total crosstalk from all adjacent lines to one victim line if the same aggressor signal is applied to all adjacent lines. See Section 15 for spacing and stack-up examples.

9.13.4. Transmission Line Characteristics

Characteristic impedance and propagation delay of signals on PCI-X add-in cards is shown in Table 9-15.

Table 9-15: Add-in Card Transmission Line Specifications

Parameter	PCI-X	Conventional PCI (ref)	Units
Board characteristic impedance (unloaded)	57 ±10%	60[1]-100	Ω
Signaling propagation delay	150-190	150-190	ps/inch

Note:

1. Minimum conventional PCI characteristic impedance shown is for a maximum C_{in} of 10 pF. The PCI-X definition allows a lower characteristic impedance but also requires a lower input capacitance.

10. Appendix—Conventional PCI vs. AGP vs. PCI-X Protocol Rule Comparison

Table 10-1: Conventional PCI vs. AGP vs. PCI-X Protocol Comparison

Arbitration	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Arbiter Monitoring Bus	No	No	No	YES
Bus Clock Speed	**Conventional PCI 32-bit bus, 64-bit bus**	**AGP 1.0**	**AGP 2.0**	**PCI-X 32bit bus, 64bit bus (MB/sec)**
33 MHz	133, 266 MB/sec	SDR -133, DDR-266 MB/sec	SDR -133, DDR-266 MB/sec	Not Supported
66 MHz	266, 533 MB/sec	SDR -266, DDR-533 MB/sec	SDR -266, DDR-533 MB/sec	266, 533 MB/sec
100 MHz	Not Supported	Not Supported	Not Supported	400, 800 MB/sec
133 MHz	Not Supported	66 MHz DDR – 533 MB/sec	66 MHz DDR – 533 MB/sec	533, 1066 MB/sec
266 MHz	Not Supported	Not Supported	QDR @ 66MHz – 1066 MB/sec	Not Supported
Transaction Types	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Memory	Supported	Supported	Supported	Supported
I/O	Supported	Not supported	Not supported	Supported
Config	Supported	Not supported	Not supported	Supported
Interrupt Acknowledge	Supported	Not supported	Not supported	supported
Special Cycle	Supported	Not supported	Not supported	Supported
Dual Address Cycle	Supported	Not supported	Not supported	Supported
Split Transactions	Not supported	Supported	Supported	Supported
Priority Transactions	Not supported	Supported	Supported	Not supported
Non-Cache-Coherent Transactions	Not supported	Supported	Supported	Supported
No/Relax Ordering Rules	Not supported	Supported	Supported	Supported
Address Re-mapping	Not supported	Supported	Supported	Not supported
Address and Data Bus	Multiplexed	De-Multiplexed	De-Multiplexed	Multiplexed
# of New Pins	N/A	16 new pins	20 new pins	1 new pin
Isochronous Transactions	Not supported	Not supported	Not supported	Not supported
Target Write Buffer Flush	No, supported using a standard read command	Yes, supported using the Flush command, with random data returned	Yes, supported using the Flush command, with random data returned	No, supported using a standard read command
Transaction Ordering	No (Bus Ordering only)	Yes, device control transaction ordering (Fence)	Yes, device control transaction ordering (Fence)	Yes, device control transaction ordering (Relaxed Ordering)
Orthogonal Protocol Support	Yes	No, Host not supported to source AGP commands	Yes, Host and AGP Master supported in enhanced protocol	Yes

Power Management Support (PME)	Optional	No	No	Yes
Non-Snooped Accesses to Host Memory Allowed While Host Memory Locked	Typically not supported	Yes	Yes	Yes
Transaction Termination	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Initiator Termination	Supported	Supported	Supported	Supported
Master-Abort	Supported	Not supported as PCI	Not supported as PCI	Supported
Target Disconnect with data	Supported	Supported	Supported	Supported
Target Disconnect without data	Supported	Not supported as PCI	Not supported as PCI	Not supported
Target Retry	Supported	Supported	Supported	Supported
Target-Abort	Supported	Not supported as PCI	Not supported as PCI	Supported
Burst Transaction	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
# burst data clocks	Two or more data clocks	Two or more data clocks	Two or more data clocks	One or more data clocks
Wait states	Target and/or Initiator can inject wait states	Target and/or Initiator can inject wait states only on QWORD boundaries	Target and/or Initiator can inject wait states only on QWORD boundaries	Initiator cannot inject wait states. Target can only inject initial wait states before data transfer starts
Cacheline Size	Programmable	Not used (replaced with QWORD boundaries)	Not used (replaced with 64 byte block size)	Not used (replaced with ADB)
Latency Timer	Programmable	Not supported	Not supported	Programmable
Memory Read	BE are valid	BE are reserved	BE are reserved	BE only for DWORD transactions
Memory Read Line	BE are valid (ignored by Target)	Not supported	Not supported	Replaced with burst transactions
Memory Read Multiple	Yes	Not supported	Not supported	Not supported
Memory Write	BE are valid	BE are valid	BE are valid	BE are valid
Memory Write and Invalidate	BE are valid (ignored by Target)	Not supported	Not supported	Replaced with Memory Write Block transaction
Burst Length	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Minimum data clocks	2 Data Clocks	2 Data Clocks	2 Data Clocks	Byte count or 32 clocks for 32 bit bus Byte count or 16 clocks for 64 bit bus
32 bit Bus	8 bytes	8 bytes	8 bytes	8 bytes
32 bit Minimum Burst Length	8 bytes	8 bytes	8 bytes	Byte count or 128 bytes
64 bit Bus	16 bytes	Not supported	Not supported	16 bytes
64 bit Minimum Burst Length	16 bytes	Not supported	Not supported	Byte count or 128 bytes
Maximum block size	Open (Unlimited)	32 bytes, 256 bytes for long reads	64 bytes, 256 bytes for long reads	4096 bytes
Standard block size	Cache Line Size	Fixed 32 bytes	Fixed 64 bytes	Fixed 128 bytes

Topology	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Port/Bus/Hierarchical Bus	No/Yes/Yes	Yes/No/No	Yes/No/No	Yes/Yes/Yes
PCI Slot Compatibility	N/A	No	No	Yes
Decode Speeds	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
1 clock after address phase(s)	Decode Speed FAST	Not supported	Not supported	Not supported
2 clock after address phase(s)	Decode Speed MED	Supported	Supported	Decode Speed A
3 clock after address phase(s)	Decode Speed SLOW	Supported	Supported	Decode Speed B
4 clock after address phase(s)	Decode Speed Subtractive	Supported	Supported	Decode Speed C
6 clock after address phase(s)	Not supported	Not supported	Not supported	Decode Speed Subtractive
Config Access	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Address Predrive	Optional, system dependent	Not supported	Not supported	Required
Transaction Phases	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Address and Command Phase	Supported	Supported	Supported	Supported
Response Phase	Decode Speed	Not supported	Not supported	Decode Speed
Attribute Phase	Not Available	Supported	Supported	Supported
Data Phase	Supported	Supported	Supported	Supported
Termination Phase	Initiator and Target	Initiator and Target	Initiator and Target	Initiator and Target
Turn-around	Required	Required	Required	Required
Attribute Field	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Byte Count	Not supported	Supported	Supported	Supported
Don't Snoop	Not supported	Supported	Supported	Supported
Relaxed Ordering	Not supported	Supported	Supported	Supported
Function #	Not supported	N/A	N/A	Supported
Device #	Not supported	N/A	N/A	Supported
Bus #	Not supported	N/A	N/A	Supported
Tag	Not supported	Supported	Supported	Supported
Bus Width	**Conventional PCI**	**AGP 1.0**	**AGP 2.0**	**PCI-X**
Memory Address	32 or 64 bits	32/36/64 bits	2/47/64 bits	64 bits
I/O Address	32 bits	N/A	N/A	32 bits
Data	32 or 64 bits	32 bits	32 bits	32 or 64 bits

11. Appendix—Use Of Relaxed Ordering

Ordering rules are specified to guarantee a consistent view of data by all devices in the system and rational behavior for communication between multiple devices and their software drivers (if any). There is a trade-off, however, between the strictness of the ordering rules and the performance and scalability of a system. In a very simple system, it is practical to have all devices in the system observe all transactions in exactly the same order. Consider, for example, a system built around a single shared bus utilizing only atomic transactions. In this system, all transactions initiated by any initiator to any target are visible in the same order by all devices on the bus.

Extending strict ordering behavior to more complex multiple-bus systems is possible but can extract a severe performance penalty. Generally, ordering requirements are relaxed in varying degrees to meet performance objectives without imposing an undue burden on software. In a two-bus system, for example, transactions between peer devices on one bus are generally allowed to proceed without regard to transactions between peer devices on the other bus. In this case, one set of (implicit) ordering rules would be applied to transactions that stay entirely on one bus, and a different set applied to transactions that cross between the buses. Allowing memory write transactions to be posted is an example of such an ordering relaxation designed to improve system performance.

The larger and more complex the system, the more difficult the trade-offs become. Systems may implement several classes of interconnects such as processor and memory buses, interconnection fabrics (e.g., crossbar, hypercube, etc.), and I/O buses. Ordering requirements for transactions between CPUs and memory or between the CPUs themselves typically vary with processor architecture and system implementation. These requirements sometimes differ substantially from the ordering requirements imposed by a standardized I/O subsystem such as PCI. For example, one of the underlying assumptions in PCI ordering is the tree structure of the bus hierarchy. Such a structure is not present in some processor-memory domains. Devices that connect the domains (e.g., a PCI host bridge) are responsible for managing differences in ordering requirements between the domains without unduly degrading performance.

Conventional PCI ordering rules apply globally to all transactions without regard to the underlying communication semantics. The Relaxed Ordering attribute in PCI-X transactions allows certain ordering requirements to be indicated explicitly on a transaction-by-transaction basis providing a tool to help system designers and software writers achieve better overall performance. Specifically, the PCI-X Relaxed Ordering attribute may be used to allow a memory write transaction to pass other memory writes and to allow a Split Read Completion to pass memory writes. An initiator permits the first case by setting the Relaxed Ordering attribute on a memory write transaction and permits the second case by setting the Relaxed Ordering attribute on a Split Read Request. (The Relaxed Ordering attribute is echoed in the corresponding completion.) The Relaxed Ordering attribute has no effect on a Split Write Request.

In general, read and write transactions to or from I/O devices are classified as payload or control. (PCI 2.2 Appendix E refers to payload as Data and control as Flag and Status.) If the payload traffic requires multiple data phases or multiple transactions, such payload traffic rarely requires ordered transactions. That is, the order in which the bytes of the payload arrive is inconsequential, if they all arrive before the corresponding control traffic. However, control traffic generally does require ordered transactions. I/O devices that follow this programming model could use this distinction to set the Relaxed Ordering attribute in hardware with no device driver intervention. Such a device could set the Relaxed Ordering attribute bit for all payload read and write transactions and not set the attribute for all control read and write transactions. Other devices may want to provide a means (beyond the scope of the PCI-X specification) for their device driver to indicate when it is permissible to set the Relaxed Ordering attribute. In all cases, no requester is allowed to set the Relaxed Ordering attribute bit if the Enable Relaxed Ordering bit in the PCI-X Command register is cleared.

11.1. Relaxed Write Ordering

When an I/O device receives a block of data destined for system memory, it typically writes that data (payload) into a location in system memory previously specified by the I/O device's software driver. After transferring all of the data, the I/O device indicates completion of the I/O operation by writing status (control) information, often to a separate area in memory, and then possibly generating an interrupt. This type of programming model generally considers the memory buffer area undefined until the status has been written. Thus, individual write transactions to that buffer area can be allowed to complete out of order as long as the status write pushes all previous writes ahead of it. An I/O device can easily accomplish this by setting the Relaxed Ordering attribute for all payload write transactions but always generating a separate transaction for the status write(s) with the Relaxed Ordering attribute not set.

Relaxed write ordering is arguably of little value within the PCI-X domain. Generally, all writes from a single device pass through the same host bridge on their way to system memory, so if one write gets blocked, so would the next one. The real value comes within the host bridge where PCI ordering must be mapped into the host system. It can be very difficult for a system with multiple paths to memory from a single host bridge to ensure all CPUs see all writes from that host bridge in order without significant performance impact. Relaxed write ordering can allow kilobytes to gigabytes of payload data to stream into memory while imposing the ordering burden on only the handful of status writes that really need it.

11.2. Relaxed Read Ordering

PCI 2.2 does not specify any ordering requirements for multiple read completions traveling in the same direction, so any device supporting more than one outstanding read request must be prepared to receive the respective completions in any order. Relaxed read ordering instead applies to the conventional PCI requirement for read completions not to pass memory writes traveling in the same direction. When one device (e.g., the CPU) is preparing new work for another, it typically writes a block of data (the payload) into a memory buffer, and then writes a control structure at another location. The first device (the CPU) generally does not change the data again after the control structure is written. At a minimum, the CPU guarantees that the data location is consistent with the control location at the time the control location is written. After the device discovers the new control structure, it reads the data buffer and begins its operation. If the control structure is read with strict ordering (Relaxed Ordering attribute not set), it pushes or pulls all the data writes to their final destinations in the same manner as required by PCI 2.2. Once the new control structure has been read by the device, the device can read the data buffer with the Relaxed Ordering bit set, thereby preventing the reads from being unnecessarily blocked behind other unrelated write transactions.

Relaxed read ordering can be of significant benefit in systems with a large number of memory write transactions addressing targets that delay the completion of those writes up to the maximum described in Section 2.13. Memory writes to these slow devices block progress of all read completions moving in the same direction, because the system hardware cannot distinguish between related and unrelated transactions. The often-cited case is that of CPU memory write transactions addressing a graphics device introducing lengthy stalls in other devices' main-memory read completions. But this is certainly not the only situation in which relaxed reads are desirable. Any device that requires a significant number of memory write transactions from a CPU potentially introduces unnecessary performance degradations on other devices. (The I_2O "push" messaging model is an example). If those other devices set the Relaxed Ordering attribute bit for their payload reads (indicating that they are not related to any write transaction that may be in progress), those reads are allowed to pass the congested memory write transactions.

11.3. Co-location of Payload and Control

Programming models that require the pay load and control (or Data and Flag) to be co-located (that is, located on the same side of all bridges in the system) are permitted to set the Relaxed Ordering attribute bit when reading the control locations as well as the payload. If there are no bridges between the payload and control locations, there are never any write transactions addressing the payload that the read of the control locations must flush. Setting the Relaxed Ordering attribute bit for control-location read transactions enables the corresponding read completions to pass unrelated congested memory writes that would otherwise block control reads.

Co-location of payload and control generally does not enable the device to set the Relaxed Ordering attribute bit for write transactions. If writes to the payload space must finish at the completer before write to the control space, the requester must not set the Relaxed Ordering attribute on writes to the control space.

11.4. Other Uses of Relaxed Ordering

Devices are permitted to set the Relaxed Ordering attribute bit on any transaction for which the programming model of the device guarantees that the system hardware does not need memory writes to be kept in order with respect to each other and Split Read Completions are allowed to pass memory write transactions moving in the same direction. It is possible for devices to expand the use of the Relaxed Ordering attribute beyond those described above by using ordered transactions only on a carefully selected subset of control transactions or through the use of explicit information passed by the device driver. It is also possible to use relaxed ordering on transactions initiated by the host bridge, if the system provides a method for a CPU to specify its ordering requirements. (This might allow more timely completion of a CPU generated read of an I/O register in the midst of a flood of device-to-memory write traffic.) These types of uses may be of little benefit to many systems but are valuable for specialized applications.

In all cases, no requester is allowed to set the Relaxed Ordering attribute bit if the Enable Relaxed Ordering bit in the PCI-X Command register is cleared.

11.5. I_2O Usage Models

The introduction of the I_2O specification for intelligent peripherals has made certain system topologies more likely to be used. Three common implementations of I_2O-based peripherals are presented: the Push Model, the Pull Model, and the Outbound Option.

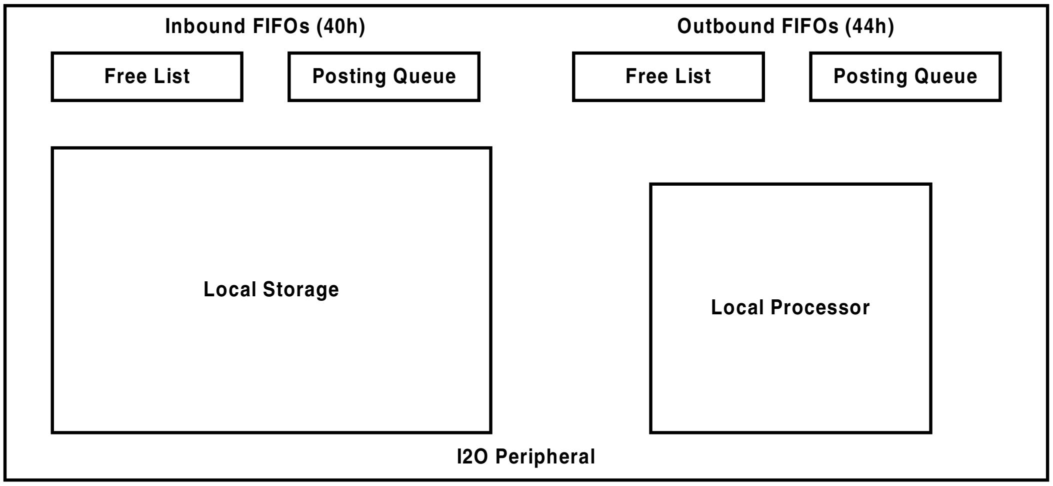


Figure 11-1: I_2O Standard Components

Every I_2O device implements a standard programming interface shown in Figure 11-1. When the system is initialized, memory locations are reserved as buffers for messages moving both to (inbound) and from (outbound) the I/O platform (IOP). Pointers to these message buffers are managed by various free lists and posting queues contained either in the IOP or in host memory depending upon the messaging protocol model in use.

The original I_2O messaging protocol model (the “Push Model”) places all free lists and posting queues in the IOP, allocates all message buffers at the receiver, and relies on the sender always pushing (writing) message buffer contents towards the receiver destination.

A different usage model called the “Pull Capability” allows message buffers used for host-to-IOP communication to reside in host memory and be pulled (read) from host memory by the IOP when needed. In addition, the free list for these host-resident inbound message buffers is also kept in host memory.

Another usage model called the “Outbound Option” allows the posting queue for outbound message buffers to reside in host memory. (Outbound message buffers themselves reside in host memory for all of these models and are used strictly for IOP–to-host communication.)

11.5.1. I_2O Messaging Protocol Operation

For the CPU to send a message to the IOP, the CPU acquires a message buffer by reading an entry from the Inbound Free List. The value either points to the next available buffer or indicates that no buffers are available. If a buffer is available, the CPU fills it with the message and writes the value of the pointer to the Inbound Posting Queue, which notifies the local processor that new work has arrived. The local processor reads the message from the buffer and processes it. When the local processor finishes using the buffer, it returns the buffer pointer to the Inbound Free List.

For the local processor to send a message to the CPU, the local processor acquires a buffer pointer from the Outbound Free List. If a buffer is available, the local processor fills it with the message and writes the value of the pointer to the Outbound Posting Queue. This operation generates an interrupt to the CPU. The CPU then reads the pointer to the buffer from the Outbound Posting Queue and begins work on the buffer. When the CPU finishes using the buffer, it returns it to the Outbound Free List.

The actual location of the Inbound Free List and Outbound Posting Queue vary according to the protocol option in use, but their logical operation remains the same.

11.5.2. Message Delivery with the Push Model

The original I_2O messaging usage model is called the “push model” because data for both inbound and outbound messages are pushed (written) to the destination. See Figure 11-2.

The CPU “pushes” both the message data (payload) and the Inbound Posting Queue (control) for inbound messages under the push model. If the system provides a method for the CPU to designate which transactions are message data and which are writes to the Inbound Posting Queue, the host bridge is permitted to set the Relaxed Ordering attribute bit on writes of the message data.

The local processor “pushes” the message data (to main memory) and then writes to the Outbound Posting Queue (a local hardware register) to interrupt the processor. Since the trigger method in this case uses a sideband path (an interrupt to the CPU) rather than the bus, the two events must be synchronized. Two alternatives are commonly used in PCI systems to synchronize the two events. The first alternative is for the IOP to read back some of the message data to guarantee that it is delivered all the way to main memory before interrupting the CPU. The second alternative is for the CPU to read from the device as part of the interrupt service routine to flush data in transit. In both cases, the IOP is permitted to set the Relaxed Ordering attribute on the writes of the message data.

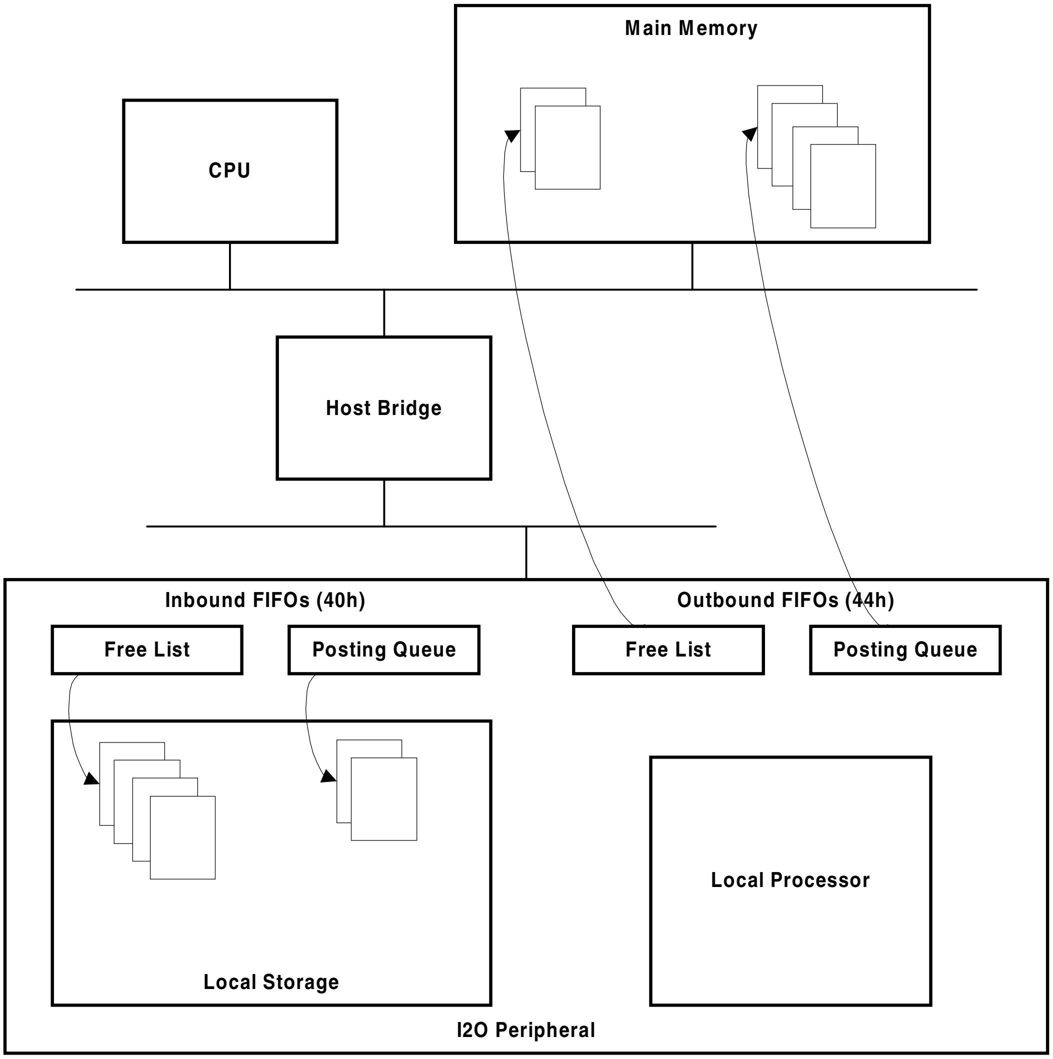


Figure 11-2: I_2O Push Model

11.5.3. Message Delivery with the Pull Model

With the pull model inbound message buffers are placed in host memory and the local processor pulls (reads) message data from them. In addition, the Free List is likewise placed in host memory. Outbound messaging is not affected by use of the pull model (same as push model). See Figure 11-3.

Under the pull model, the IOP is permitted to set the Relaxed Ordering attribute when reading the message data. This allows the host bridge (and other intervening bridges) to avoid unnecessary blocking of the Split Read Completion transactions (containing the message data) by unrelated memory write transactions moving in the same direction.

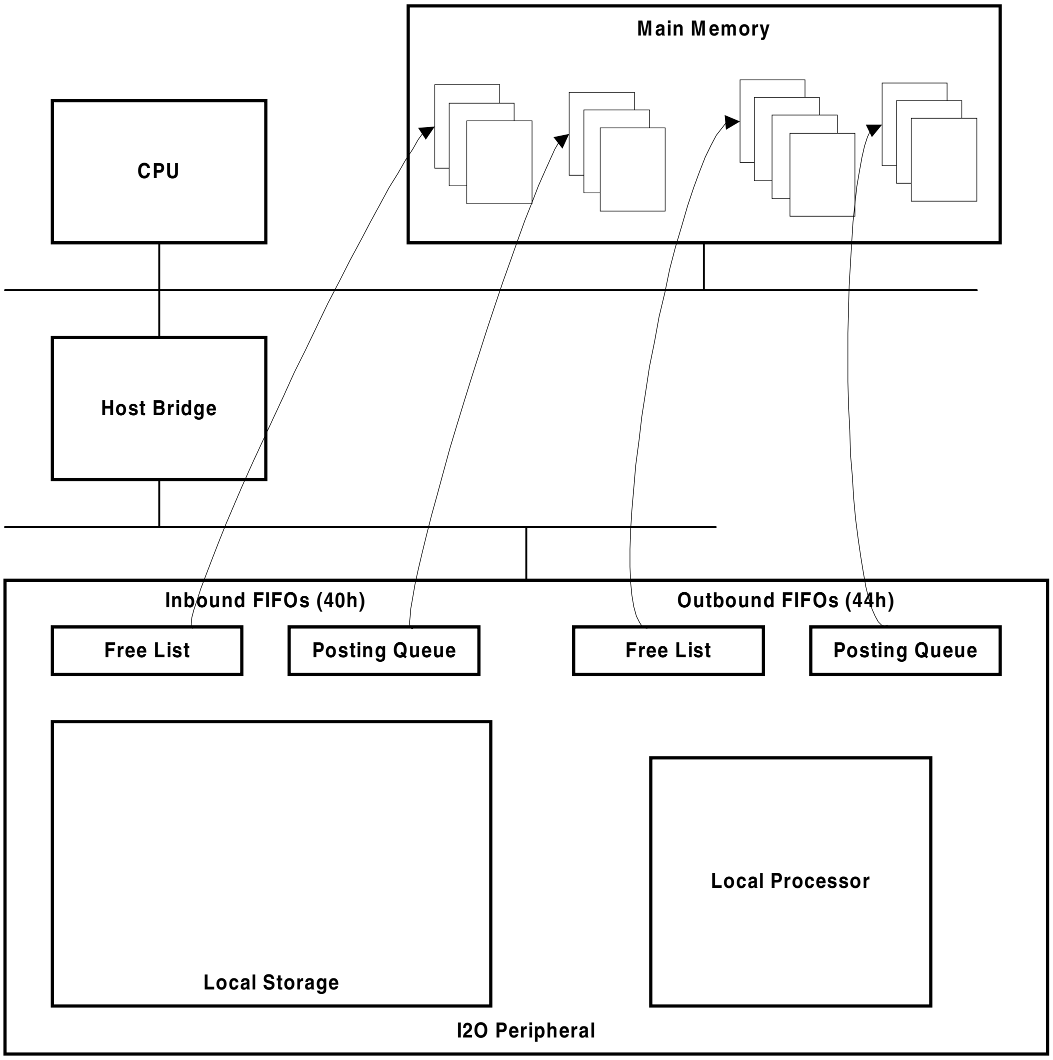


Figure 11-3: I_2O Pull Model

11.5.4. Message Delivery with the Outbound Option

With the Outbound Option, the Outbound Posting Queue is placed in host memory using software rather than hardware to manage the queue. Inbound messaging is not affected by use of the Outbound Option feature.

With the Outbound Option, the IOP is permitted to set the Relaxed Ordering attribute bit on writes of the message data and must not set the bit on writes to the Outbound Posting Queue. The strictly ordered writes to the Outbound Posting Queue push the message data ahead of them.

11.5.5. Message Delivery with Peer to Peer

As the I_2O specification is expanded to make peer-to-peer operations practical, many messages move directly from one IOP to another rather than between an IOP and main memory.

Messages between IOPs are always handled as inbound messages with respect to the destination IOP and follow the original I_2O messaging protocol for inbound messages.

The writing IOP is permitted to set the Relaxed Ordering attribute bit on writes of the message data and must not set the bit on writes to the Inbound Posting Queue. The strictly ordered writes to the Inbound Posting Queue push the message data ahead of them.

12. Appendix—Minimal PCI Power Management Support

If a device has no need for allowing its power to be managed, PCI PM 1.1 specifies the minimal support required to enable software to determine the power management capability of the device and to allow it to operate in an environment that manages the power of other devices. The following list is a summary of the minimum hardware requirements for PCI power management. This information is provided for reference purposes only. Refer to PCI PM 1.1 for complete information. In case of conflict, PCI PM 1.1 takes precedence.

1. **Implementation of the Capabilities List Data Structure – Note that in a minimal implementation these may all be Read-Only bits**
    - *PCI Status* register modified such that bit(4) set to "1b" indicating presence of Capability List
    - *Cap_Ptr* register (PCI Config Header offset 34h, indicating offset to PCI-PM register file or another Capabilities List item)
    - *Cap_ID* register (8 bit register actually residing within the PCI-PM register file)
    - *Next_Item_Ptr* register (8 bit register actually residing within the PCI-PM register file)

2. **Implementation of the PCI-Power Management register file**

• *PMC*	Power Management Capabilities Register (16 bit Read-Only register)
• *PMCSR*	Power Management Control/Status Register (16 bit register–bits 1, 0 (Power State) Read-Write)
• *PMCSR-BSE*	P2P Bridge Register (Non-0 for P2P bridges only) (8 bit Read-Only register)
• *DATA*	Power Data Register (Minimally read all 0s) (8 bit Read-Only register)

3. Implementation of Power Management States

- *D0* – Represents a fully configured and operational PCI function. (All devices automatically support this state.)
- *D3*$_{cold}$ – Device has no power applied. (All devices automatically support this state.)
- *D3*$_{hot}$ – Device is in a state in which it can be restored without a full boot sequence. While in this state, the device is ready to have its clocks stopped and its power removed. (This state enables software to effectively single out an idle PCI function and selectively shut it "off" via program control enabling power savings even when the system is otherwise in use.)

  D3$_{hot}$ is the only new device power managed state (D-State) from a legacy PCI perspective. Implementers should note the following restrictions on devices in *D3*$_{hot}$ (from Sections 5.4 and 5.6 of PCI PM 1.1):

  – Device must respond to Configuration cycles.

  – Device must **NOT** respond to I/O or Memory cycles.

  – Device must **NOT** generate interrupts.

  – Device must **NOT** initiate PCI transactions other than Split Completions.

  – Device must "perform the equivalent of a warm (soft) reset internally" when programmed to *D0*.

4. The following pins are required only by devices needing to wake the system.

- **3.3Vaux** – Provides standby power to a powered down device
- **PME#** – Used by a power-managed device to request attention

3.3Vaux and **PME#** are only required by a device (like a LAN controller or modem) that needs to wake the system because of some external stimulus; e.g., received a packet from a network or a ring to a modem. Refer to PCI PM 1.1 for more details.

13. Appendix—Setting Performance Registers

13.1. Setting the Maximum Memory Read Byte Count Register

The Maximum Memory Read Byte Count register sets the maximum byte count a PCI-X device uses when initiating a Sequence with one of the burst memory read commands (see Section 7.2.3). System configuration software is permitted to use any algorithm for selecting the setting of the Maximum Memory Read Byte Count register that best meets the requirements of the system. The default register value (512 bytes) is effective for bus segments that share resources such as host or PCI-X bridge buffers with other devices. Some devices that don't share the source bridge with other devices are more efficient by using larger byte counts.

Systems that support PCI hot-plug optionally adjust the settings for all devices on the bus after each hot-plug operation or leave the registers in a state that is satisfactory for all possible configurations of full and empty slots.

13.2. Optimizing the Split Transaction Commitment Limits in PCI-X Bridges

PCI-X bridges include upstream and downstream Split Transaction Commitment Limit registers to control the maximum cumulative size of all transactions forwarded by the bridge (see Sections 8.4.2.1, 8.6.2.5, and 8.6.2.6). The optimum settings for these registers are a function of the number and behavior of devices that share bus segments with transactions that cross the bridge. If the commitment limit is set too high, Split Completion data returns to the PCI-X bridge faster than it can be forwarded to the requester causing Split Completions to back up toward the completer. If the commitment limit is set too low, Split Requests are delayed unnecessarily, and requesters on one side of the bridge experience additional latency when reading from completers on the other side. The Split Request Delayed and Split Completion Overrun bits in the PCI-X Bridge Status register (see Section 8.6.2.4) and Secondary Status register (see Section 8.6.2.3) are available to help determine the optimum setting of the Split Transaction Commitment Limit register.

Devices with a single PCI-X bridge between them and the host bridge generally experience less latency when reading from main memory than devices that must cross more PCI-X bridges to reach the host bridge. If a bridge whose primary bus is connected directly to the host bridge has a Split Transaction Capacity of at least 4 Kbytes, and the associated host bridge has a typical read latency of 3 µs or less, setting the upstream Split Transaction Commitment Limit register equal to the upstream Split Transaction Capacity register generally allows requests to be forwarded upstream as quickly as necessary without any risk of terminating a Split Completion with Retry.

The read latency for downstream transactions and for transactions that cross multiple PCI-X bridges is generally harder to predict. One method for identifying the optimum setting for the Split Transaction Commitment Limit register in these cases is to adjust it based on the behavior of previous traffic as indicated by the Split Request Delayed and Split Completion Overrun status bits. The general guideline for setting the commitment limit is that if the Split Request Delayed bit is set, the limit is *potentially* too low. If the Split Completion Overrun bit is set, the limit *is* too high. The optimum setting of the Split Transaction Commitment Limit register is one less than the smallest setting for which the Split Completion Overrun bit sets. (There is always room to store Split Completion data up to the capacity of the bridge, so there is never a need to set the limit less than the capacity of the bridge.) If bus traffic is heavy on the requester side, or if the requester-side bus width or frequency is less than the completer side, the Split Request Delayed bit may set even though the Split Transaction Commitment Limit register is set optimally.

The following outline shows an example of a continuously running optimization routine that adjusts the setting of this register:

1. The system powers up with the Split Transaction Commitment Limit register set equal to the Split Transaction Capacity register. The Split Completion Overrun bit never sets when the limit is equal to the bridge capacity.
2. Wait an appropriate time interval.
3. Check the Split Request Delayed bit and the Split Completion Overrun bit and adjust the Split Transaction Commitment Limit register as follows:

   If neither bit is set, the commitment limit value is good.

   If the Split Request Delayed bit is set and the Split Completion Overrun bit is *not* set, the commitment limit is too low. Increase the limit.

   If the Split Request Delayed bit is *not* set and the Split Completion Overrun bit is set, the limit is too high. Decrease the limit.

   If both bits are set, the limit is too high. Decrease the limit.
4. Go to step 2.

The appropriate time interval for algorithms such as this depends upon the rate at which traffic patterns in the system change.

If traffic patterns change over time, an algorithm such as the one described above tracks those changes and adjusts the Split Transaction Commitment Limit appropriately. More sophisticated algorithms that adapt to historical traffic patterns, or use varying change increments for the register, or varying delay-time intervals are also possible.

14. Appendix—Detection of PCI-X Add-in Card Capability

As described in Section 9.10, add-in cards indicate their PCI-X mode and frequency capabilities by the way they connect the **PCIXCAP** pin. PCI-X 133 cards leave the pin unconnected (except for a decoupling capacitor). PCI-X 66 cards connect the pin to ground through a 10 kΩ resistor (plus the decoupling capacitor). Conventional PCI cards connect the pin to ground.

The system board provides a circuit to detect the three possible states of **PCIXCAP**, Vcc (PCI-X 133), ground (conventional PCI), or in between (PCI-X 66). Two alternative approaches for this circuit are presented in this section. The first approach uses a single pull-up resistor on the system board and multiple voltage comparators to detect the three states. The second alternative uses programmable pull-up resistor values and a standard binary input buffer.

14.1. Three-level Comparator

The three-level comparator alternative is recommended for systems that need a static indication of the state of **PCIXCAP** and can accommodate non-standard input buffers or external comparators.

Figure 14-1 shows a three-level comparator circuit for detecting the state of **PCIXCAP**. The circuit shows two PCI-X 66 cards installed in slots. This system board connects **PCIXCAP** for both slots together in a non-hot-plug system. (Hot-plug systems must determine the state of PCIXCAP for each slot independently.) Alternatives that connect more or fewer slots to the circuit or provide multiple copies of the comparator circuit are also possible. Most systems with more than two slots are too large to operate above 66 MHz, so they have no need for hardware differentiation between PCI-X 133 and PCI-X 66 add-in cards.

The value of the pull-up resistor ("R Pullup" in the figure) determines the range of voltages that occur on **PCIXCAP** for different numbers of PCI-X 66 cards. Figure 14-2 shows two such ranges. The first case uses a pull-up resistor value of 10 kΩ and produces a range of voltages on **PCIXCAP** similar to standard 3.3 signaling voltages. This alternative is preferred if the comparator has good noise margin close to ground but less margin as the input approaches Vcc. The second case uses a 5.1 kΩ pull-up resistor and produces a higher range of voltages. The second alternative is preferred if the comparator has good noise margin over the full range of input voltages from ground to Vcc.

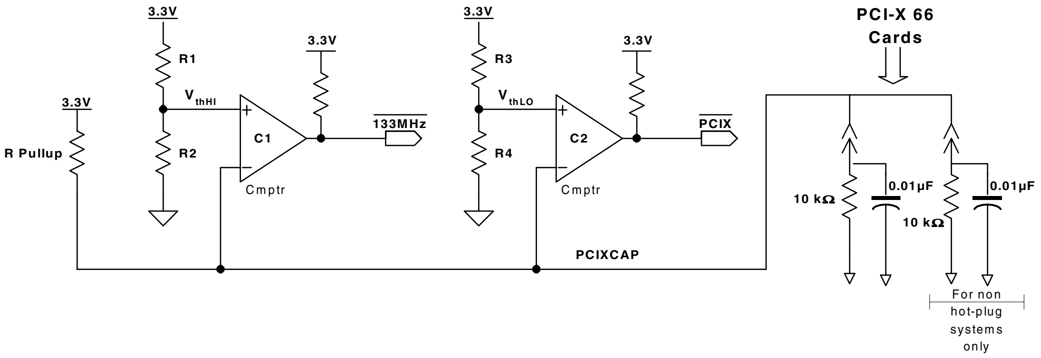


Figure 14-1: Three-Level Comparator Circuit

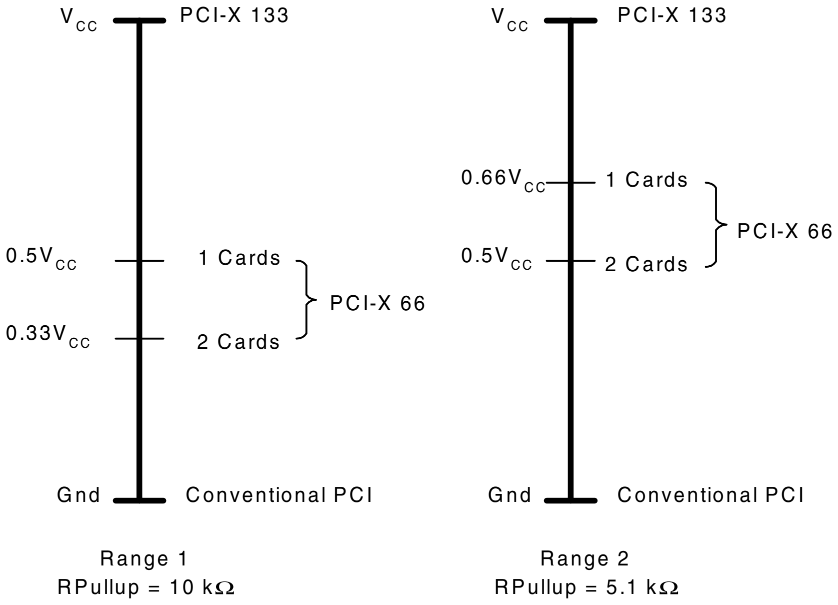


Figure 14-2: Three-Level Comparator Voltage Ranges

Table 14-1 shows values for the resistors that provide the comparator reference voltages and the corresponding reference voltages. Values are calculated assuming negligible input offset and bias currents in the comparitors.

Table 14-1: Comparator Resistors and Thresholds

Parameter	Range 1	Range 2	Units
R Pullup	10	5.1	kΩ
R1	5.6	2	kΩ
R2	10	10	kΩ
R3	10	10	kΩ
R4	2.2	2	kΩ
V_{thHI}	2.12	2.75	V
V_{thLO}	0.60	0.55	V

14.2. Programmable Pull-up and Binary Input Buffer

The programmable pull-up alternative is recommended for systems that require standard input buffers. Systems that support PCI hot-plug slots sometimes fall into this category. Hot-Plug Controllers often require many pins to support hot-plug operations on multiple slots. High pin count often leads to implementations that wotk best if all inputs use standard binary input thresholds.

Figure 14-3 shows a circuit for a programmable pull-up resistor. The weak pull-up resistor value is selected to set the voltage of **PCIXCAP** below the low level of a standard binary input buffer if at least one slot contains a PCI-X 66 card. The strong pull-up resistor value is selected to set the voltage of **PCIXCAP** above the high level of a standard binary input buffer if all slots contain PCI-X 66 cards (no conventional card connecting **PCIXCAP** to ground). Logic controls the buffer that drives the strong pull-up resistor. The system must provide sufficient delay after switching the pull-up resistor for the decoupling capacitors to charge or discharge before reading the state of the **PCIXCAP** pin.

Figure 14-4 shows the range of voltages on **PCIXCAP** for the two pull-up resistor values and one or two PCI-X 66 add-in cards.

Figure 14-3 shows **PCIXCAP** for both slots connected together. A PCI hot-plug system must provide a means to detect the states of **PCIXCAP** for each slot separately.

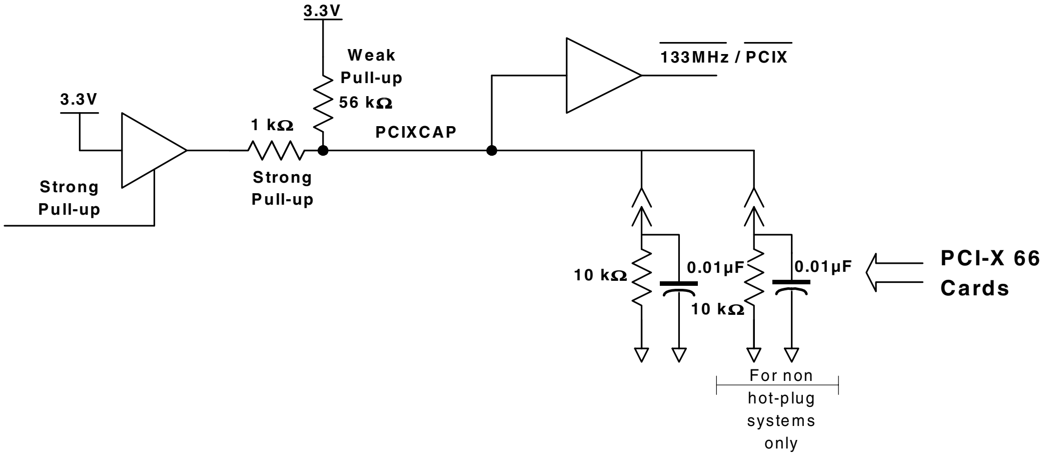


Figure 14-3: Programmable Pull-Up Circuit

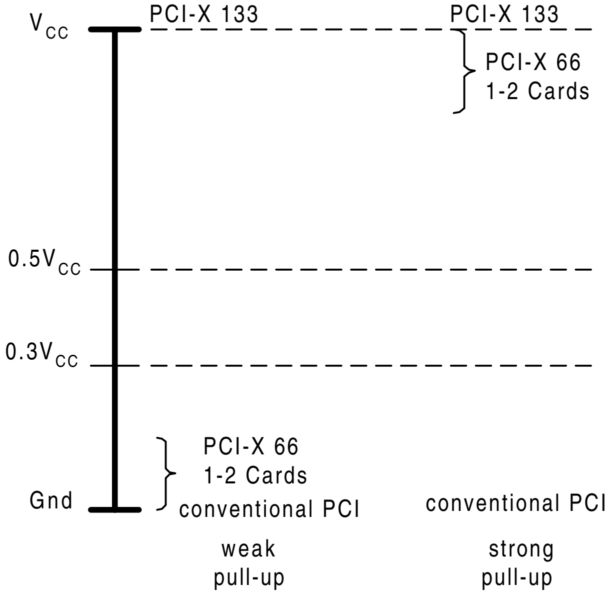


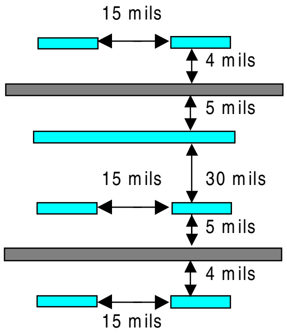
Figure 14-4: Threshold Ranges with Programmable Pull-Up

15. Appendix—Add-In Card Multilayer Board Spacing and Stack-Up Examples

15.1. Six-Layer-Board Examples

Figure 15-1 and Figure 15-2 show example six-layer board stack-ups that meet the PCI-X requirements.

Figure 15-1 shows four signal layers and two power plane layers (one for Vcc and one for ground). Layers 3 and 4 are dual-microstrips; that is, they reference the same two power planes. All four signal layers in this stack-up are suitable for routing PCI-X signals. Their single-line impedances meet the requirements of Section 9.13.4, and the trace geometry and spacing meet the crosstalk requirements shown in Section 9.13.3.



Layers	Trace width	Cu weight	Zo	Trace delay
L1 Signal	5 mils	0.5 oz. Cu., 1.0 oz. finish	57 ohm	165 ps/in.
L2 GND		1.0 oz.		
L3 Signal	5 mils	0.5 oz.	57 ohm	185 ps/in
L4 Signal	5 mils	0.5 oz.	57 ohm	185 ps/in
L5 Power		1.0 oz.		
L6 Signal	5 mils	0.5 oz. Cu., 1.0 oz. finish	57 ohm	165 ps/in.

Figure 15-1: Example Six-Layer Stack-Up Using Stripline and Dual Microstrip

The following assumptions are included in Figure 15-1:

1. Internal dielectrics are standard FR4 with relative dielectric constant of 4.5.
2. External layers are covered with a 2-mil thick solder mask with relative dielectric constant of 3.0.
3. Signal trace cross sections are trapezoidal. Trace widths listed are the base of the trapezoid. The top of the trapezoid is 1 mil smaller. The minimum base-to-base trace separations are shown. For all signal layers, the base of the trapezoid is oriented toward the nearest power plane.

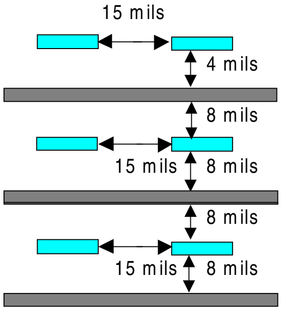
Note that the starting thickness of the traces on top and bottom (layers 1 and 6) is 0.5 oz copper with final thickness after plating of 1 oz. Starting with 1.0 oz copper and plating to 1.5 oz does *not* meet the requirement for characteristic impedance and crosstalk. Using 1.5 oz finished dimensions lowers the nominal characteristic impedance to 52 ohms and increases the cumulative crosstalk to 5.3%.

The minimum edge-to-edge spacing shown in Figure 15-1 assumes the dual-microstrip layers (layers 3 and 4) are routed orthogonally. If traces on layers 3 and 4 run parallel, the minimum edge-to-edge spacing between signals within each layer must increase to keep the cumulative crosstalk within the specified limit. Table 15-1 shows the minimum edge-to-edge spacing for adjacent signals on layer 3 and adjacent signals on layer 4 if signals on layer 3 and 4 are routed parallel to each other for the length shown in the table and have the same horizontal location in the cross section (i.e., traces on layer 3 and 4 are routed directly opposite each other).

Table 15-1: Trace Spacing for Parallel Traces on Dual Microstrip

Minimum Spacing (between signals on same layer)	Maximum Parallel Length (between Layer 3 and 4)
15 mils	0
17 mils	0.38 inches
18 mils	0.59 inches
20 mils	0.97 inches

Figure 15-2 shows an alternate stack-up with three signals layers and three power plane layers (for Vcc and ground). All three signal layers are suitable for routing PCI-X signals (single-line impedances meet the requirements of Section 9.13.4, and the trace geometry and spacing meet the crosstalk requirements shown in Section 9.13.3). There are no orthogonal routing requirements for this stack-up, since all signal layers are isolated by power planes.



Layers	Trace width	Cu weight	Zo	Trace delay
L1 Signal	5 mils	0.5 oz. Cu., 1.0 oz. finish	57 ohm	165 ps/in.
L2 Power		1.0 oz.		
L3 Signal	5 mils	0.5 oz.	57 ohm	185 ps/in
L4 GND		1.0 oz.		
L5 Signal	5 mils	0.5 oz.	57 ohm	185 ps/in
L6 GND		1.0 oz.		

Figure 15-2: Example Six-Layer Stack-Up Using Stripline and Single Microstrip

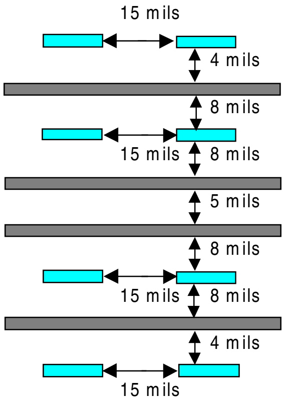
The following assumptions are included in Figure 15-2:

1. Internal dielectrics are standard FR4 with relative dielectric constant of 4.5.
2. External layers are covered with a 2-mil thick solder mask with relative dielectric constant of 3.0.
3. Signal trace cross sections are trapezoidal. Trace widths listed are the base of the trapezoid. The top of the trapezoid is 1 mil smaller. The minimum base-to-base trace separations are shown. For layer 1, the base of the trapezoid is oriented toward the power plane. Layers 3 and 5 are symmetric and, therefore, are allowed to be oriented in either direction.

Note that the starting thickness of the traces on top and bottom (layers 1 and 6) is 0.5 oz copper with final thickness after plating of 1 oz for the same reason as shown above for Figure 15-1.

15.2. Eight-Layer-Board Examples

Figure 15-3 shows an example eight-layer board stack-up that meets the PCI-X requirements. It shows four signal layers and four power plane layers (for Vcc and ground). All four signal layers in this stack-up are suitable for routing PCI-X signals. Their single-line impedances meet the requirements of Section 9.13.4, and the trace geometry and spacing meet the crosstalk requirements shown in Section 9.13.3.



Layers	Trace width	Cu weight	Zo	Trace delay
L1 Signal	5 mils	0.5 oz. Cu. 1.0 oz. finish	57 ohm	165 ps/in.
L2 Power		1.0 oz.		
L3 Signal	5 mils	0.5 oz.	57 ohm	185 ps/in
L4 GND		1.0 oz.		
L5 Power		1.0 oz.		
L6 Signal	5 mils	0.5 oz.	57 ohm	185 ps/in
L7 GND		1.0 oz.		
L8 Signal	5 mils	0.5 oz. Cu. 1.0 oz. finish	57 ohm	165 ps/in.

Figure 15-3: Example Eight-Layer Stack-up

The following assumptions are included in Figure 15-3:

1. Internal dielectrics are standard FR4 with relative dielectric constant of 4.5.
2. External layers are covered with a 2-mil thick solder mask with relative dielectric constant of 3.0.
3. Signal trace cross sections are trapezoidal. Trace widths listed are the base of the trapezoid. The top of the trapezoid is 1 mil smaller. The minimum base-to-base trace separations are shown. For layers 1 and 8, the base of the trapezoid is oriented toward the power plane. Layers 3 and 6 are symmetric and, therefore, are allowed to be oriented in either direction.

Note that the starting thickness of the traces on top and bottom (layers 1 and 8) is 0.5 oz copper with final thickness after plating of 1 oz for the same reason as shown above for Figure 15-1.